Machine Learning in Portfolio Management

STEVEN Y. K. WONG

BE (Hons), BCom, MFin

Supervisor: A/Prof. Jennifer S. K. Chan
Associate Supervisor: Dr. Lamiae Azizi

A thesis submitted in fulfilment of
the requirements for the degree of
Doctor of Philosophy

School of Mathematics and Statistics
Faculty of Science
The University of Sydney
Australia

19 September 2023

Abstract

Financial markets are difficult learning environments. The data generation process is time-varying, returns exhibit heavy tails and signal-to-noise ratio tends to be low. These contribute to the challenge of applying sophisticated, high capacity learning models in financial markets. Driven by recent advances of deep learning in other fields, we focus on applying deep learning in a portfolio management context. This thesis contains three distinct but related contributions to literature. First, we consider the problem of neural network training in a time-varying context. Conventional batch training methods assume a stationary data generation process, where training and out-of-sample data are assumed to be drawn from the same distribution. This is suboptimal for applications where the data generation process changes over time, such as in financial markets. To address this, we extend the *early stopping* algorithm into the online context, which we term the Online Early Stopping algorithm. We show that a neural network trained using this algorithm can track a function changing with unknown dynamics. We provide a *regret-bound* for the algorithm and show that the worst-case tracking performance of the algorithm is bound by the time-variance of the data generation process. We compare the proposed algorithm to current approaches in predicting monthly U.S. stock returns and show its superiority. Second, we consider the problem of learning in noisy environments. Noisy learning environments such as financial markets are characterised by low noise-to-signal ratio. This differs to information-rich applications such as image recognition. We propose an approach that regularises the temporal convolutional network using a supervised autoencoder, which we term the Supervised Temporal Autoencoder. We show that the addition of the auxiliary reconstruction task is beneficial to the primary supervised learning task in the context of stock return time-series forecasting. The supervised autoencoder denoises the input and encourages the main network to retain features that are beneficial to both prediction and reconstruction tasks. We also show that the supervised temporal autoencoder is able to learn features directly from the transformed price series, alleviating the need for

handcrafted features. The autoencoder also improves interpretability as users can observe the output of the decoder and inspect features retained by the network. Third, we consider the problem of quantifying forecast uncertainty in time-series with complex structures. Time-varying variance, such as volatility clustering as seen in financial time-series, can lead to large mismatch between predicted uncertainty and realised forecast error. We propose a novel framework to deal with uncertainty quantification under the presence of volatility clustering, building and extending the recent methodological advances in uncertainty quantification for non-time-series data. We outline several methodological advancements, including the use of scale mixture distribution and separate modelling of distribution hyperparameters. To illustrate the performance of our proposed approach, we apply it onto cryptocurrency and U.S. equities time-series forecasting for the designed use-case. We demonstrate superior performance to the current state-of-the-art in both data sets. We further provide an evaluation using a non-time-series benchmark data set (Appendix) to show the general applicability of our framework. Finally, potential future research directions in advancing machine learning in portfolio management is discussed.

Acknowledgements

After 5 arduous years, here I am, putting the final touches to my thesis. Looking back, I am glad that I have spent 5 years of my life (albeit part time) to learn something new about machine learning, and to give back my knowledge to science, however trivial my contributions may be. I would like to start by thanking my supervisors, A/Prof. Jennifer Chan and Dr. Lamiae Azizi, for whom I am forever grateful to have been mentored by. Without their patience, support and knowledge, I would not have made it this far. I would like to thank Prof. Richard Xu for initially accepting me into his PhD cohort. Even though we had to part ways, his machine learning classes were immensely helpful to my PhD. A special thanks to Prof. Maurice Pagnucco for supervising my Honours and leading the UNSW RoboCup team — an unforgetable journey through artificial intelligence that I still cherish today. I would also like to thank the many past and present colleagues who have shaped my understanding of quantitative investing. All of these experiences culminated in this thesis.

Finally, I would like to thank my family for their support and patience. They are the reason for my perseverance.

Contents

List of Figures

List of Tables

Acronyms

APT: Arbitrage Pricing Theory

ARCH: Autoregressive Conditional Heteroskedasticity

ARMA: autoregressive-moving-average

BNN: Bayesian neural network

CAPM: Capital Asset Pricing Model

CNN: convolutional neural network

CRSP: Center for Research in Security Prices

DGP: data generation process

DTS-SGD: Dynamic Exponentially Time-Smoothed Stochastic Gradient Descent

ELU: exponential linear unit

EWNN: expanding window neural network

GARCH: Generalised Autoregressive Conditional Heteroskedasticity

IC: information coefficient

IG: Inverse-Normal

KL divergence: Kullback-Leibler divergence

LSTM: long short-term memory

MCMC: Monte Carlo Markov Chain

MLP: multilayer perceptrons

MOM1: reversal effect

MOM12: momentum effect

MPT: Modern Portfolio Theory

MSE: mean squared error

MTL: multi-task learning

N-BEATS: Neural Basis Expansion Analysis for interpretable Time Series

NG: Normal-Gamma

NIG: Normal-Inverse-Gamma

NLL: negative log-likelihood

NLP: natural language processing

OES: Online Early Stopping

OLS: ordinary least squares

PCA: principal component analysis

PTVII: Pearson Type VII

ReLU: rectified linear unit

RMSE: root mean squared error

RNN: recurrent neural network

SAE: supervised autoencoder

SGD: stochastic gradient descent

SIC: Standard Industrial Classification code

SMD: scale mixture distribution

SML: Security Market Line

STAE: Supervised Temporal Autoencoder

SVM: Support Vector Machine

TCN: temporal convolutional network

TRI: total return index

UCI: University of California Irvine Machine Learning Repository

UQ: uncertainty quantification

List of Notations

Symbol	Description
$\boldsymbol{X}$	Uppercase bold font denotes matrix
$\boldsymbol{x}$	Lowercase bold font denotes vector
$\boldsymbol{x}_i$	i-th row of $\boldsymbol{X}$
$x_{i,j}$	Element in the i-th row and j-th column of $\boldsymbol{X}$
x_i	i-th element of $\boldsymbol{x}$
$(\boldsymbol{X}, \boldsymbol{y}) \sim \mathcal{D}$	Input $\boldsymbol{X}$ and output $\boldsymbol{y}$ (or $\boldsymbol{r}$) drawn from dataset $\mathcal{D}$
p	Probability density function
y	Dependent variable of a regression (e.g., forward returns)
r	Contemporaneous returns ($t-1$ to t)
F	Model (e.g., neural network)
f	Activation function or a single network layer
ξ	Factor return
B	Batch size
b	b-th batch
M	Number of features or independent variables
N	Number of stocks
T	Number of periods
L	Number of layers in a neural network
ℓ	ℓ-th layer
$\boldsymbol{W}^{(\ell)}$	Network weights of layer ℓ
$\boldsymbol{b}^{(\ell)}$	Network bias of layer ℓ
$\boldsymbol{\theta}^{(\ell)}$	$(\boldsymbol{W}^{(\ell)}, \boldsymbol{b}^{(\ell)})$ weight set of layer ℓ
$\boldsymbol{\theta}$	$\bigcup_{\ell=1}^{L} \boldsymbol{\theta}^{(\ell)}$ (all weight sets of neural network)

Symbol	Description
$\boldsymbol{a}$	Activation values
τ	Number of optimisation epochs
k	Kernel size (of convolutional layer)
K	Sequence length
$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})$	Loss between true $\boldsymbol{y}$ and predicted $\hat{\boldsymbol{y}}$
$J(\boldsymbol{\theta})$	Abbreviation for $\mathcal{L}(F(\boldsymbol{X}; \boldsymbol{\theta}), \boldsymbol{y})$
$\hat{\nabla} J(\boldsymbol{\theta})$	Stochastic gradient of $J(\boldsymbol{\theta})$
η	Learning rate

Author attribution statement

The proceeding thesis (with publications) is based on the following two published papers and one working paper:

(1) **Steven Y. K. Wong**, Jennifer S. K. Chan, Lamiae Azizi, Richard Y. D. Xu, "Time-varying neural network for stock return prediction," *Intelligent Systems in Accounting, Finance and Management*, 29(1), 3–18, 2022. This work is presented in Chapter 3.

(2) **Steven Y. K. Wong**, Jennifer S. K. Chan, Lamiae Azizi, Richard Y. D. Xu, "Supervised Temporal Autoencoder for Stock Return Time-series Forecasting," *Proceedings of the IEEE 45th Annual Computer Software and Applications Conference*, Madrid, Spain, 2021. This work is presented in Chapter 4.

(3) **Steven Y. K. Wong**, Jennifer S. K. Chan, Lamiae Azizi, "Quantifying neural network uncertainty under volatility clustering," *working paper*, 2022. This work is presented in Chapter 5.

Steven Wong was responsible for developing new methods, implementation, and writing the 3 papers. Steven Wong was responsible for at least 50% of the contribution, and was the first and corresponding author in all aforementioned papers.

Declaration

This is to certify that to the best of my knowledge, the content of this thesis is my own work. This thesis has not been submitted for any degree or other purposes, except where specified for publication.

I certify that the intellectual content of this thesis is the product of my own work, except where acknowledged with others, and that all the assistance received in preparing this thesis and sources have been acknowledged.

Signed: STEVEN Y. K. WONG

1st February 2023

As supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements are correct.

Signed: JENNIFER S. K. CHAN

1st February 2023

CHAPTER 1

Introduction

Forecasting in financial markets is one of the most difficult problems in machine learning. The prediction problem is *time-varying* and plagued by *low signal-to-noise* in the data. It is markedly different to traditional applications of machine learning which have observed tremendous success. To truly appreciate the unique challenges in applying machine learning to financial markets, the reader has to first develop at least a cursory understanding of financial markets. This chapter will first outline the motivations of this thesis, provide a primer on quantitative portfolio management, discuss the challenges of forecasting in financial markets and provide potential applications of machine learning in portfolio management.

1.1 Motivation

Machine learning has made significant advances across a wide range of applications, such as achieving human-like accuracy in image recognition (e.g. Krizhevsky et al., 2012; Simonyan and Zisserman, 2015; Szegedy et al., 2015; Schroff et al., 2015; He et al., 2016), speech recognition (Graves et al., 2013; Maas et al., 2013), natural language processing (NLP) (Collobert and Weston, 2008; Sutskever et al., 2014), win against a human champion in the game of *Go* (Silver et al., 2016), fully autonomous driving (Bojarski et al., 2016), synthesis of high quality speech from text (van den Oord et al., 2016), and medical image reconstruction (Kang et al., 2017). By contrast, machine learning applied to financial applications is still in its infancy. There have been some adoption of machine learning in financial applications, such as reinforcement learning for optimal trade execution (Mounjid and Lehalle, 2021) and loan default prediction (Turiel and Aste, 2020). However, linear regression is still a staple of

the financial forecasting toolkit[1]. This begs the question — can machine learning techniques revolutionise financial forecasting, as they have done in many other fields? This question is an important one. Australia has the fastest growing pension market in the world, which ranks as the 4th largest globally (Pham, 2019). Employing forecasting techniques, practitioners manage this pool of capital with the aim of generating higher returns for their clients. Any improvements to forecasting and portfolio management using machine learning could benefit the lives of many future retirees.

Recently, Gu et al. (2020) compared a suite of machine learning models on stock return forecasting and offered a glimpse of the potential of machine learning in financial applications. However, for machine learning models to be truly successful in financial applications, they must first overcome some characteristics of asset returns that complicate the forecasting problem, such as heavy tails, low signal-to-noise ratio and time-varying data generation process (DGP). Guided by this, in this thesis, we review and extend the machine learning literature to tackle some of the challenges as highlighted in Section 1.5. We focus on *deep learning*[2] techniques due to their successes in Gu et al. (2020) and in many other fields, such as image recognition and speech.

1.2 Mechanics of financial markets

In the simplest terms, a financial market (or capital market) is a place where investors (buyers and sellers) exchange financial assets, such as stocks, bonds and foreign currencies[3] at an agreed price (Drake and Fabozzi, 2010). Such activity is fundamental to a well-functioning financial market, which facilitates the transfer of capital[4] from savers (as providers of capital) to companies (as users of capital) and allows savers to earn a return on excess capital (Drake

[1]For example, see Grinold and Kahn (1999) for a discussion on forecasting models used by practitioners for stock return forecasting. The models are predominately linear.

[2]Deep learning is a subfield of *machine learning*. Details are outlined in Section 2.

[3]Common stocks are equity instruments which entitle the holder to fractional ownership of a company. An instrument where the company has agreed to repay the amount borrowed plus interest to the holder is called debt (Drake and Fabozzi, 2010).

[4]In the context of this thesis, *capital* refers to money that is available for investment. In financial theory, capital has a more philosophical meaning.

and Fabozzi, 2010). On any given trading day, buyers and sellers offer to buy or sell quoted quantities of assets. The stock market employs a *limit order book* system (Fabozzi et al., 2011a). Two queues are maintained, bid and ask queues, as illustrated in Table 1.1. In the illustrated state, there is no transaction. Suppose the last transaction occurred at $1.00/share. A new buyer is willing to pay $1.01/share for 10,000 shares. The buyer would have executed an order of 5,000 shares at $1.01 and exhausted the first row of the ask queue. The remaining 5,000 shares are added to the top of the bid queue at $1.01. Now, suppose this buyer is willing to buy 10,000 stocks at any price (as opposed to $1.01 in the original example). Then the buyer would have pushed the price up to $1.02 (second row of the ask queue). This buying (selling) pressure pushing the price up (down) is called *market impact* Each change in last traded price is called a *tick*. The amount of trading activity in a stock is called *liquidity*. A highly liquid stock allows a high volume of trading with a relatively small change in price (Amihud, 2002). In our hypothetical example in Table 1.1, the buyer may value the stock at more than $1.00/share (based on the information they have access to). Thus, they are willing to pay a higher price than the previous traded price of $1.00. Conversely, if a seller decides to sell 20,000 shares at $0.99, they are expecting the stock to be worth less than $0.99. Over the course of trading, buyers and sellers continuously impound information into the price, pushing the price higher if information is positive and lower if information is negative. A bad product review that dissuades would-be customers from purchasing the company's products may have a minuscule impact on the share price. On the contrary, some exogenous shocks, such as a company's profit for the quarter, can have a large impact on the share price. It is important to note that the last traded price reflects information of only the marginal investor. A marginal

TABLE 1.1: An illustrative order book for a hypothetical stock. The bid queue reflects potential buyers willing to buy the stock at the specified price. Similarly, the ask queue reflects potential sellers at the specified price.

#	Bid ($/share)	Quantity	#	Ask ($/share)	Quantity
1	1.00	10,000	1	1.01	5,000
2	0.99	3,000	2	1.02	11,000
3	0.98	24,000	3	1.03	2,500
4	0.95	23,000	4	1.05	51,000
5	0.90	2,000	5	1.07	4,000

investor is the investor making the trade at any point-in-time and determining the next traded price (Damodaran, 2022). In the example described at the beginning of this section, the marginal investors are the buyer and seller executing a trade at \$1.01. If a potential buyer values the stock at only \$0.40/sh, the buyer will sit deep in the bid queue and the order is unlikely to be executed. Thus, the price reflects only the information of the marginal investor and not the average information of all (potential) buyers and sellers. Now, suppose that the investor purchased the stock at \$1.00. One month later, the stock rose to \$1.10 and paid a \$0.10 dividend, the stock's *total return* for the month is $\frac{1.1+0.1}{1} - 1 = 20\,\%$, *price return* is $\frac{1.1}{1} - 1 = 10\,\%$ and *dividend yield* is $\frac{0.1}{1} - 1 = 10\,\%$. Unless specified otherwise, "return" refers to total return the investor received for holding the asset over the said period. In this thesis, we denote *contemporaneous return* (i.e., at time t, total return from $t-1$ to t) as r_t and future return (i.e., total return over t to $t+1$) as y_t. We will often use the term *cross-sectional*, which refers to computing certain quantities on a *per period* basis.

1.3 Return expectations in financial markets

Before we attempt to predict financial markets using machine learning, we have to first ask the question — *are financial markets predictable?* If so, what does the finance literature say about how are financial markets predictable? In finance literature, the study of return expectations (in other words, the prediction of returns) is known as *asset pricing*. In this section, we provide a brief discussion of asset pricing models and empirical findings. For interested readers, more details on this topic is provided in Appendix A1.

Underpinning mainstream financial theories is the *efficient market hypothesis* (Fama, 1970). In the weak form, the hypothesis postulates that investors cannot outperform the market (i.e., achieve higher return than the market at the same level of risk) using publicly known information. This assumption forms the basis of well-known theories such as the Modern Portfolio Theory (MPT) (Markowitz, 1952) and Capital Asset Pricing Model (CAPM) (Sharpe, 1964). CAPM is a theoretically grounded asset pricing model, which stipulates that sensitivity to market return is the *only factor* that is predictive of asset returns. This sensitivity measure

is known as *beta* or CAPM β. CAPM β can be found by regressing a stock's returns on returns of the market (as stipulated by Equation (A.4); Jensen, 1968). To readers familiar with physics, in my view, the importance of CAPM to finance is akin to the *Standard Model* (Oerter, 2006) to quantum physics — it provided a return forecasting model with strong theoretical underpinning and won the joint discoverers the Nobel Prize in Economics (The Nobel Foundation, 1990).

However, unlike the Standard Model, CAPM did not withstand the empirical test. Jensen (1968) was the first to note that CAPM did not align with empirical observations of asset returns. Since the publication of CAPM, numerous *anomalies* are found to be predictive of stock returns, such as the *size effect* (small capitalisation stocks outperform large capitalisation stocks; Banz, 1981) and *value premium* (cheap stocks outperform expensive stocks; Stattman, 1980; Rosenberg et al., 1985). Hundreds of firm characteristics are said to contain information on future stock returns — a survey by Harvey et al. (2016) contained 313 published asset pricing anomalies. The true DGP is likely to be significantly more complex than originally suggested by CAPM and that there may be a large set of factors that drive stock returns. A sufficiently large predictor set that could overwhelm linear regression models. The functional forms of predictors are also unknown. For instance, Fama and MacBeth (1973) tested CAPM β and β^2 and found that both were statistically significant in predicting returns and thus, opening the door to the potential use of machine learning in predicting returns.

1.4 A primer on quantitative portfolio management

Investors provide practitioners (investment managers) with capital, either through the pension system or through excess savings. In doing so, investors expect a positive return on their capital. Grinold and Kahn (1999) stated the objective of an investment manager is to *achieve higher risk-adjusted returns than the market*. More formally, the objective can be stated as,

$$\max_{r_p} \frac{\mathrm{E}[r_p - r_b]}{\sigma[r_p - r_b]}, \tag{1.1}$$

where r_p is return of the portfolio, r_b is return of the benchmark[5] and $\sigma[r_p - r_b]$ is standard deviation of portfolio return in excess of the benchmark. From Equation (1.1), it is immediately obvious that $r_p > r_b$ is required in order for the ratio to be positive. For the rest of this section, we provide a high-level overview of quantitative portfolio management as described in Alford et al. (2011) and Grinold and Kahn (1999). We have intentionally left out details of forecasting models in this section, deferring discussions to Appendix A2 for interested readers.

Quantitative portfolio management involves the use of empirical, systematic and mathematical methods to achieve the objective of the investment manager. The process of conducting quantitative portfolio management (a *quantitative investment process*) is comprised of four stages, as illustrated in Figure 1.1.

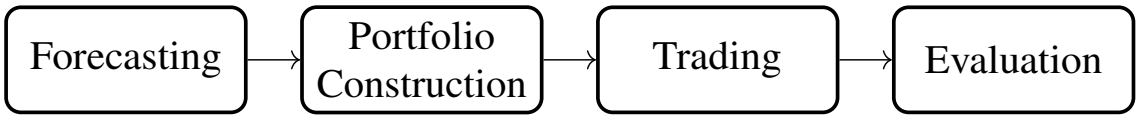


FIGURE 1.1: Illustrative stages of a quantitative investment process. Based on the process described in Alford et al. (2011).

Forecasting is comprised of three components: 1) return forecasts; 2) risk forecasts; and, 3) transaction cost forecasts. Practitioners select stocks based on their return forecasts. Thus, r_p is driven by the predictive power of the practitioner's models. For this reason, return forecasting is described by Alford et al. (2011) as the first and most critical step of an investment process. We start with some features of each stock (also known as *signals* in Grinold and Kahn, 1999; or *factors* and *anomalies* in Section 1.3). Feature examples include market capitalisation, earnings-to-price ratio, and past 12-month return of the stock. Each feature is the result of feature engineering from raw data by practitioners, either by applying domain knowledge or through machine learning (e.g., in Chapter 4, we use time-series neural networks to extract information from stock prices). Let $\tilde{\boldsymbol{X}}_t \sim \mathbb{R}^{N \times M}$ be a matrix of M features of N stocks at time t. Raw feature values are typically converted into scores. Popular

[5] A *stock index* that the portfolio is benchmarked against, e.g., S&P 500 and S&P/ASX 200.

methods include converting raw values into a $[0, 1]$ rank interval (e.g., in Gu et al., 2020) or by standardisation (Grinold and Kahn, 1999),

$$\boldsymbol{x}_{t,m} = \frac{\tilde{\boldsymbol{x}}_{t,m} - \bar{x}_{t,m}}{\sigma(\tilde{\boldsymbol{x}}_{t,m})},$$

where $\tilde{\boldsymbol{x}}_{t,m}$ is the m-th column of feature matrix $\tilde{\boldsymbol{X}}_t$ and $\bar{x}_{t,m}$ is the mean of the m column. Practitioners estimates model F to forecast returns $\hat{\boldsymbol{y}}_t \in \mathbb{R}^N$,

$$\hat{\boldsymbol{y}}_t = F(\boldsymbol{X}_t). \tag{1.2}$$

A popular choice of F amongst practitioners is the *cross-sectional* linear regression (Zhou and Fabozzi, 2011), while neural networks are used in Gu et al. (2020) and Chapter 3. As cross-sectional regression problems are discussed extensively in this thesis, using the example of a linear model as F, we formally introduce the concept of *cross-sectional* prediction in here. Suppose there are N stocks in the market, each with M features, forming input matrix $\boldsymbol{X}_t \in \mathbb{R}^{N\times M}$ at time $t = 1, \ldots, T$. The i-th row in $\boldsymbol{X}_t$ is score vector $\boldsymbol{x}_{t,i} \in \mathbb{R}^M$ of stock i and the m-th column in $\boldsymbol{X}_t$ is score vector $\boldsymbol{x}_t^{(m)} \in \mathbb{R}^N$ of feature m. We define return of stock i as the percentage change in price plus dividends, $r_{t,i} = (p_{t,i} + d_{t,i})/p_{t-1,i} - 1$, where $p_{t,i}$ is price at time t and $d_{t,i}$ is dividend at t if a dividend is paid, and zero otherwise. Regression target at t is return vector $\boldsymbol{y}_t$, where entry i is next period's return $y_{t,i} = r_{t+1,i}$ of stock i. The input-output pair $(\boldsymbol{X}_t, \boldsymbol{y}_t)$ forms a *cross-section* which contains all features at t and realised returns at $t + 1$. The time-series of cross-sections $(1, \ldots, t-1)$ form a *panel dataset* (Wooldridge, 2008): $\mathcal{D}_{t-1} = \bigcup_{\mathtt{t}=1}^{t-1}\{\boldsymbol{X}_\mathtt{t}, \boldsymbol{y}_\mathtt{t}\}$. Given the time-series of cross-sections $\mathcal{D}_{t-1}$, a popular estimation procedure used in finance literature is the *Fama-MacBeth* two-step regression procedure (Fama and MacBeth, 1973). For each $\mathtt{t} \in \{1, ..., t-1\}$, estimate the linear model on the cross-section $\{\boldsymbol{X}_\mathtt{t}, \boldsymbol{y}_\mathtt{t}\}$,

$$\boldsymbol{y}_\mathtt{t} = \hat{\xi}_{\mathtt{t},1}\boldsymbol{x}_{\mathtt{t},1} + \cdots + \hat{\xi}_{\mathtt{t},M}\boldsymbol{x}_{\mathtt{t},M} + \boldsymbol{\epsilon}_\mathtt{t}, \tag{1.3}$$

where $\{\boldsymbol{x}_{\mathtt{t},m} \in \mathbb{R} \,|m = 1, ..., M\}$ are scores of M features at time $\mathtt{t}$, $\hat{\xi}_{\mathtt{t},m}$ are regression coefficients (in finance literature, also known as *factor returns*), and $\boldsymbol{\epsilon}$ are regression residuals. This results in M time-series estimates of factor returns $\{\hat{\xi}_{1,m}, \hat{\xi}_{2,m}, \ldots, \hat{\xi}_{t-1,m}\}, m = 1, \ldots, M$.

Then, for prediction purposes, the expected return of factor m at t is the average of the time-series of observed factor returns over $1, \ldots, t-1$: $\xi'_{t,m} = \frac{1}{t-1} \sum_{j=1}^{t-1} \hat{\xi}_{j,m}$.

The pioneering work by Markowitz (1952) led to the use of variance as a measure of risk and mean-variance optimisation as a method for portfolio construction. Let $\widehat{\boldsymbol{V}}_t \in \mathbb{R}^{N \times N}$ be the estimated variance-covariance matrix,

$$\widehat{\boldsymbol{V}}_t = F^{(\text{risk})}(\boldsymbol{X}_t), \tag{1.4}$$

where $F^{(\text{risk})}$ is the model for forecasting risk. For simplicity, we assume that the risk model uses the same features as the return forecasting model (in practice, they do not have to share the same features). The diagonal of $\widehat{\boldsymbol{V}}_t$ are variance of each stock and off-diagonals are covariance between the row-th and column-th stocks. A linear-regression-based $F^{(\text{risk})}$ is simply an extension of the Fama-MacBeth regression. $t-1$ cross-sectional regressions produces M time-series of factor returns $\{\hat{\boldsymbol{\xi}}_1, \ldots, \hat{\boldsymbol{\xi}}_M\}$. Computing the variance-covariance matrix using the factor returns produces a matrix of factor risks $\boldsymbol{V}'_t \in \mathbb{R}^{M \times M}$. If a factor variance-covariance matrix is used, then the matrix must be expanded back into a stock-level variance-covariance matrix of $N \times N$ dimensions, by multiplying by the feature scores,

$$\widehat{\boldsymbol{V}}_t = \boldsymbol{X}_t \boldsymbol{V}'_t \boldsymbol{X}_t^{\mathsf{T}}. \tag{1.5}$$

As this thesis is mainly focused on return forecasting, we provide further details on risk estimation in Appendix A2.2.

We further define $\hat{\boldsymbol{c}}_t \in \mathbb{R}^N$ to be estimated transaction costs,

$$\hat{\boldsymbol{c}}_t = F^{(\text{cost})}(\boldsymbol{X}_t^{(\text{cost})}), \tag{1.6}$$

where $F^{(\text{cost})}$ and $\boldsymbol{X}_t^{(\text{cost})}$ are transaction cost model and inputs into transaction cost model, respectively. An example of transaction cost model used by practitioners is the square root model (Grinold and Kahn, 1999),

$$\hat{c}_{i,t} = \text{commission} + \frac{\text{bid-ask spread}_{i,t}}{p_{i,t}} + \kappa^{(\text{cost})} \sqrt{\frac{\mathsf{p}_{i,t}^{(\text{trade})}}{\mathsf{p}_{i,t}^{(\text{daily})}}}, \tag{1.7}$$

where commission is payable to facilitators of the trade (e.g., $0.1\,\%$ payable to brokers), bid-ask spread is the difference between the top bid and ask prices in the order book (e.g., bid-ask spread in Table 1.1 is \$0.01), $p_{i,t}$ is price of stock i at t, $\kappa^{(\text{cost})}$ is a scaling factor and is the sole parameter of the model, and $\mathsf{p}_{i,t}^{(\text{trade})}$ and $\mathsf{p}_{i,t}^{(\text{daily})}$ are dollar value of the hypothetical trade and average daily traded value (e.g., 12-month average daily traded value, where daily traded value is the day's share price $\times$ number of shares traded on the day) of stock i at t, respectively. Computation of the hypothetical trade $\mathsf{p}_{i,t}^{(\text{trade})}$ is described later in this section. Equation (1.7) indicates that transaction cost is comprised of three components: 1) a fixed percentage commission; 2) bid-ask spread (represents the cost of buying (selling) at the lowest asking (highest bidding) price rather than waiting in the bid (ask) queue); 3) a market impact component that is proportional to the relative sizes of our trade and liquidity in the stock. Both bid-ask spread and market impact have been introduced in Section 1.2 and are further explained in Appendix A2.3 For example, if the average daily traded value in Stock A is \$1 million, a \$100 trade is unlikely to cause any market impact. However, a \$100,000 buy (sell) is likely to cause the price to move higher (lower). Thus, the investor may pay a higher (or lower) price than the last traded price. In Equation (1.7), this is assumed to be proportional to the square-root of the ratio between value of the trade and daily average traded value in the stock. A further description of the transaction cost forecasting model is also provided in Appendix A2.3.

Combining return, risk and transaction cost forecasts, the portfolio is constructed using mean-variance optimisation (Markowitz, 1952; Grinold and Kahn, 1999),

$$\begin{aligned} \mathbf{w}_t' = \underset{\mathbf{w}}{\operatorname{argmax}}\, & \hat{\boldsymbol{y}}_t^{\mathsf{T}}\, \mathbf{w} - \lambda\, \mathbf{w}^{\mathsf{T}}\, \hat{\boldsymbol{V}}_t\, \mathbf{w} - \hat{\boldsymbol{c}}_t^{\mathsf{T}}(\mathbf{w} - \mathbf{w}_{t-1}) \\ & \text{subject to} \sum_i \mathrm{w}_i = 1, \end{aligned} \tag{1.8}$$

where w_i is the weight of the i-th stock in $\mathbf{w}$, $\mathbf{w}_t'$ are optimal portfolio weights[6] and λ is investor's risk aversion parameter. Note that $\hat{\boldsymbol{y}}_t^{\mathsf{T}}\, \mathbf{w}$ and $\hat{\boldsymbol{c}}_t^{\mathsf{T}}(\mathbf{w} - \mathbf{w}_{t-1})$ are both in units of

[6]Note that the vector of portfolio weights $\mathbf{w}$ (upright) differs from neural network weights $\boldsymbol{W}$ (introduced in Chapter 2) and auxiliary loss weight ω (introduced in Chapter 4).

returns, but $\mathbf{w}^\top \hat{\boldsymbol{V}}_t \mathbf{w}$ is in unit of return variance. Thus, λ also serves as a scaling factor to bring the risk penalty into the same scale as portfolio returns.

We use the following basic example to motivate the discussion on mean-variance optimisation (Equation (1.8)). Suppose that are two stocks with characteristics listed in Table 1.2. The

TABLE 1.2: Hypothetical risks and returns of Stock A and B. *Std Dev* stands for standard deviation of expected return. ρ is correlation between the expected return of Stock A and B. Optimal Portfolio is solved by maximising the Return/Risk of the portfolio by allocating to both Stock A and B.

	Stock A	Stock B	Optimal Portfolio
Expected Return (%)	10.0	8.0	8.7
Std Dev (%)	12.0	9.0	9.2
Return/Risk	0.83	0.89	0.95
$\rho_{A,B}$	0.65	0.65	

goal of the investor is to find the optimal wealth allocation within a set of assets (in this basic example, between Stock A and B). Clearly, if the investor seeks the highest expected return, the investor should allocate $100\,\%$ of their wealth into Stock A. If the investor seeks the lowest risk, then the investor should allocate their wealth into Stock B. Markowitz (1952) showed that the portfolio optimisation problem can be generalised into Equation (1.8), a convex optimisation problem, and that various portfolio objectives (e.g., maximising return, minimising risk, or the simultaneous trade-off of both) can be achieved by varying level of λ. Markowitz (1952) also showed that a better return/risk can be achieved (compared to investing into a single asset) by diversifying across assets with expected returns that are not perfectly correlated (theoretical background on *Modern Portfolio Theory* is given in Appendix A1). In our two-stock basic example, expected return of the portfolio is,

$$\mathrm{E}[r_p] = w_A\,\mathrm{E}[r_A] + w_B\,\mathrm{E}[r_B],$$

where $\mathrm{E}[r_{\{p,A,B\}}]$ and $w_{\{p,A,B\}}$ are expected return and weights of portfolio, Stock A and Stock B, respectively. Expected risk (when measured in variance) of the portfolio is,

$$\sigma_p^2 = w_A^2\sigma_A^2 + w_B^2\sigma_B^2 + 2w_A w_b \sigma_A \sigma_B \rho_{A,B},$$

where $\sigma^2_{\{p,A,B\}}$ is expected risk of portfolio, Stock A and Stock B, respectively. Continuing with our basic example, for illustrative purposes, suppose the investor is seeking the maximum $\frac{\mathrm{E}[r_p]}{\sigma_p}$ (this is known as the *Sharpe ratio* when computed on realised portfolio returns and is introduced in Section 3.2). Then, using a solver, we find that the optimal allocation is $w_A = 0.36$ and $w_B = 0.64$, which results in $\mathrm{E}[r_p] = 0.087, \sigma_p = 0.092, \frac{\mathrm{E}[r_p]}{\sigma_p} = 0.95$ (also shown in the last column of Table 1.2). Thus, by diversifying across two stocks, the portfolio has higher expected return and lower risk than investing in Stock B and Stock A alone, respectively. Solving Equation (1.8) using a solver and assuming transaction cost is zero, we find that the maximum $\frac{\mathrm{E}[r_p]}{\sigma_p}$ objective is equivalent to $\lambda = 5$ in Equation (1.8). Other special cases include maximum return ($\lambda = 0$) and minimum risk ($\lambda \to \infty$). In general, risk aversion parameter λ is subjectively chosen by the investor depending on their risk appetite.

So far, our basic example involves a single period optimisation. Suppose the investor rebalances their portfolio at the end of every month and that the weights immediately prior to rebalancing are $w_A = 0.3$ and $w_B = 0.7$. Changing from $w_A = 0.3$ to $w_A = 0.36$ ($w_B = 0.7$ to $w_A = 0.64$) incurs transaction costs. Thus, the expected *after cost* return of Stock A (B) is likely less than $15\,\%$ ($8\,\%$), and the optimal portfolio in the presence of transaction costs will differ to the theoretical frictionless optimal portfolio. Suppose that the portfolio is \$1 million in value and transaction costs are assumed to be $\hat{\boldsymbol{c}}_t = [0.002, 0.002]^\mathsf{T}$. Then, the hypothetical trade of Stock A is computed as $\mathsf{p}_{i,t}^{(\text{trade})} = 1,000,000 \times (0.36 - 0.3) = 60,000$, which incurs cost of $60,000 \times 0.002 = 120$.

In analysing Equation (1.8), it can be seen that the objective is maximised if the portfolio places $100\,\%$ weight onto the stock with the highest expected return, if risk and costs are ignored. However, with a covariance matrix where the entries $\hat{v}_{t,i,j} < \hat{v}_{t,i,i}, i \neq j$, the risk penalty $\mathbf{w}^\mathsf{T} \widehat{\boldsymbol{V}}_t \mathbf{w}$ encourages diversification and prevents the portfolio from being fully aligned with return forecasts. The portfolio is further constrained by the cost to trade $\hat{\boldsymbol{c}}_t^\mathsf{T}(\mathbf{w} - \mathbf{w}_{t-1})$, which only allows the portfolio to switch between stocks if the increase in expected return is greater than the two-way cost to trade (i.e., the cost of a buy and a sell). A stock position can become "stale" if its return forecast no longer ranks highly but no other stock offers a sufficiently high return forecast to cover transaction costs. In this case, a portfolio can

continue to hold a suboptimal stock even if better (theoretical) options are available. Thus, over time, portfolio weights reflect the weighted averages of past return forecasts, rather than the latest return forecasts. In sum, mean-variance optimisation is a balancing act between maximising return (forecasts), while minimising risk penalty and transaction costs.

Next, the desired portfolio is then implemented by trading the difference between the desired portfolio and the existing portfolio,

$$\mathbf{w}_t = f^{(\text{trade})}(\mathbf{w}'_t - \mathbf{w}_{t-1}),$$

where $f^{(\text{trade})}$ denotes the *trading function* that produces the actual portfolio $\mathbf{w}_t$ (i.e., an abstract function that involves sending orders to the market and observing actual execution of the order) and the trades are given by $\mathbf{w}'_t - \mathbf{w}_{t-1}$. Actual transaction costs incurred by trading is given by,

$$\boldsymbol{c}_t = f^{(\text{tcost})}(\mathbf{w}'_t - \mathbf{w}_{t-1}),$$

where $f^{(\text{tcost})}$ is the actual market impact (as discussed in Section 1.2) and commissions paid for the trades. For example, if inputs into $f^{(\text{tcost})}$ are \$1 million worth of trades, and costs are comprised of $0.15\,\%$ of market impact and $0.05\,\%$ of commission, then actual transaction cost is $c_t = \$1\,\text{million} \times (0.0015 + 0.0005) = \$1000$. To minimise market impact, a practitioner may choose to trade patiently[7]. In doing so, the practitioner incurs opportunity cost (if return forecasts are predictive of returns, waiting for a favourable trade price will lead to foregone returns) and risk around the eventual execution price (Alford et al., 2011). The resultant portfolio may differ from the desired portfolio, due to adverse price movements or prevailing liquidity of the stocks.

Finally, performance of the portfolio is computed as the sum of actual realised stock returns less actual transaction costs,

$$r_{p,t+1} = \mathbf{w}_t^{\mathsf{T}}\, \boldsymbol{r}_{t+1} - \boldsymbol{c}_t^{\mathsf{T}}(\mathbf{w}'_t - \mathbf{w}_{t-1}). \tag{1.9}$$

[7]Using the example in Section 1.2 to illustrate, suppose the maximum price the investor is willing to pay is \$1.00/share. Then, the investor's bid will sit in the bid queue, waiting for a seller who is willing to sell at \$1.00/share. The investor is guaranteed that the price paid is \$1.00 but there is uncertainty as to when the trade will occur (if at all). Conversely, if the investor is willing to pay any price, then the investor can trade immediately but will consume the ask queue and thus "move" the market price with the trade.

This completes the link from return forecasts (the most important step of the investment process) to the outcome of the portfolio (objective of the investment manager).

1.5 Challenges of forecasting in financial markets

So far, we have introduced quantitative portfolio management, basic financial theory and the vast potential feature set. We have also briefly discussed the importance of forecasting. In this section, we start by providing several stylised facts on financial markets before formally describing the challenges of forecasting in financial markets.

Fact 1: Asset returns have heavier tails than stipulated by the Normal distribution (Cont, 2001).

For example, on 26th July 2018, Facebook Inc. reported lower than expected second quarter revenue and daily active user count (Salinas and Castillo, 2018). The stock fell $18.96\,\%$ on the day, as illustrated in Figure 1.2. This event is so rare that assuming daily returns are normally distributed and using observations from *initial public offering*[8] to 25th July 2018, the probability of observing such an event is $6e^{-17}$.

Fact 2: Financial markets can exhibit endogeneity.

Literature and media outlets have documented some evidence of endogenous factors driving stock returns. Over December 2020 to January 2021, social media users coordinated trading activity in GameStop Inc., causing its share price to rise $1998\,\%$ in two months. The dramatic rise in value of GameStop shares was partly driven by a phenomenon known as a *short squeeze* (El-Erian, 2021), where investors betting against a rising stock are forced to unwind their bet, causing further buying pressure on the stock. This caused cascading buying pressure on the stock and an escalating stock price, as shown in Figure 1.3. Endogeneity is also said to have caused some stock market crashes, such as the crash of October 1929 and the "Dot-com" crash of April 2000 (Johansen and Sornette, 2002).

[8] Initial public offering (IPO) is when a private company sell shares to the public for the first time. After IPO, the stock becomes a publicly traded stock on the stock exchange.

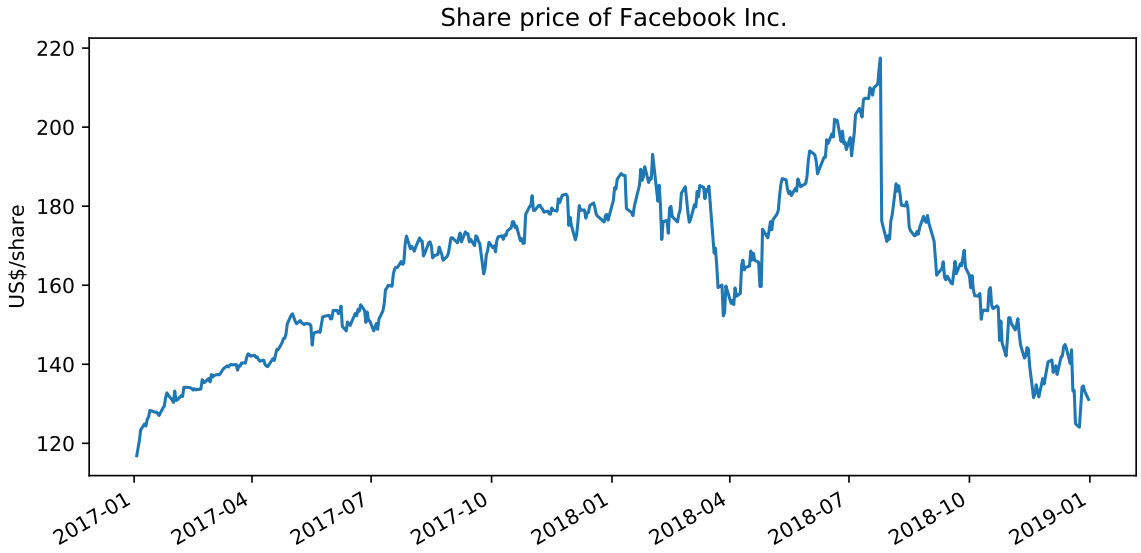


FIGURE 1.2: Share price of Facebook Inc. over 2017–18. The share price of Facebook is observed to be on an upward trajectory prior to July 26, 2018 and a downward trajectory afterwards. Source: Center for Research in Security Prices (CRSP) database.

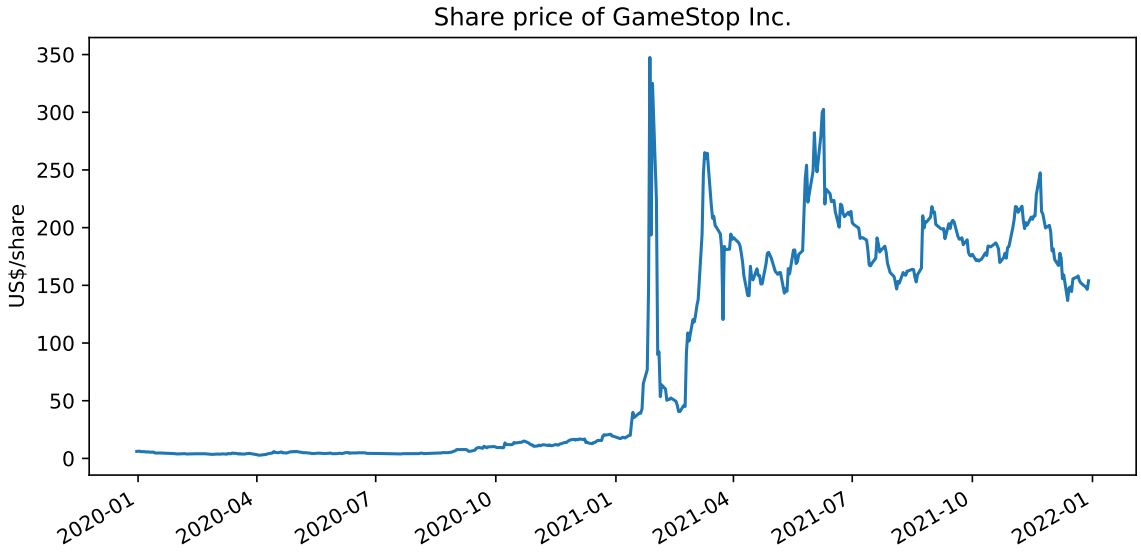


FIGURE 1.3: Share price of GameStop Inc. over 2020–21. The share price of GameStop peaked on January 27, 2021. Source: Yahoo! Finance (2022a).

Fact 3: Asset returns exhibit volatility clustering, where returns display irregular bursts of volatility that are localised in time (Cont, 2001).

Stock market crashes can also be caused by an exogenous shock, such as the 2020 global stock market crash due to an emerging pandemic (Song et al., 2022). For example, the price of Devon Energy, a U.S. oil and gas producer, plummeted during the March 2020 stock market crash, as shown in Figure 1.4. The square of daily returns jumped to 0.14 during the height of the crash, compared to a mean of 0.001 over 2017–2021. Volatility is also seen to *cluster* in time. Following the peak in March 2020, volatility remains elevated over the next 6–12 months.

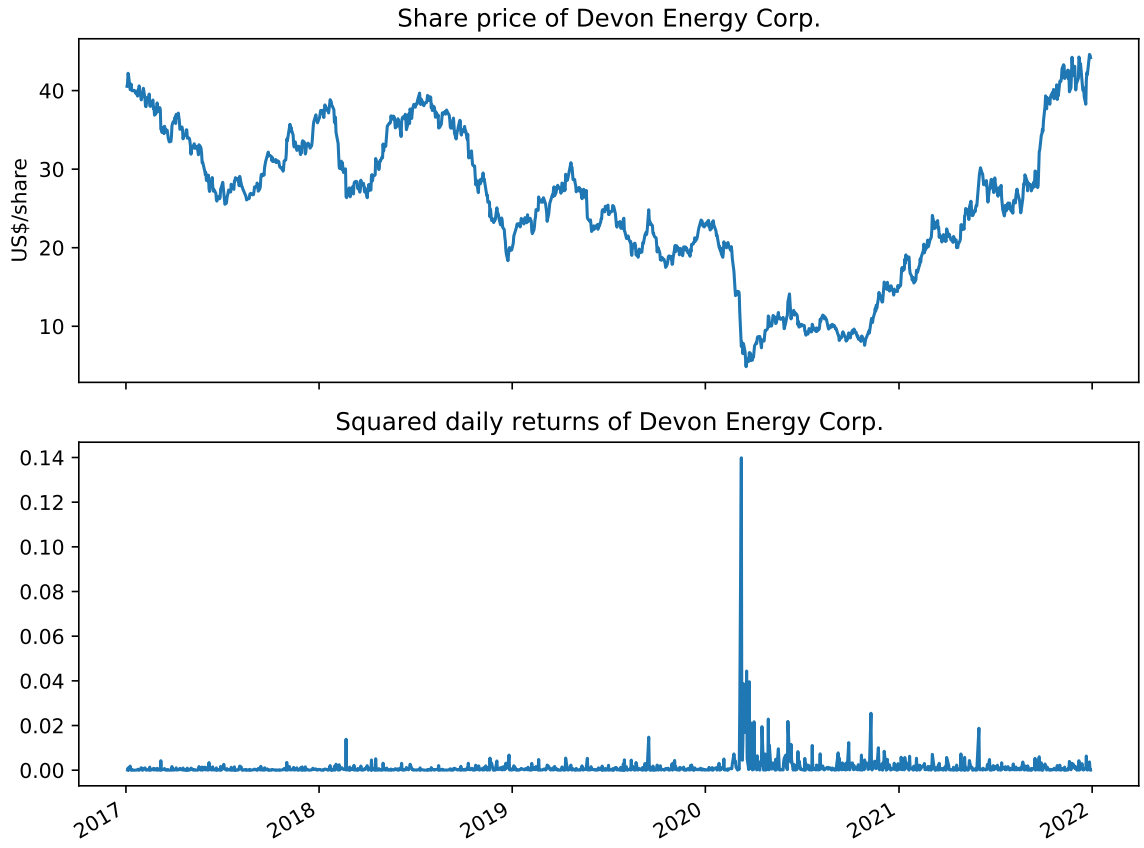


FIGURE 1.4: Share price of Devon Energy Corp. over 2019–21. Daily return volatility spiked to $14\,\%$ during the height of the crash. Source: Yahoo! Finance (2022b).

In addition to **heavy tails** and **volatility clustering**, Cont (2001) has documented other asset return characteristics such as *absence of autocorrelation*, *aggregational Gaussianity*, *skewness* and *conditional heavy tails*.

There is one major distinction between the setup of conventional machine learning applications and financial applications — stationarity of the DGP. In time-series analysis, *non-stationarity* typically refers to properties (e.g., mean and variance) of a time-series changing over time (Nason, 2006). In machine learning literature, non-stationarity refers to the DGP changing over time, such as changes in the conditional distribution of the output given the input (Gama et al., 2014). To avoid ambiguity, we use the terms *time-varying* or *time-variability* of the DGP to refer to non-stationary DGP. However, we continue to use the term *stationary* to describe DGP that do not change over time.

Conventional machine learning models are trained offline, using a historical set of training data and are deployed after batch training (Gama et al., 2014). This training scheme is suitable for stationary problems where the training set is assumed to be drawn from the same DGP as out-of-sample data. As out-of-sample data are drawn from the same distribution as training data, the *generalisation gap*[9] is expected to be relatively small and the model is expected to perform well after deployment. Examples of stationary problems include image recognition (Schroff et al., 2015) and text translation (Sutskever et al., 2014). However, some prediction problems are time-varying, yielding a phenomenon known as *concept drift* (Schlimmer and Granger, 1986; Widmer and Kubat, 1996; Gama et al., 2014). For example, predicting a user's interests when following an online news stream is likely time-varying (Gama et al., 2014). Finance literature has also documented evidence of time-variation of the DGP. Pesaran and Timmermann (1995) estimated linear models with permutations of firm characteristics over time, and performing model selection using both statistical and financial measures on U.S. stocks. Both the selected variables and their coefficients of the best model change over time. Bossaerts and Hillion (1999) reported similar findings in international stocks. There is no consensus on the cause of time-varying predictability in the academic discourse. Some argued that this time-variability is driven by macroeconomic conditions (e.g., Angelidis et al., 2015). While explanations offered by McLean and Pontiff (2016) relate to data-mining bias and effects of arbitrage by investors (which the authors referred to as publication-informed trading). In other words, investors taking advantage of this effect causes its "mispricing" to

[9]Generalisation gap is defined as the difference between out-of-sample loss and training loss (Goodfellow et al., 2016).

disappear (for example, see Dong et al., 2020 for a proposed mechanism with which this occurs). Thus, it is unsatisfactory for a practitioner to learn a static model as out-of-sample performance can vary.

As noted in Section 1.3, there are hundreds or more factors that exhibit predictive power over stock returns. These include (but are not limited to):

- Price-derived features, such as a stock's past performance (Jegadeesh and Titman, 1993).
- Financial statement-derived features, such as valuation metrics (Asness et al., 2013).
- Social media (as illustrated by the GameStop example) and web searches (Huang et al., 2020).
- Media reports (Fang and Peress, 2009).

The dataset in Gu et al. (2020) contains mainly price and financial statement features and is the same dataset used in Chapter 3. In Chapter 4 and 5, only price features are used as these chapters focuses on time-series predictions and uncertainty quantification (of time-series predictions). Such a vast feature set has the potential of overwhelming conventional regression techniques such as ordinary least squares (OLS), due to multi-collinearity (Gu et al., 2020). Thus, any proposed machine learning alternatives to linear models must be able to handle a large feature set.

In sum, stock return prediction poses a unique challenge for machine learning research. Stock returns exhibit difficult to handle statistical characteristics such as heavy tails and volatility clustering, and suffer from low signal-to-noise ratio and time-variability of the DGP. These challenges are distinct from conventional applications of machine learning which have seen significant advances.

1.6 Potential applications of machine learning in portfolio management

As discussed in Section 1.4 and 1.5, machine learning in portfolio management presents some unique challenges and requires new approaches that differ from conventional applications. In this section, we discuss gaps in literature and identify ways in which machine learning can be advanced or applied in financial markets.

1.6.1 Cross-sectional prediction using online deep learning

As noted in Section 1.5, return forecasting is an arduous task. Stock returns are not well-behaved, plagued with heavy tails, low signal-to-noise ratio and time-varying cross-sectional relationships. However, it is also the most important step of an investment process.

Much of the finance industry still relies on linear models. By contrast, machine learning has achieved significant progress in other fields and could similarly offer prediction performance improvements to empirical finance. Weigand (2019) provided a recent survey of machine learning applied to empirical finance and noted that machine learning algorithms show promise in addressing shortcomings of conventional linear models (such as the inability to model non-linearities and handle large number of covariates). Notable works applying neural networks to cross-sectional stock return prediction using a large feature set include Messmer (2017), Abe and Nakayama (2018) and Gu et al. (2020). Both Messmer (2017) and Abe and Nakayama (2018) are straightforward applications of feedforward networks[10] on stock returns, where the input consists of tens of features, predicting U.S. and Japan stock returns, respectively. Gu et al. (2020) compared a set of well-known machine learning models on forecasting U.S. stock returns and found neural networks to provide the best performance. Potential time-variability is assumed to be driven by macroeconomic conditions and is modelled by interacting firm level features with macroeconomic indicators. Arguably, this is an inefficient way of modelling interaction effects, as the 94 firm-level features are

[10] Feedforward neural networks are discussed in Section 2.1.

interacted with 8 macroeconomic variables, resulting in 920 features (together with dummy variables of 74 industries, $94 \times (8 + 1) + 74 = 920$). Of the firm-level features used in the three works, Gu et al. (2020) contains the most firm-level features (94). Messmer (2017) contains 61 and are all contained within Gu et al. (2020). Abe and Nakayama (2018) contains the least, at 25. However, due to different naming conventions, we cannot ascertain how many are contained with Gu et al. (2020). Moreover, they do not consider all possible avenues of time-variability of asset pricing models, such as the effects of investors' own trading, as highlighted by McLean and Pontiff (2016), and exogenous shocks. For instance, Lev and Srivastava (2019) noted that the prominent *value* factor[11] (Rosenberg et al., 1985; Fama and French, 1992) has been unprofitable for almost 30 years — a period that includes multiple business cycles and thus cannot be explained by macroeconomic conditions alone. The authors noted that returns to the value factor have been negative since 2007, suggesting a change in the underlying relationship. There are further empirical evidence of changes in DGP. Employing *genetic algorithms*[12] (Mitchell, 1996) to predict U.S. stock returns, Brogaard and Zareei (2022) also found stock return predictability to have declined over time, which implies that markets have become increasingly more efficient. Other works have sought to incorporate finance theory directly into the network architecture and have used more advanced network architectures. Gu et al. (2021) used an autoencoder to form "latent factors" and factor exposures, in similar spirit as principal component analysis (PCA) (Hastie et al., 2020). The resultant model is analogous to the Arbitrage Pricing Theory (APT) model but with latent factors constructed by the autoencoder from a large feature set. Chen et al. (2021) used a *generative adversarial network* (Goodfellow et al., 2014) to enforce the *no arbitrage condition* in APT and reported strong performance in predicting U.S. stock returns. Changing DGP is modelled with 178 macroeconomic indices. The authors reported declining performance over time, and that using a small number of macroeconomic indices is only marginally better than no macroeconomic data (i.e., a fixed DGP), and that including all 178 indices led to severely

[11] Suppose the share price is \$1.00 and the firm's asset value (net of debt) is \$2.00 per share. Then the *book-to-market* score is $2/1 = 2$. A high score is interpreted as the stock trading cheaply relative to value of its assets. This factor has been profitable since 1920s but its profitability has greatly diminished since its discovery in 1986. Lev and Srivastava (2019) argue that this is due to deficiencies in accounting standards and economic development.

[12] Genetic algorithms randomly search through candidate model specifications through simulated evolution.

worse performance. Financial markets have also observed plenty of exogenous shocks over time, some of which do not have a parallel in history (e.g., the COVID-19 pandemic).

These empirical evidence suggest changes in the DGP may be unpredictable (e.g., due to investors' own arbitrage and exogenous shocks). Thus, there exists a gap in literature for deep learning models that can track changes in the DGP of financial markets driven by unknown dynamics.

1.6.2 Time-series pattern recognition in noisy environments

Stock returns are notoriously noisy. The best performing model in Gu et al. (2020) had R^2 of $0.4\,\%$[13]. In practice, cross-sectional correlation between expected return and actual realised return of $5\,\%$ can be considered as "good" and $10\,\%$ is "great" (Grinold and Kahn, 1999). By contrast, state-of-the-art image recognition models can achieve image classification accuracy of over $90\,\%$ (for example, see Zhai et al., 2021). Thus, from a signal-to-noise perspective, financial markets are vastly different from fields where deep learning has excelled, such as image recognition.

In neural network architectural design, a network with greater *depth* is thought to be more efficient than a shallow but *wide* network (i.e., a network with only a few layers but each layer has many nodes) in approximating an arbitrary function (Lu et al., 2017). Training very deep neural networks (over 100 layers deep) for image classification problems saw a breakthrough in the form of *ResNet* (He et al., 2016), where skip connections that "jump" over one or more layers are added to allow uninterrupted information flow between connecting layers. This alleviates the problem of *vanishing gradient*, where the magnitude of the gradient diminishes as training error is backpropagated through the network (Kolen and Kremer, 2001). However, in very noisy environments such as financial markets, focus of training should be on robustness rather than expressiveness, as highly expressive networks may overfit on noise, leading to poor out-of-sample performance. To this end, there is no conclusive evidence in literature regarding the robustness of neural networks in noisy environments. Drawing on

[13] Gu et al. (2020) used a non-standard definition of R^2. Further details are provided in Section 3.2.1 and Section 3.5.

findings in other applications, Rolnick et al. (2018) finds that neural networks are robust to high levels of artificially injected mis-classified labels in simple image recognition tasks. The authors note that effective batch size (a concept that we will introduce in Section 3.2.2) decreases as the level of white noise increases. Thus, highly noisy environments require larger batch sizes. On the contrary, Moradi et al. (2021) finds that neural networks are not robust to noise in clinical text. The authors inject character-level and word-level perturbations to reflect realistic typographic errors encountered in the real world and find three different language models trained specifically on clinical texts to have experienced material accuracy declines in medical diagnostic tasks.

Ways to combat noisy data include increasing the amount of data used in training (Rolnick et al., 2018)), which may not be readily available, and regularising the model. Popular regularisation techniques for neural networks are L_1 and L_2 penalties (Goodfellow et al., 2016), *early-stopping* (Goodfellow et al., 2016) and *dropouts* (Srivastava et al., 2014). L_1 and L_2 penalties in neural networks are analogous to their counterparts in linear models and shrink network weights toward zero. Early-stopping can be interpreted as L_2 penalties, and dropout can be interpreted as ensembling using subnetworks. Both of these techniques are introduced in Section 2.4. For classification problems, Patrini et al. (2017) propose to estimate noise rates in class labels and introducing a correction term in the loss function which negates the probability that a label is assigned due to noise. Multi-task learning (MTL) has also been shown to improve generalisation performance across a range of classification tasks, such as facial landmark recognition (Zhang et al., 2014) and natural language processing (Collobert et al., 2011). MTL involves the addition of an auxiliary learning task that is related to the primary learning task. The auxiliary learning task is thought to encourage representation sharing and is introduced more formally in Section 4.1. There remains a need for regularisation techniques specifically designed for regression noisy environments that may potentially have broader applications outside of finance.

In Section 1.6.1, we have introduced the cross-sectional prediction problem in finance. An alternative to cross-sectional prediction that is applicable in financial markets is *time-series forecasting*. This can be interpreted as pattern recognition on past stock price or return

patterns to forecast future returns. Convolutional neural networks (CNNs) have proved to be invaluable in image recognition tasks (e.g., Krizhevsky et al., 2012) and could extract more patterns from share prices beyond anecdotal patterns documented in technical analysis. Sezer et al. (2020) provided a recent survey on financial time-series forecasting with deep learning and noted long short-term memory (LSTM) was the most popular method, followed by CNN. Most works are straightforward applications of different neural network architectures on stock returns (e.g., Chen et al., 2015), and are on 1–3 days ahead forecasts. This differs to the 1-month ahead forecast of the momentum effect which, we argue, is more relevant for investment managers due to constraints of transaction costs. There are two approaches to dealing with noise in financial time-series forecasting in literature, both of which we see as being deficient. First is to treat the time-series forecasting problem as a classification problem (i.e., 1 if the stock rose over the next day, 0 otherwise; see Chen et al., 2015; Altilio et al., 2019). This neglects the magnitudes of expected returns which will help practitioners in differentiating relative performance of stocks. Second is to first apply wavelet transform (Meyer, 1993) to denoise the sequence, then fit the denoised trend using a neural network (e.g., Yan and Ouyang, 2017; Li and Tam, 2017). Wavelet transform treats share price oscillation around a trend as "waves" which are then removed. This relies on fitting parametric waves onto the sequence and may inadvertently remove useful features from the sequence. Gap exists in existing literature for an end-to-end neural pattern recognition technique that is robust to noisy patterns in stock prices.

1.6.3 Forecast uncertainty quantification

Consider the following thought experiment. Suppose a practitioner has a model that can perfectly forecast next day's asset returns and that the practitioner's goal is to maximise terminal wealth. Then, on each day, the most rational decision would be to place all of the investor's wealth into the asset with the highest expected return on the next day. Next, suppose that the investor's model is a noisy estimator of future asset returns. Then, the investor may choose to diversify across multiple assets and not place all their wealth on a single bet. Based on this thought experiment, we would expect forecast certainty to have a role in

the portfolio optimisation process. Various bet allocation models have been developed. In wagering, where bets are independent and have well-defined binary outcomes, the optimal bet allocation strategy is the *Kelly criterion* (Kelly, 1956). Forecast certainty is incorporated into the Kelly criterion via expected probabilities of discrete outcomes. The Kelly criterion has been extended to the case of Gaussian distributed outcome, where the optimal bet size is scaled by the inverse of variance (Byrnes and Barnett, 2018). Mean-variance portfolio optimisation (Markowitz, 1952) assumes that asset returns are described by mean and variance of their expected returns. The resultant portfolio is extremely sensitive to expected returns which are difficult to forecast. Black-Litterman portfolio optimisation was proposed to address this shortcoming (Black and Litterman, 1992). In Black-Litterman, practitioners provide both their "views" (expected returns) and "strength" of their views (forecast certainty). These views are then incorporated into portfolio optimisation as priors in a Bayesian manner. Thus, it is useful for a neural network to provide both the conditional mean (forecast) and conditional variance (forecast uncertainty) which can then be used downstream in portfolio optimisation, such as in determining optimal bet size.

Bayesian neural networks offer both forecasts and forecast uncertainties through imposing a full Bayesian treatment over the entire network (Mitros and Namee, 2019). This involves placing priors on network weights and training the network using Monte Carlo Markov Chain (MCMC). However, a full Bayesian treatment incurs a high computational cost (Quiroz et al., 2019). Recent advances focus on generating parameters of a distribution that is assumed to have generated the data, (e.g., Lakshminarayanan et al., 2017; Amini et al., 2020). These works provide an interesting way of quantifying forecast uncertainty, without the cost penalty of a full Bayesian approach. However, both Lakshminarayanan et al. (2017) and Amini et al. (2020) were developed for non-time-series applications and do not consider the possibility of variance changing over time. In the context of financial time-series forecasting, stock returns are known to exhibit *volatility clustering* (as discussed in Section 1.5). Gaps exist in literature for neural network forecast uncertainty quantification techniques that can handle time-varying uncertainty, particularly in the context of financial time-series. From a finance application perspective, forecast uncertainty can be used to size bets, or as advanced warning to protect the portfolio from increasing risk. For example, if forecast uncertainty reaches

a certain threshold, an investor could purchase portfolio insurance (e.g., put options which allow the investor to sell stocks to the issuer of the options at a pre-agreed price) or liquidate positions to reduce risk.

1.6.4 Other possible directions

In addition to the aforementioned applications of machine learning in portfolio management, machine learning can also be used to extract useful information from unstructured data and for efficient trading (in cost minimisation). These two topics are not addressed in this thesis. Nonetheless, we provide a discussion on the two topics as future research directions.

Inputs form the bedrock of any prediction model. Pre-existing feature sets used in literature and in practice typically evolve around price, company financial information and other economic variables. Recently, there is an emerging trend towards incorporating *alternative data* in the investment process. These datasets comprise of unconventional and often unstructured information about a company or industry. For instance, Ranco et al. (2015) used Support Vector Machine (SVM) to classify $1.5\,\text{million}$ tweets on Twitter[14] on 30 stocks in the Dow Jones Industrial Average index (Dow Jones) into positive, neutral and negative sentiment over 15 months. The authors found that polarity of tweet sentiment was associated with $1\,\%$ of cumulative excess return in the three days following the peak. Souma et al. (2019) used a LSTM to classify Reuters[15] news articles into positive and negative sentiment. The authors reported positive prediction accuracy when applied to high frequency intraday tick data on stocks in the Dow Jones index. Other potentially useful unstructured data include job listing websites, product reviews, website traffic, satellite imagery of factories and car parks, and product internet search trends. Techniques such as deep learning can be used to convert these unstructured data into quantifiable data and incorporated in a prediction model.

After a portfolio has been selected, investors would implement the portfolio in the most cost efficient manner. As introduced in Section 1.2 and discussed in Appendix A2.3, market

[14]Twitter is a social media platform where users can post short messages (called *tweets*) of any topic. URL: twitter.com

[15]Reuters is a global news outlet. URL: www.reuters.com

impact could substantially reduce realised returns. Therefore, the goal is to design a trading policy (e.g., when and how to split a trade into parcels) that minimises costs incurred. This is related to the *one-way trading problem* in computer science (El-Yaniv et al., 2001), where a player observes a price sequence and decides whether or not to accept the current price. The game ends when the required amount has been traded. There are two approaches to this problem — online learning and reinforcement learning. Online learning (also called no-regret learning, a concept to be introduced in Section 3.2.3) approaches the optimal trading problem through the lens of game theoretics. The problem is set as a game against an adversary (also called the *nature*) and the goal is to compare favourably to the best expert in hindsight (e.g., the best model parameters trained using all observations up to t). Dworkin et al. (2014) proposed the *Pursuit-Evasion Without Regret* algorithm for optimal trading, which extends online learning to incorporate a *state*[16]. The authors showed that the proposed algorithm outperformed the *constrained follow-the-lazy-leader* algorithm on inventory management. Distinct features of no-regret learning are the assumption of adversarial outcome and the focus on worst-case performance (as nature can arbitrarily choose an outcome that is to the worst detriment of the player). Possible extensions to this line of literature is to allow for *some* predictability in intraday returns, such as the reversal pattern. Another possible approach is *reinforcement learning*, which focuses on learning the optimal policy itself. Nevmyvaka et al. (2006) was the first to apply reinforcement learning on the optimal trading problem and reported substantial reduction in trading costs. Optimal policy learnt was based on time left to trade and quantity remaining. One potential advancement is to use *deep reinforcement learning* (e.g., see François-Lavet et al., 2018) to learn a more complex Q-value function, beyond simply using the time and quantity dimensions. For instance, other variables such as market capitalisation and recent share price performance may contain useful information for optimal trade execution.

[16]The application of Dworkin et al. (2014) is in algorithmic trading, where an algorithm is used to autonomously trade stocks in order to achieve an objective (e.g., minimise cost). In this context, state refers to the inventory position in a given stock. Conventional no-regret learning algorithms are stateless. This stateful augmentation blends reinforcement learning with no-regret learning (Dworkin et al., 2014).

1.7 Contributions and structure of the thesis

At this point, the reader has been introduced to the portfolio management problem and relevant financial theory. As discussed in Section 1.5 and 1.6, there are several gaps in literature on applying deep learning in a portfolio management context.

This thesis provides three significant advances in deep learning techniques applicable to financial markets. Each contribution details a methodological improvement to deep learning that addresses a challenge in financial markets (e.g., time-varying DGP, low signal-to-noise and volatility clustering), combined with a demonstration of the contribution in a financial market application. As deep learning is featured in all three main contributions of this thesis, a comprehensive review of neural networks is provided in Chapter 2, covering common network architectures (Section 2.1, 2.5.1, 2.5.2), network training and optimisation (Section 2.2), weight initialisation (Section 2.3), activation functions (Section 2.1), and neural network regularisation techniques (Section 2.4). This chapter provides readers with an essential understanding of neural networks which is required to fully grasp the advances proposed in the rest of this thesis.

In Chapter 3, we address the time-varying cross-sectional prediction problem (as described in Section 1.6.1) by introducing the Online Early Stopping (OES) algorithm for training neural networks online. The neural network trained using OES is able to adapt to changes in the DGP over time. We provide an optimality guarantee, where the performance of the algorithm is lower bound by a multiple of the variance of the DGP. We compare OES to a stationary network (i.e., a network that is trained offline and does not vary with time) and Dynamic Exponentially Time-Smoothed Stochastic Gradient Descent (DTS-SGD), a state-of-the-art online non-convex optimisation algorithm (introduced in Section 3.2.3). We demonstrate the benefits of OES on a synthetic dataset and in predicting U.S. stock returns as a direct comparison to Gu et al. (2020). This application tackles both the cross-sectional stock return forecasting problem (as discussed in Section 1.6.1) and time-variability problem of financial markets (as discussed in Section 1.5). We show that a neural network trained using OES outperformed the state-of-the-art on the aforementioned problems. We also demonstrate that

the network is able to track changes in the market, such as turning points of markets, with compelling results that are likely to be useful to practitioners.

In Chapter 4, we simultaneously tackle the time-series pattern recognition and noise-robust learning problems (as discussed in Section 1.6.2) by introducing a supervised autoencoder[17] into a temporal convolutional network. We name this network the Supervised Temporal Autoencoder (STAE). We argue that the supervised autoencoder imposes a nonparametric functional form on the model, encouraging the network to retain features that are beneficial to both the primary forecasting task and the auxiliary reconstruction task. The reconstruction task also improves the interpretability of the model, as users can visualise features retained by the network by inspecting the reconstructed sequence. We show that the proposed STAE provides economically meaningful improvements over the *momentum effect*, a known predictor of stock returns in finance literature. We also provide a precedence on applying sequential neural networks[18] to financial time-series at a large scale and is investable through forecasting 1-month ahead returns. We provide a benchmark against popular sequential neural networks, namely temporal convolutional network (TCN), LSTM, Neural Basis Expansion Analysis for interpretable Time Series (N-BEATS) and transformer (both N-BEATS and transformer are state-of-the-art architectures for time-series/sequential applications), and demonstrate class-leading performance. We hypothesise that supervised autoencoder is a potent regularisation technique for neural networks that may find applications in other noisy environments.

In Chapter 5, we combine and extend two state-of-the-art methods, Ensemble and Evidential, into a unified framework for the quantification of uncertainty in financial time-series (as discussed in Section 1.6.3). The framework comprises of four improvements, namely the use of SMD, separate modelling of distribution hyperparameters, ensembling and use of second order return information. We propose a simplified parameterisation of the problem as a scale mixture, which leads to the use of a Gamma prior on a variance scaling variable. Using the UCI benchmark dataset, we demonstrate uncertainty quantification performance that is overwhelmingly in favour of SMD over Normal-Inverse-Gamma (NIG). Comparing to

[17] A network architecture introduced in Section 2.5.3.

[18] In this context, sequential neural network refers to neural networks that are applicable for time-series applications, such as recurrent neural network (RNN), LSTM and CNN.

Ensemble and Evidential on the UCI dataset, cryptocurrency and U.S. equities time-series forecasts, we demonstrate class-leading performance on all three datasets in terms of widely used prediction performance measures. We show that only our proposed framework can provide uncertainty estimates that track realised forecast errors. On the UCI benchmark dataset, we show that our proposed framework benefits uncertainty quantification in non-time-series applications. Thus, we conjecture that some or all of our proposed improvements may benefit applications in other areas of machine learning.

In Chapter 6, we summarise the contributions provided in this thesis and discuss future research directions. In particular, we highlight several potential improvements to our work.

CHAPTER 2

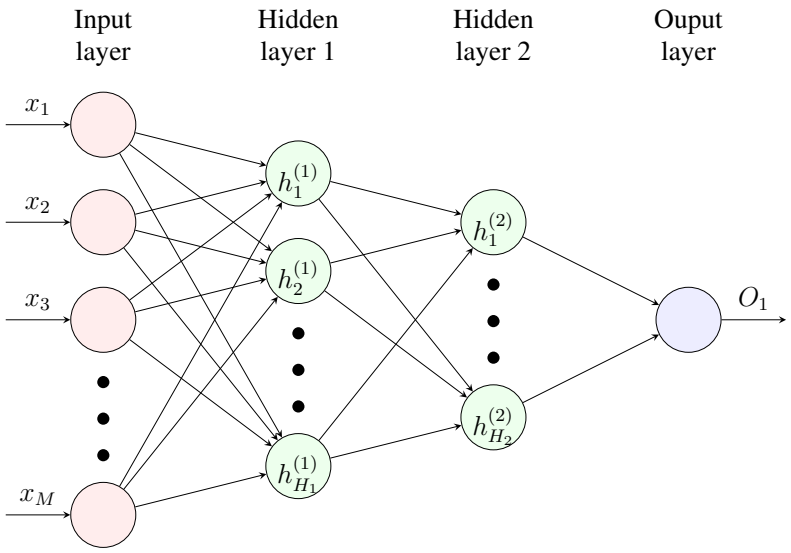
Deep learning

This thesis introduces several advances to deep learning models for applications in financial markets. As deep learning is the main apparatus of this thesis, a literature review is provided in this chapter.

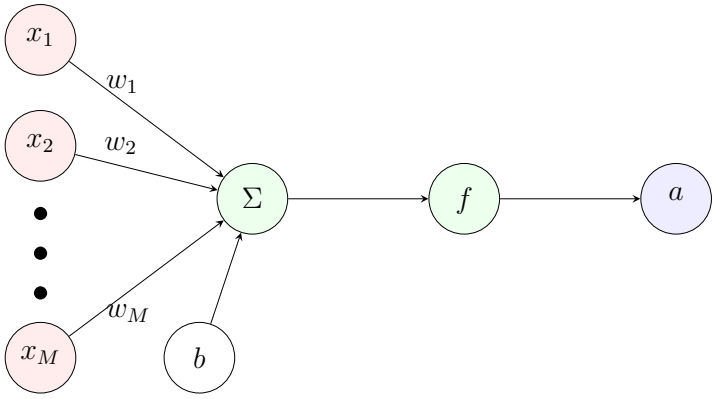
Neural networks are a broad class of high capacity models which were inspired by the biological brain and can theoretically learn any function (a property known as the *Universal Approximation Theorem*; see Hornik et al., 1989; Cybenko, 1989; Goodfellow et al., 2016). In the 19th century, studies of brain activity led to the discovery of neurons that "fire an activation" in response to some activity or stimulus (Bain, 1873; James, 1890). The introduction of the *perceptron* (Rosenblatt, 1958), *multilayer perceptron* (Rosenblatt, 1961) and the *backpropagation* training algorithm (Rumelhart et al., 1986a) heralded the beginning of neural networks. However, it is the increase in computing power and advances in training deeper networks that led to the development of modern neural networks. The term *deep learning* refers to learning with a neural network with many hidden layers, a property that is thought to contribute to the *approximation capacity* of the network (Malach et al., 2021). Further advances in neural network design and training led to the success of deep learning across multiple domains. The rest of this section presents neural network architectures used in this dissertation.

2.1 Feedforward neural networks

The simplest form of neural network, the feedforward network, also known as multilayer perceptrons (MLP), is a subset of neural networks which forms a finite acyclic graph (Goodfellow et al., 2016). There are no loop connections and values are fed forward, from the



(a) Feedforward network



(b) A neuron

FIGURE 2.1: In Figure 2.1(a), an illustration of a fully connected network with two hidden layers. Red, green and blue nodes signify input, hidden and output neurons, respectively. H_ℓ refers to the number of units in ℓ-th layer. Arrows indicate direction of flow for the output value of the respective node. Input to the network has M dimensions. In Figure 2.1(b), a single neuron (depicted as $h_j^{(\ell)}$ in Figure 2.1(a)). Input to the neuron ($\{x_1, x_2, \ldots, x_M\}$) are first linearly transformed and aggregated (depicted as Σ in the centre node). Then, a non-linear activation f is applied, leading to activation value a. More formally, the output of the neuron is $a = f(w_1x_1 + w_2x_2 + \cdots + w_Mx_M + b)$, where $w_M \in \mathbb{R}$ is network weight corresponding to input dimension M and $b \in \mathbb{R}$ is bias.

input layer to hidden layers, and to the output layer. Each layer contains one or more neurons (also called perceptrons). The operation of a neuron involves two steps. First, the input is

linearly combined with layer weights. Then, a non-linear activation function is applied on the result, as depicted in Figure 2.1(b). A feedforward network is also called a fully connected network if every node has every node in the preceding layer connected to it, as illustrated in Figure 2.1(a).

Each layer (denoted $\ell \in \{1, 2, ..., L\}$) consists of H_ℓ units (i.e., dimension of the output of the layer), activation function $f^{(\ell)}$, network weights $\boldsymbol{W}^{(\ell)} \in \mathbb{R}^{n_{\ell-1} \times n_\ell}$ and bias $\boldsymbol{b}^{(\ell)} \in \mathbb{R}^{n_\ell}$. This includes the output layer (denoted $\ell = L$). Input to the ℓ-th layer is the output of the previous layer $\boldsymbol{a}^{(\ell-1)}$. Output of the layer is computed as (in matrix form),

$$\begin{aligned} \boldsymbol{z}^{(\ell)} &= (\boldsymbol{a}^{(\ell-1)})^\mathsf{T} \boldsymbol{W}^{(\ell)} + \boldsymbol{b}^{(\ell)} \\ \boldsymbol{a}^{(\ell)} &= f^{(\ell)} \left(\boldsymbol{a}^{(\ell-1)}; \boldsymbol{W}^{(\ell)}, \boldsymbol{b}^{(\ell)} \right) = f^{(\ell)} \left(\boldsymbol{z}^{(\ell)} \right). \end{aligned} \tag{2.1}$$

where $\boldsymbol{a}^{(\ell)} = (h_1^{(\ell)}, \ldots, h_{H_\ell}^{(\ell)})$, and $\boldsymbol{z}^{(\ell)}$ is termed *weighted input*. In other words, each layer of the feedforward network performs a transformation of the input (output of the previous layer) using the specified activation function. The Universal Approximation Theorem stipulates that activation functions have the following properties (Liew et al., 2016):

(1) The output is non-constant for all ranges of inputs;
(2) Bounded within a range;
(3) Continuous over all values of point c of its domain, where $\lim_{x \to c} f(x) = f(c)$;
(4) Monotonically increasing;
(5) Differentiable everywhere, where $\lim_{c \to 0} \frac{f(x+c) - f(x)}{c}$ exists for all x.

The fifth property is not a requirement of universal approximation but is required for back-propagation learning algorithms. Popular choices of activation functions include (Goodfellow et al., 2016):

- **ReLU**: $f(x) = \max(0, x)$;
- **Sigmoid** (σ): $f(x) = \frac{1}{1+e^{-x}}$;
- **Hyperbolic tangent** ($\tanh$): $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

Sigmoid and tanh are closely related activation functions, as $\tanh(x) = 2\,\sigma(2x) - 1$. tanh resembles the identity function near 0, as the gradient is close to 1 and $\tanh(0) = 0$. Thus, when training a neural network with small activation values (i.e., close to zero), a feedforward network with tanh activation functions behaves like a linear model (Goodfellow et al., 2016). Both sigmoid and tanh are used in recurrent networks, an architecture type that we will discuss in Section 2.5.1. ReLU is a simple piece-wise activation function, where non-linearity is provided by the $\max$ function. Owing to its simplicity, unimpeded gradient flow (when the input value is positive) and ease of computation, ReLU (Jarrett et al., 2009; Nair and Hinton, 2010) quickly became the activation function of choice for many applications and has enabled the breakthrough in training state-of-the-art deep networks (Krizhevsky et al., 2012; Ramachandran et al., 2018). However, ReLU also has several weaknesses (Goodfellow et al., 2016). First, due to the $\max$ function, ReLU is discontinuous at zero. Thus, the derivative is undefined when output is exactly zero. Second, its second derivative is zero almost everywhere and there is no information pass through if the input value is negative. The zeroing out of negative values has led to a phenomenon known as "dead neurons", where a neuron can be "stuck" in a non-active state (either by random initialisation or subsequent gradient updates) as negative bias can cause the affine transformation of the input to be negative. As the neuron is always outputting 0, its weights are not updated by gradient-based learning algorithms (Maas et al., 2013). Note that both sigmoid and tanh suffer from the same problem when input is extremely positive or negative, leading to gradient that is close to zero. Third, a ReLU layer that has non-zero mean activation acts as bias for the next layer (Clevert et al., 2016). A network with many ReLU layers will thus have accumulating biases. This phenomenon is known as *bias shift* which slows down training. Various improvements to ReLU have been proposed, such as exponential linear unit (ELU) (Clevert et al., 2016),

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{Otherwise} \end{cases}, \tag{2.2}$$

and Leaky ReLU (Maas et al., 2013),

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{Otherwise} \end{cases}. \tag{2.3}$$

At the time of writing, a potential candidate for the state-of-the-art in activation function is the *Swish* function (Ramachandran et al., 2018), defined as $f(x) = x \cdot \text{sigmoid}(x)$. The Swish function is a smooth convex function that dips as it approaches 0 from the negative end, turning positive at 0 and asymptotically approaches ReLU at the infinity. The authors have reported improved performance over other variants of ReLU in image recognition tasks. Illustrative activation of each activation functions is shown in Figure 2.2.

Recall that $\boldsymbol{a}^{(0)}$ is the input layer (input data in the form of vector or matrix). For regression problems, the output layer typically has linear activation, where the output is a linear combination of the input and layer weights, and no non-linearity is applied. For classification problems,

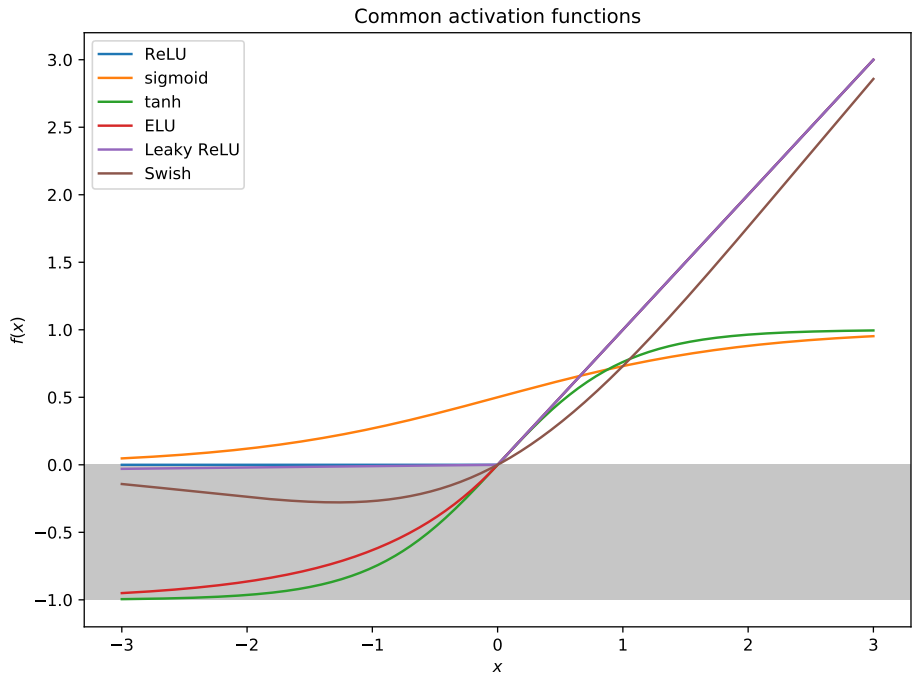


FIGURE 2.2: lllustration of *ReLU*, *sigmoid*, *tanh*, *ELU*, *leaky ReLU* and *Swish*. For ELU, $\alpha = 1$ is used. Note that Leaky ReLU is the same as ReLU where $x > 0$ and marginally negative $x < 0$. ELU and ReLU are also the same when $x > 0$. Region with $f(x) < 0$ is shaded in grey.

popular choices for output layer activation are the *sigmoid* function $\text{sigmoid} : \mathbb{R} \to (0, 1)$ (which can be used to model probability of observing a single class), or *softmax*,

$$\text{softmax}_i(\boldsymbol{x}) = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}},$$

where $\text{softmax}_i(\boldsymbol{x})$ is the predicted probability of observing class $i = 1, \ldots, C$, normalised by the probability of observing every other class.

2.2 Neural network training

In this section, we discuss the learning objective, backpropagation and optimisation methods for neural networks.

Learning, in a theoretical context, involves finding a useful approximation $\hat{G}(x)$ to the function $G(x)$ (Hastie et al., 2020), where G is function of interest and x is input to the function. In this dissertation, we are concerned with finding approximations to $\hat{G}(x)$ using a neural network. For brevity, we drop the layer designation and denote the entire network by F and weight vector set $\boldsymbol{\theta} = \bigcup_{\ell=1}^{L} \{\boldsymbol{W}^{(\ell)}, \boldsymbol{b}^{(\ell)}\}$, where L is the number of layers and $\boldsymbol{\theta}^{(\ell)} = \{\boldsymbol{W}^{(\ell)}, \boldsymbol{b}^{(\ell)}\}$ is the weight set of layer ℓ. We further denote a training instance of a *supervised learning* problem[1] as $(\boldsymbol{x}, y) \sim \mathcal{D}$, where $\mathcal{D}$ is the training set. For regression problems, both inputs $\boldsymbol{x} \in \mathbb{R}^M$ and outputs $y \in \mathbb{R}$ are real values, where M is the dimension of the input (i.e., the number of *features*). A popular choice of loss function is the mean squared error (MSE) (Goodfellow et al., 2016),

$$\mathcal{L}(F(\boldsymbol{X}; \boldsymbol{\theta}), \boldsymbol{y}) = \frac{1}{N} \sum_{j=1}^{N} (y_j - F(\boldsymbol{x}_j; \boldsymbol{\theta}))^2, \tag{2.4}$$

where $N = |\mathcal{D}|$ is number of observations in training set. For classification problems, loss is measured as terms of relative entropy between the data distribution $\text{p}_{\mathcal{D}}$ and distribution

[1] Supervised learning refers to a learning problem where the goal is to predict values of the outputs given some values of inputs, which may be measured or preset (Hastie et al., 2020).

provided by the network $\mathrm{p}_{\text{model}}$,

$$\mathrm{D}_{\mathrm{KL}}(\mathrm{p}_{\mathcal{D}}||\mathrm{p}_{\text{model}}) = \int_y \mathrm{p}_{\mathcal{D}}(y) \log \left(\frac{\mathrm{p}_{\mathcal{D}}(y)}{\mathrm{p}_{\text{model}}(y)} \right) \mathrm{d}y,$$

and is typically implemented as the sum over all training instances,

$$\mathrm{D}_{\mathrm{KL}}(\mathrm{p}_{\mathcal{D}}||F) = \sum_{i=1}^{N} y_i \log \left(\frac{y_i}{F(\boldsymbol{x}_i)} \right),$$

where the neural network $F(\boldsymbol{x}_i)$ is assumed to output an approximation to the empirical distribution $\mathrm{p}_{\mathcal{D}}$. This loss function is called Kullback-Leibler divergence (KL divergence) (Kullback and Leibler, 1951; Goodfellow et al., 2016). A related loss function for classification is *cross-entropy* (Goodfellow et al., 2016),

$$\begin{aligned} \mathrm{H}(\mathrm{p}_{\mathcal{D}}, \mathrm{p}_{model}) &= \mathrm{H}(\mathrm{p}_{\mathcal{D}}) + \mathrm{D}_{\mathrm{KL}}(\mathrm{p}_{\mathcal{D}}||\mathrm{p}_{\text{model}}) \\ \mathrm{H}(\mathrm{p}_{\mathcal{D}}) &= -\int_y \mathrm{p}_{\mathcal{D}}(y) \log \mathrm{p}_{\mathcal{D}}(y)\, \mathrm{d}y, \end{aligned} \tag{2.5}$$

where $\mathrm{H}(x, y)$ is cross-entropy between the distributions of x and y and $\mathrm{H}(x)$ is the *entropy* of the distribution of x. Cross-entropy can be interpreted as the difference between two distributions (the observed data and output of the model). For the purposes of neural network training, it can be seen that cross-entropy and KL divergence are equivalent as the entropy of the data is a constant.

Neural networks are trained using stochastic gradient descent (SGD). SGD is an iterative optimisation algorithm. The algorithm begins with a set of starting weights (weight initialisation is discussed in Section 2.3) and a randomly sampled batch of observations (termed *minibatch*). There are two reasons why a stochastic gradient is used as opposed to the full gradient over the entire training set (in which case it is simply referred to as *gradient descent* or *batch gradient descent*). First, SGD is the main way to train linear models on very large datasets (Goodfellow et al., 2016). Large training sets are typically required to achieve good *generalisation*[2]. Suppose the training set consists of $N = 1\,\text{billion}$ observations. A single

[2]Generalisation refers to prediction or classification performance on unseen data (Goodfellow et al., 2016). For a model to have good generalisation performance, the difference in performance of the model on training and out-of-sample data is minimised.

gradient step that utilises the entire training set will be very computationally expensive. If one minibatch consists of $B = 100$ observations, a single pass through the data will result in $\frac{1\,\text{billion}}{100} = 10\,\text{million}$ gradients steps. SGD relies on updating weights by the *expected gradient*, calculated over a small batch and is thus computationally inexpensive. This greatly reduces the computational cost per weight update (Goodfellow et al., 2016). Secondly, noisy gradients can help the optimiser escape *saddle points*[3] (Ge et al., 2015; Jin et al., 2017; Kleinberg et al., 2018). For highly non-convex functions with many local minima and saddle points, stochastic gradient can be interpreted as working on a smoothed, convolved version of the loss function (Kleinberg et al., 2018). Thus, SGD has greater capacity to traverse the loss surface of highly non-convex loss functions than batch gradient descent.

Next, we formally describe the SGD algorithm. Let $\mathcal{D}$ be the training set and $(\tilde{\boldsymbol{X}}_b, \tilde{\boldsymbol{y}}_b) \sim \mathcal{D}$ be the b-th randomly-drawn minibatch (where $\tilde{\cdot}$ signifies a minibatch; not to be confused with network bias $\boldsymbol{b}$). The basic stochastic gradient update equation is (Goodfellow et al., 2016),

$$\begin{aligned} \boldsymbol{\theta}_{b+1} &= \boldsymbol{\theta}_b - \eta \hat{\nabla} J(\boldsymbol{\theta}_b) \\ J(\boldsymbol{\theta}_b) &= \mathcal{L}(F(\tilde{\boldsymbol{X}}_b; \boldsymbol{\theta}_b), \tilde{\boldsymbol{y}}_b), \end{aligned} \tag{2.6}$$

where the gradient of $J(\boldsymbol{\theta}_b)$ at $\boldsymbol{\theta}_b$ is $\nabla J(\boldsymbol{\theta}_b)$, $\boldsymbol{\theta}_b$ is weight vector set at minibatch b, η is step size (also called learning rate), $\hat{\nabla}$ is stochastic gradient and $\mathcal{L}$ is the loss function. The $J(\boldsymbol{\theta}_b)$ notation is used for brevity. Equation (2.6) describes gradient update at the network level. A *forward pass* of information through the network, from the input layer, through hidden layers and to the output layer is called *forward propagation*. Once training loss of the minibatch is computed, the backpropagation algorithm (Rumelhart et al., 1986a) is used to attribute the loss to weights and bias of each layer by computing the partial derivatives of each layer. Let $c_b = J(\boldsymbol{\theta}_b)$ be the scalar loss for batch b. Then, the partial derivatives with respect to the weights and bias of the output layer are (dropping the batch subscript for clarity),

$$\hat{\nabla}^{(\ell)} = \frac{\partial c}{\partial \boldsymbol{a}^{(\ell)}} \frac{\partial \boldsymbol{a}^{(\ell)}}{\partial \boldsymbol{\theta}^{(\ell)}}. \tag{2.7}$$

[3] A saddle point is a point on the surface of the graph of a function where derivatives in orthogonal directions are all zero, but is not a local extremum of the function (Wainwright and Chiang, 2005).

We denote the *gradient vector set* attributable to the ℓ-th layer as $\hat{\nabla}^{(\ell)}$. The first partial derivative $\frac{\partial c}{\partial \boldsymbol{a}^{(\ell)}}$ is the rate of change of loss with respect to output of the final layer (i.e., $\ell = L$, the derivative of the loss function). For a regression problem where the output is a scalar, $\frac{\partial c}{\partial \boldsymbol{a}^{(\ell)}}$ is also a scalar. If the output of the network is multi-dimensional, then $\frac{\partial c}{\partial \boldsymbol{a}^{(\ell)}}$ is a vector of gradients. The second partial derivative $\frac{\partial \boldsymbol{a}^{(\ell)}}{\partial \boldsymbol{\theta}^{(\ell)}}$ is a gradient vector of the ℓ-th layer's output w.r.t. its weights. For squared loss (Equation (2.4)), the derivative is (a single instance of observation shown),

$$\frac{\partial c}{\partial a_i^{(\ell)}} = \frac{2}{B}(a_i^{(\ell)} - y_i),$$

where B is the size of a minibatch. Similarly, loss attributable to the $\ell - 1$-th layer can be computed using the chain rule,

$$\hat{\nabla}^{(\ell-1)} = \frac{\partial c}{\partial \boldsymbol{a}^{(\ell)}} \frac{\partial \boldsymbol{a}^{(\ell)}}{\partial \boldsymbol{a}^{(\ell-1)}} \frac{\partial \boldsymbol{a}^{(\ell-1)}}{\partial \boldsymbol{\theta}^{(\ell-1)}}.$$

Gradient of the network in Equation (2.6) is the collection of gradients of weights and biases of all layers,

$$\hat{\nabla} J(\boldsymbol{\theta}) = \{\hat{\nabla}^{(1)}, \hat{\nabla}^{(2)}, \ldots, \hat{\nabla}^{(\ell-1)}, \hat{\nabla}^{(\ell)}\}. \tag{2.8}$$

A single gradient update step is the simultaneous update of all layers by the gradient collection scaled by the learning rate, computed using backpropagation.

Equation (2.6) is a recursive algorithm. The network is trained iteratively using each minibatch. Minibatches are drawn from the training set without replacement until the exhaustion of the training set. One cycle through the training set is called an *epoch*. The optimal number of epochs (denoted τ) to train the network is found by monitoring loss on the validation set[4]. Training is stopped when the validation loss decreases by less than a predefined amount, called *tolerance*. This procedure is called *early stopping* (Morgan and Bourlard, 1990; Reed, 1993; Prechelt, 1998; Mahsereci et al., 2017). Algorithm 1 contains the schematics of an early stopping algorithm, adapted from Algorithm 7.1 and Algorithm 7.2 in Goodfellow et al. (2016). Early stopping can be seen as a regularisation technique which limits the optimiser to search in the parameter space near the starting parameters (Sjöberg and Ljung, 1995; Goodfellow et al., 2016), as training is terminated when training no longer decreases

[4] A validation set is a portion of data that is withheld from training and is used for hyperparameter tuning.

validation loss. The early termination of training restricts network weights to be closer to the initial values. In particular, given τ, the product $\eta\tau$ can be interpreted as the effective capacity which bounds reachable parameter space from $\boldsymbol{\theta}_0$ (network weights at the start of training), thus early stopping behaves in a similar way to L_2 regularisation (Goodfellow et al., 2016). In Chapter 3, we propose an online version of early stopping that allows the network to learn "online".

Algorithm 1 Early stopping procedure. Training stops when validation loss does not improve by at least ε for Q iterations.

Require: Maximum iterations $\{\mathcal{T} \in \mathrm{N} \,|\mathcal{T} > 0\}$; tolerance $\{\varepsilon \in \mathbb{R} \,|\varepsilon > 0\}$; patience $\{Q \in \mathrm{N} \,|Q > 0\}$; step size $\{\eta \in \mathbb{R} \,|\eta > 0\}$, training set $\{\boldsymbol{X}_{train}, \boldsymbol{y}_{train}\}$, validation set $\boldsymbol{X}_{test}, \boldsymbol{y}_{test}$

function EARLYSTOPPING($\boldsymbol{\theta}, \boldsymbol{X}_{train}, \boldsymbol{y}_{train}, \boldsymbol{X}_{test}, \boldsymbol{y}_{test}$)
  $\boldsymbol{\theta}_{best} \leftarrow \boldsymbol{\theta}$
  $q \leftarrow 0$
  $J_{best} \leftarrow \infty$
  for $k = 1, ..., \mathcal{T}$ **do**
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\hat{\nabla} J(\boldsymbol{y}_{train}, F(\boldsymbol{X}_{train}; \boldsymbol{\theta}))$
    $J' \leftarrow J(\boldsymbol{y}_{test}, F(\boldsymbol{X}_{test}; \boldsymbol{\theta}))$
    if $J' < J_{best}$ **then**
      $\tau_{best} \leftarrow k$
      $\boldsymbol{\theta}_{best} \leftarrow \boldsymbol{\theta}$
      $J_{best} \leftarrow J'$
    end if
    if J' did not improve by at least ε **then**
      $q \leftarrow q + 1$
      if $q \geq Q$ **then**
        break ▷ Assume convergence
      end if
    else
      $q \leftarrow 0$
    end if
  end for
  return τ_{best}, $\boldsymbol{\theta}_{best}$
end function

Advances in optimisation have allowed deeper and more sophisticated neural networks to be trained. In the rest of this section, we will describe, at a high level, improvements to neural network optimisation.

SGD uses a universal learning rate for all parameters of the network. However, some parts of the network may require less updates than others (e.g., if they learn rarely seen but are highly predictive features). A universal learning rate may lead to over-learning (under-learning) in parts of the network that are frequently (infrequently) updated. Parameter-specific learning rate was introduced in AdaGrad (Duchi et al., 2011), where learning rate for each parameter is scaled by the cumulative sum of the square of past gradients. This has the effect of decreasing learning rate for parameters that are frequently updated.

It was also observed that the magnitude of gradients can be very different across different parts of the network. Hinton and Tieleman (2012) proposed RMSProp, which deals with this problem by dividing the gradients by the square root of the moving average of squared gradients for each weight. This effectively standardises the gradients. Adam (Kingma and Ba, 2015) extends AdaGrad and RMSProp, computing adaptive learning rates based on estimates of first and second moments of the gradients. Rather than updating by the actual gradient $\hat{\nabla} J(\boldsymbol{\theta}_k)$, Adam updates by the exponentially weighted average of past gradients (thereby creating a momentum effect) scaled by the square root of the exponentially weighted average of squared gradients. The exponentially weighted estimates are updated recursively on each iteration, multiplied by a fixed decay rate. Kingma and Ba (2015) demonstrated faster learning (faster decrease in training loss) on benchmark datasets. This was further extended by Dozat (2016), incorporating Nesterov's accelerated gradient (also called Nesterov momentum; Nesterov, 1983) into Adam, named NAdam. The addition of Nesterov momentum has previously been shown to improve regular SGD in hard to optimise problems (Sutskever et al., 2013). At each update step, regular momentum (as used in Adam) is the weighted average between the latest gradient and the weighted average of gradients of the previous step. It was observed that a better quality gradient can be obtained by computing the latest gradient using weights updated by the previous step's weighted average gradients. This was shown to further improve optimisation speed and, at the time of writing, can be considered as the state-of-the-art in neural network optimisation.

2.3 Network weight initialisation

Weight initialisation is an area of active research and is thought to play crucial roles in enabling deeper networks and speeding up network training (Glorot and Bengio, 2010). Early neural networks used sigmoid activation with randomly initialised network weights. This type of network architecture was not conducive to training deep neural networks. Randomly initialised weights are typically drawn from a zero mean distribution (e.g., $\mathrm{U}[\frac{1}{\sqrt{H_\ell}}, \frac{1}{\sqrt{H_\ell}}]$, where H_ℓ is the number of units in the layer and is also called dimension of the layer) (Glorot and Bengio, 2010). Note that dimension of the first layer of a neural network (i.e., the input layer) has the same dimension of the input (i.e., $H_0 = M$). A neural network that uses sigmoid as activation function is prone to an accumulation of bias in a deep network. This is due to mean activation of 0.5 for the sigmoid function. Shift in the distribution of inputs to each hidden layer will cause the sigmoid function to become saturated[5], as illustrated in Figure 2.3. In here, it can be seen that if the expected value of inputs to a sigmoid function is very high or very low, its derivative asymptotically approaches 0. For this reason, Glorot

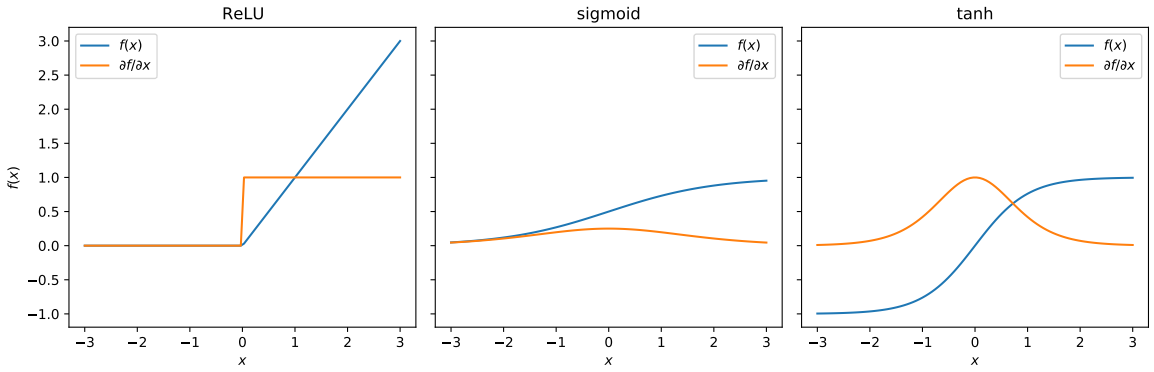


FIGURE 2.3: Illustration of *ReLU*, *sigmoid* and *tanh*, and their respective first derivatives. ReLU has a discontinuity at 0, while both sigmoid and tanh suffer from saturation at extreme values as gradient approaches 0.

and Bengio (2010) recommended against initialising network weights with small random values if the sigmoid function is used — an issue that is partially alleviated with the advent of ReLU, where gradient does not saturate for positive input values. ReLU expedited training

[5] *Saturated sigmoid* occurs when the input to the sigmoid function is close to zero. The sigmoid function outputs values that asymptotically approach zero. Thus, as gradient vanishes, training is impeded.

and allowed deeper networks to be trained[6]. Another contributing factor to slow learning is the decrease in variance of backpropagated gradient as one moves from the output layer backwards (Bradley, 2010; Glorot and Bengio, 2010). This is a deficiency of backpropagation, where loss is progressively attributed to each layer starting from the output. Variance of the gradient diminishes as loss is backpropagated through a deep network. To minimise the decrease in gradient variance attributable to weight initialisation, Glorot and Bengio (2010) proposed *normalised initialisation* (also called *Glorot initialisation* or *Xavier initialisation*), where initial weights are drawn from $\mathrm{U}\left(-\sqrt{\frac{6}{H_{\ell-1}+H_\ell}}, \sqrt{\frac{6}{H_{\ell-1}+H_\ell}}\right)$ or $\mathrm{N}\left(0, \frac{2}{H_{\ell-1}+H_\ell}\right)$ (note that $H_{\ell-1}$ is the input dimension and $M^{(\ell)}$ is the output dimension of the layer). Variance of the distribution with which weights are drawn accounts for the change in dimension between layers. This ensures that variance is constant throughout the network at the initial stage of training[7]. Derivation for the variance parameter was based on linear activation. However, the authors showed that normalised initialisation is still beneficial for networks with sigmoid or tanh activations. He et al. (2015) argued that the assumption of linear activation is invalid for a network with ReLU activation, and proposed an alternative activation (also called *He initialisation*), where weights are drawn from $\mathrm{N}\left(0, \frac{2}{M^{(\ell-1)}}\right)$. The authors demonstrated superior image classification performance using ReLU activation. In sum, weight initialisation strategy can have profound influence on network convergence and the ability to successfully train deep neural networks. Kumar (2017), Glorot and Bengio (2010) and He et al. (2015) have demonstrated the importance of maintaining stable variance of gradient across the layers. To this end, He initialisation is shown to be superior for networks with ReLU activation, while Xavier initialisation can be used for networks with sigmoid and tanh activations.

[6]However, as noted earlier in Section 2.1, this was until network architecture hit the bottleneck of ReLU. ReLU has positive expected activation. This leads to bias shift which slows down training and is addressed by ELU (Clevert et al., 2016) and other variants of ReLU that allow negative activation.

[7]However, it is not until the advent of *residual connections* that "depth barrier" of deep neural network is truly broken.

2.4 Other network architectural considerations

In addition to the aforementioned architectural considerations, modern neural networks typically also use *batch normalisation* (Ioffe and Szegedy, 2015) and *dropout* (Srivastava et al., 2014).

Batch normalisation is the transformation of the output of each hidden layer. For each minibatch with batch size B, input to the batch normalisation layer $\boldsymbol{X} \in \mathbb{R}^{B \times M}$ (i.e., output of the preceding hidden layer $\boldsymbol{a}^{(\ell-1)}$), each instance of the batch is standardised along each dimension,

$$\tilde{x}_{i,m} = \frac{x_{i,m} - \mathrm{E}[\boldsymbol{x}_m]}{\sqrt{\mathrm{Var}[\boldsymbol{x}_m]}},$$

where $x_{i,m}$ is input dimension $m = 1, \ldots, M$ of instance i of the batch and $\boldsymbol{x}_m \in \mathbb{R}^{B \times 1}$ is the column vector of the m-th dimension (batch subscript dropped for legibility). Note that mean and variance are computed per batch, not over the entire training set. The standardised values are then linearly transformed,

$$\boldsymbol{a}_m = \alpha \tilde{\boldsymbol{x}}_m + \beta,$$

where α and β are learnable parameters of the batch normalisation layer that shift and scale the standardised input. Note that a batch normalisation layer is added after a hidden layer and is thus a transformation of the output of the preceding hidden layer. Ioffe and Szegedy (2015) argued that batch normalisation reduced *internal covariate shift*, a phenomenon where the distribution of network activations changes as network parameters are updated during training. The authors argued that by fixing the distribution of activation of each hidden layer, both training speed and accuracy of classifiers improved (in benchmark image classification datasets).

However, subsequent works have disputed the working mechanism of batch normalisation. Various alternative explanations have been proposed. Santurkar et al. (2018) found that batch normalisation induces a smoother loss function surface and gradients. This surface is easier to traverse by the optimiser, allowing for faster training. Approaching the problem from a classical optimisation perspective, Kohler et al. (2019) argued that batch normalisation

works by splitting the optimisation task into optimising length and direction of the parameters separately. This was motivated by a related technique, *weight normalisation*, which normalises neural network weights to separate lengths from their directions (Salimans and Kingma, 2016). Kohler et al. (2019) demonstrated that this reparameterisation provided faster convergence. In sum, both alternative explanations relate batch normalisation to aspects of optimisation.

Dropout (Srivastava et al., 2014) involves randomly dropping out units (along with their connections) throughout the network during training. Implementation is straightforward. Each forward pass through the dropout layer (typically placed after a hidden layer) is multiplied by a 1-and-0 mask, where the zeros are randomly drawn according to the pre-specified dropout rate. Each time random dropout is applied, the masked network is a subnetwork of the full network. At inference time, the full network is used (without masking) where the weights are the result of averaging over many different subnetworks. Thus, dropout can be interpreted as an inexpensive way of ensembling within the network. Dropout was shown to improve neural network performance across a range of vision, speech recognition, document classification and computational biology tasks (Srivastava et al., 2014).

2.5 Specialised network architectures

2.5.1 Recurrent neural networks

RNN (Rumelhart et al., 1986b; Goodfellow et al., 2016) and its variants have been the workhorses of speech and languages (Chiu et al., 2018) and other sequential applications. RNNs differ from feedforward networks in that they are (directed or undirected) graphs along a temporal dimension. RNNs maintain a vector of hidden states that encode the observed sequence and are recursively updated as new observations become available. Thus, recurrent networks can be thought of as sharing parameters for each temporal step. A recurrent network can be configured to have different feedback connections, with the most basic being recurrence of the hidden state, as illustrated in Figure 2.4. In here, the hidden state of each temporal step is recursively updated. Input x_t is first combined with the hidden state of the previous step,

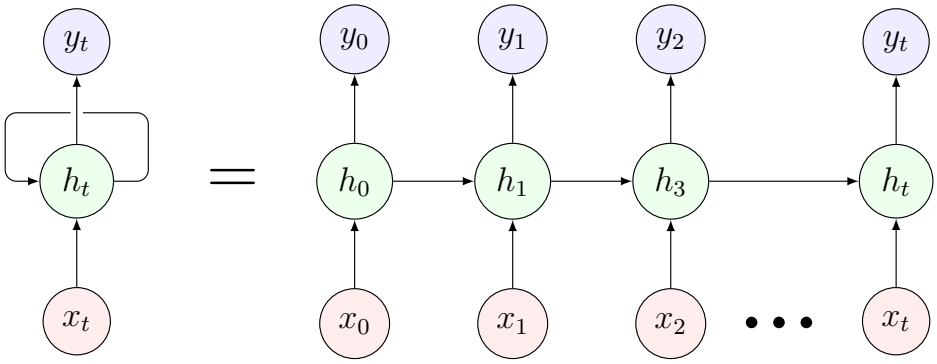


FIGURE 2.4: **Left**: An illustration of a recurrent layer with cyclical connection back to itself for each temporal step. After each activation, the hidden state is carried forward to the next temporal step. **Right**: The unfolded *computational graph* of the recurrent layer for each temporal step. In here, x, h and y signify input, hidden state and output, respectively.

resulting in an updated hidden state,

$$\boldsymbol{h}_t = f(\boldsymbol{x}_t^\mathsf{T}\boldsymbol{W}_x + \boldsymbol{h}_{t-1}^\mathsf{T}\boldsymbol{W}_h + \boldsymbol{b}_h),$$

where $\boldsymbol{W}_x$ are weights for the input, and $\boldsymbol{W}_h$ and $\boldsymbol{b}_h$ are weights and bias for the previous hidden state. The updated hidden state is then used to compute output of the layer,

$$\boldsymbol{y}_t = g(\boldsymbol{h}_t^\mathsf{T}\boldsymbol{W}_o + b_o),$$

where g, $\boldsymbol{W}_o$ and b_o are activation function, weights and bias for converting the hidden state to output.

In the example presented in Figure 2.4, the recurrent connection is hidden-to-hidden. The recurrent connection can also be output-to-hidden. However, such recurrence is less powerful as it requires that the output units capture all past information that the network uses to make predictions (Goodfellow et al., 2016). There are advantages in using specialised network architectures for sequential problems (Goodfellow et al., 2016). First, the same recurrent network can be used to model sequences of varying length, a type of input that is common in NLP problems where sentences and texts are of arbitrary length. A feedforward network requires a fixed structure and can only model sequences of pre-defined length. Second, consider a classification problem where sequences of arbitrary length K map to C different classes. If one were to model such a problem using feedforward networks, the number of

parameters in the model will scale by $\mathrm{O}(C^K)$. However, as parameters are shared across temporal steps in a recurrent network, the number of parameters in a RNN is $\mathrm{O}(1)$ with respect to sequence length (Goodfellow et al., 2016). This is also implicitly assuming that the conditional distribution of y_{t+1} is stationary given y_t.

Due to its flexibility in modelling sequences (both input and output) of arbitrary length, recurrent network architectures can be designed to suit many different applications. The different architecture types are illustrated in Figure 2.5. The one-to-one architecture corresponds to a

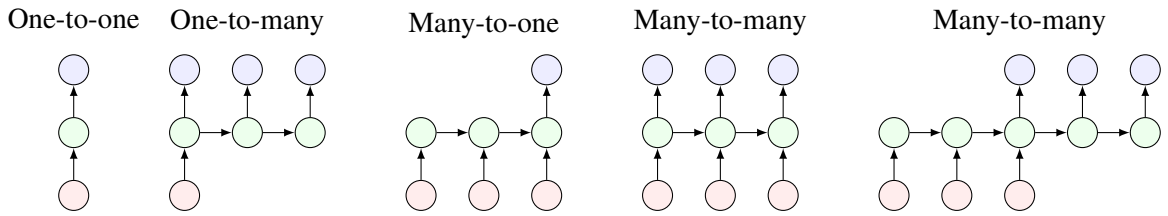


FIGURE 2.5: Types of recurrent architectures. Red, green and blue nodes signify input, hidden (recurrent) and output neurons, respectively. Each type differs by the type of association between input, recurrence and output.

conventional use of recurrent networks, such as weather forecasting (Cebeci, 2019). Similar to a pooled regression, the RNN is presented with one observation of input and makes a forecast in a one-to-one manner on each time step. The one-to-many architecture receives an encoding of a task and generates a sequence. This type of architecture have been used for music synthesis (Jaques et al., 2017). Many conventional time-series forecasting applications can be classified as many-to-one, where the network observes a sequence of inputs and makes a classification or prediction. Financial time series forecasting problems such as Li et al. (2017) and those in Chapter 4 are examples of many-to-one networks. There are two types of many-to-many architectures — one where inputs and outputs are in sync along the time dimension; and where outputs lag inputs by an offset. The former is used in applications such as named entity recognition (Katiyar and Cardie, 2018), while the latter is used in machine translation, such as the *Seq2seq* model (Sutskever et al., 2014). In Seq2seq, the encoder part of the recurrent network converts the sentence to be translated into a latent representation, which is then re-generated by the decoder portion of the network in the target language.

However, despite their success, recurrent networks are notoriously difficult to train (Pascanu et al., 2013; Bai et al., 2018). In an unrolled computational graph, recurrent networks can be considered as neural networks with unlimited depth. Modelling long term dependencies is especially challenging as gradients propagated through many layers tend to explode or vanish (Bengio et al., 1994; Pascanu et al., 2013). Even if we assume that gradients remain stable, long-term dependencies are difficult to maintain as recursively multiplied Jacobians lead to exponentially smaller weights given to long-term interactions (Goodfellow et al., 2016). To address this, LSTM networks (Hochreiter and Schmidhuber, 1997; Goodfellow et al., 2016) were introduced with a *memory cell* containing a self-loop that allows gradients to flow over long durations. The basic schematics of a LSTM is illustrated in Figure 2.6. *Gating* is central

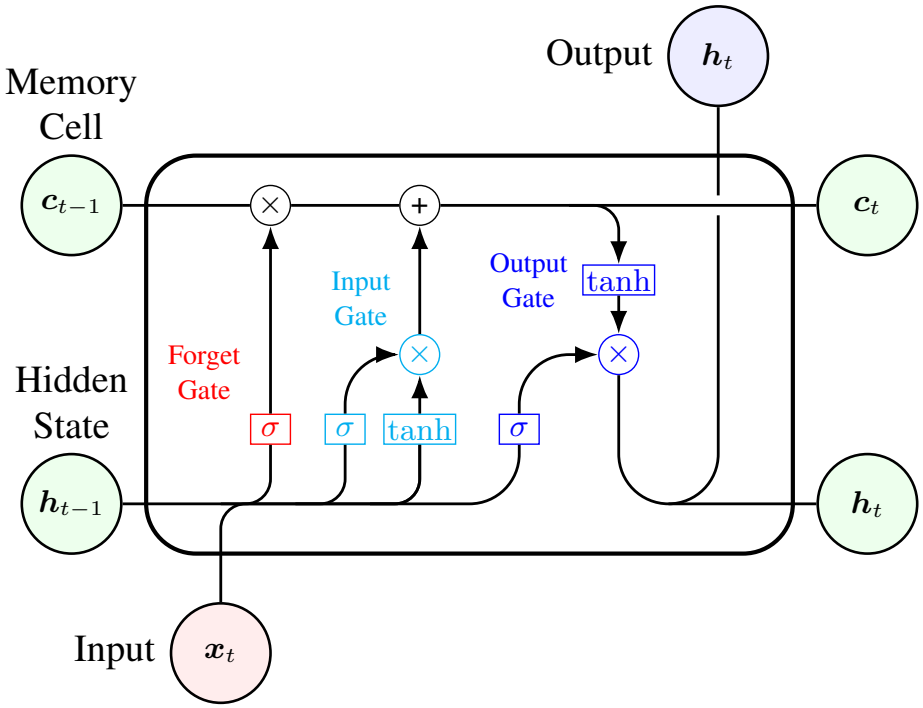


FIGURE 2.6: An illustration of a long short-term memory "cell". The forget gate controls whether information stored in the memory cell is retained. The input gate controls whether the new input is stored into the memory cell. The output gate incorporates information in the memory cell into the output.

to the LSTM, which consists of three gates: the *forget gate*, the input gate (also called update gate) and the output gate. The forget gate controls how much memory from the previous time

step is retained[8],

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}^{(f)}\boldsymbol{x}_t + \boldsymbol{U}^{(f)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(f)}),$$

where $\boldsymbol{f}_t$ is the forget gate unit at time step t, σ denotes the sigmoid function, $\boldsymbol{W}$, $\boldsymbol{U}$ and $\boldsymbol{b}$ are network weights applied the input and hidden state, and bias, respectively. The forget gate combines the input and hidden state into a $\boldsymbol{f}_t \in (0,1)^H$ vector, where H is the dimension of the LSTM unit. The input gate $\boldsymbol{i}_t \in (0,1)^H$ controls how much information is incorporated into the memory cell,

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}^{(i)}\boldsymbol{x}_t + \boldsymbol{U}^{(i)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(i)}).$$

The input and hidden state are transformed into a cell input vector $\tilde{\boldsymbol{c}}_t \in (-1,1)^H$,

$$\tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}^{(c)}\boldsymbol{x}_t + \boldsymbol{U}^{(c)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(c)}).$$

Memory carried forward from $t-1$ is modulated by the forget gate and the cell input vector is modulated by the input gate. The two are then combined to form the updated memory cell state,

$$\boldsymbol{c}_t = \boldsymbol{f}_t \otimes \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \otimes \tilde{\boldsymbol{c}}_t,$$

where $\otimes$ is the Hadmard product (i.e., element-wise multiplication). The output gate $\boldsymbol{o}_t \in (0,1)^H$ controls the amount of information from the memory cell that is incorporated into the hidden state,

$$\begin{aligned}\boldsymbol{o}_t &= \sigma(\boldsymbol{W}^{(o)}\boldsymbol{x}_t + \boldsymbol{U}^{(o)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(o)}) \\ \boldsymbol{h}_t &= \boldsymbol{o}_t \otimes \tanh(\boldsymbol{c}_t).\end{aligned} \tag{2.9}$$

The updated hidden state $\boldsymbol{h}_t$ is the output of the LSTM cell at t. Advances in LSTM (and other recurrent networks) had enabled major breakthroughs in machine translation (Sutskever et al., 2014), speech recognition (Graves et al., 2013) and handwriting recognition (Graves et al., 2009).

[8] In this section, $\boldsymbol{f}$ is used to denote the forget gate, not to be confused with activation function f used throughout this dissertation.

2.5.2 Temporal convolutional networks

CNNs are specialist networks for data that has a known, grid-like topology (Goodfellow et al., 2016). Examples include time-series of fixed length (a 1-D grid) or images (2-D grid). CNN has achieved human-like accuracy in image recognition tasks (Krizhevsky et al., 2012; Szegedy et al., 2015; Schroff et al., 2015). In this section, the mathematical operation *convolution* and a CNN-derived network achitecture known as temporal convolutional network (TCN)[9] are discussed. We refer readers to Goodfellow et al. (2016) for a comprehensive discussion on general CNNs.

Convolution is the modification of one function by another, producing a third function. More formally, the convolution of functions[10] f and g results in function o indexed by i (Goodfellow et al., 2016),

$$o(i) = \int \mathrm{f}(j)\,\mathrm{g}(i-j)\,\mathrm{d}j \tag{2.10}$$
$$o(i) = (\mathrm{f} * \mathrm{g})(i),$$

where $*$ is the convolution operator. To put it more concretely, suppose $\mathrm{g}(\cdot)$ in Equation (2.10) is a weighting function (e.g., a probability density function). Then, $\mathrm{f} * \mathrm{g}$ results in a function that returns the weighted average of f by g. In this case, f is the *input* and g is the *kernel* (also known as *filter*). In image recognition, the input is typically 2-D with discrete index (Goodfellow et al., 2016),

$$O(i,j) = \sum_m \sum_n \mathrm{F}(m,n)\,\mathrm{G}(i-m, j-n).$$

An example of convolution is illustrated in Figure 2.7. This example uses *one* 2-D kernel, with *kernel size* of 2. Size of the kernel relates to how much local information is used each time the kernel is convolved with the input. Each value in output O is the result of a sum-product of four adjacent values (2×2) of input F and kernel G. For time-series input, a 1-D kernel (e.g., with dimensions 2×1) is used which slides along the sequence during convolution. Number

[9] Also known as *dilated convolutional networks*.

[10] Note that in this section, we use (upright) f and g symbols and their capitalised counterparts to illustrate the workings of convolution. Not to be confused with the meaning of f (activation function) in the rest of the thesis.

of kernels[11] is analogous to number of units of a hidden layer in a feedforward network, and dictates the output dimension of the convolution layer. Suppose that number of kernels is 3. Then, G has dimensions $2 \times 2 \times 3$ (three 2×2 kernels stacked together) and convolution is performed three times, resulting in $O \in \mathbb{R}^{3\times3\times3}$. In Figure 2.7, size of the resultant matrix shrank by 1. Various zero-padding strategies exist if the desire is to maintain the dimensions of the output, such as by padding the surroundings of the input with 0.

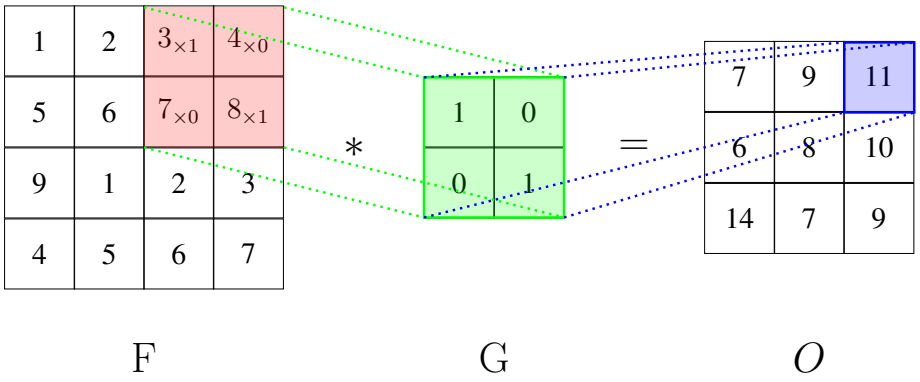


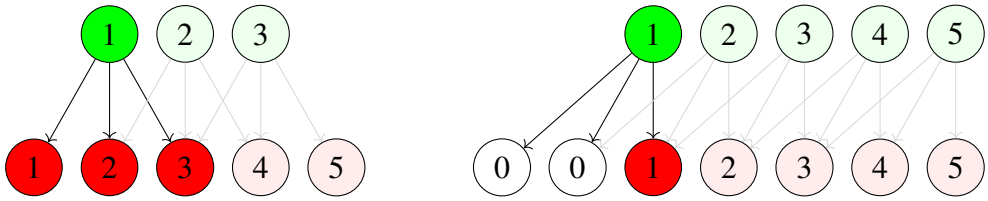
FIGURE 2.7: An illustration of a convolution operation. Input F is convolved with kernel G, producing O. The highlighted cells is a single convolution operation, yielding $3 \times 1 + 4 \times 0 + 7 \times 0 + 8 \times 1 = 11$. The same operation is repeated horizontally and vertically.

Given that regular feedforward networks are underpinned by universal approximation properties, one may wonder why CNNs are required for image recognition problems. Suppose an image has dimensions of 1000×1000 pixels and three colour channels per pixel. The resulting input tensor contains 3 million values. Next, suppose a feedforward network is used for this problem and the first hidden layer contains 500 units. The resultant number of connections at the first hidden layer is $3\,\text{million} \times 500 = 1.5\,\text{billion}$. Such a parameter space is likely infeasibly large to be trained efficiently. CNN solves this problem by utilising three important concepts: *sparse interactions*, *parameter sharing* and *equivariant representations* (Goodfellow et al., 2016). Sparse interactions (also known as *sparse connectivity*) is achieved by using a kernel that is much smaller than the input dimensions. Memory requirement and efficiency can be significantly improved if small, meaningful features can be detected out

[11] Also called number of filters or number of channels. The latter due to it being analogous to colour channels in image applications.

of thousands of pixels. The same small (in dimensions) kernel is used throughout the layer, leading to parameter sharing. This encourages the network to learn a kernel that can extract useful features that are common across the input and greatly reduces the parameter space. This also leads to equivariance, where the same feature appearing in different location in an image (or at different times in a time-series) will lead to the exact same output, just at a different location (different point in time). Thus, only one representation needs to be learned and can detect the same feature anywhere in the input.

As noted in Section 2.5.1, recurrent neural networks are difficult to train and suffer from exploding/vanishing gradient that renders modelling long term dependencies challenging. Sequential processing also makes scaling up computation to take advantage of recent advances in parallel hardware acceleration difficult (Bai et al., 2018). Recently, convolutional networks have been shown to be competitive against recurrent network-based architectures in a range of sequence learning tasks (van den Oord et al., 2016; Kalchbrenner et al., 2016; Bai et al., 2018). van den Oord et al. (2016) proposed a novel CNN architecture, known as *WaveNet* (also known as *dilated convolutional network*), and showed that it generated more naturally sounding speech than LSTM. The main components of WaveNet consist of dilated convolution layers with causal padding (termed *causal convolutions* in van den Oord et al., 2016) and residual connections (He et al., 2016). In standard convolution, the filter convolves with a symmetrical number of values to the left and right of the centre of the filter (as depicted in Figure 2.7). This is problematic in time-series as values on the right represent future observations that would not have been known at the centre. This is mitigated with causal padding which shifts the sequence such that the rightmost value of each convolution corresponds to the centre of the original sequence. Causal padding refers to appending $d \times (k-1)$ zeros to the beginning of the sequence, where d is dilation rate and k is kernel size. This is illustrated in Figure 2.8. In standard convolution (Figure 2.8(a)), the illustration depicts the first convolution of a kernel of size 3 with the first 3 elements of the input sequence. The first output value uses up to the third value from the input sequence and is thus "looking ahead in time." In causal convolution (Figure 2.8(b)) and assuming a dilation rate of 1, $1 \times (3-1) = 2$ zeros are padded to the beginning (left) of the input sequence. The kernel is convolved with the first three elements of the padded sequence (two leading zeros and the first element of the original sequence),



(a) Standard convolution (b) Causal convolution

FIGURE 2.8: In the top row, green nodes denote outputs after convolution with a kernel of size 3, numbered by the n-th convolution operation (striding from left to right). In the bottom row, red nodes denote the input sequence, numbered by the n-th element in the sequence. White nodes denote zero padding. Darker colours denote nodes undergoing convolution. In Figure 2.8(a), the 3rd convolution operation will access the 4th element in the input.

producing the first output value. Thus, the output contains no look ahead information. Note that due to the two padded zeros, causal convolutions produce a longer output sequence than standard convolution with no padding. Standard convolution can also have zeros padded to both beginning and end of the sequence to maintain length of the sequence if required. Dilated convolution (also called *à trous convolution*) expands the *receptive field* of the node by skipping input nodes (i.e., inserting zeros to the kernel). For example, suppose that a network with dilation rates are multiples of 2 and kernel size of $k = 3$, as depicted in Figure 2.9. The first convolution layer in Figure 2.9 has dilation rate of $2^0 = 1$, the second is $2^1 = 2$, third is $2^2 = 4$, and so on. In each dilated convolution layer, $d - 1$ zeros are added in between each value in the kernel (dilation rate of 1 equates to no dilation). For example, the third convolution layer in Figure 2.9 has $2^2 - 1 = 3$ zeros inserted between each value of the kernel, with receptive field that spans the entire sequence of 15 elements. Values that are multiplied by 0 in the kernel (depicted using dashed grey connections) do not contribute to the final result. Thus, the network can cover sequences of arbitrary length by stacking multiple dilated convolution layers and having $d \geq 2$.

Bai et al. (2018) proposed the TCN as a generalisation of WaveNet — without conditioning, context stacking or gated activation. However, it retains the key ingredient of dilated causal convolution layers organised into *residual blocks*, first introduced in ResNet (He et al., 2016) and is Illustrated in Figure 2.10. Each residual block in Figure 2.10 consists of four layers — dilated causal convolution, batch normalisation, spatial dropout (Tompson et al., 2015) and

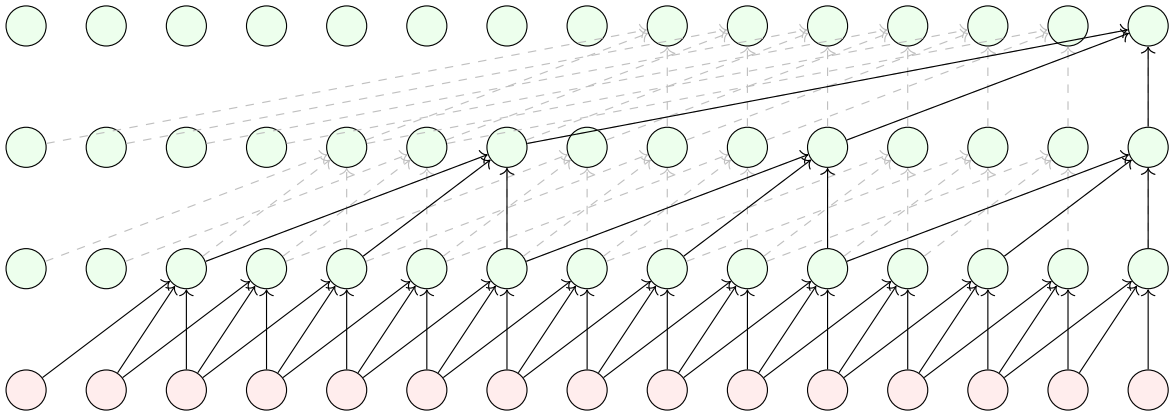


FIGURE 2.9: Causal dilated convolutions with kernel size of 3 and dilation rate of multiples of 2. Red nodes represent the input sequence and the two green layers represent convolution layers. Nodes without a connection or are connected with a dashed grey line are multiplied by zeros in the kernel. The top (output) node reaches the entire input sequence without using every node in the intermediate layer. The receptive field can be varied by increasing or decreasing the number of dilated convolutional layers.

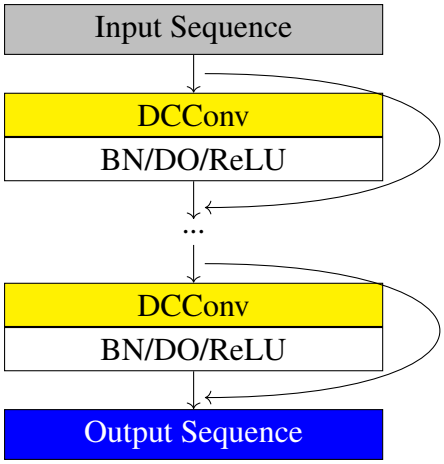


FIGURE 2.10: Illustration of a TCN. Residual blocks correspond to green layers in Figure 2.9 and consist of skip connection, dilated causal convolution (abbrev. *DCConv*), batch normalisation , spatial dropout and rectified linear unit activation layers (abbrev. *BN/DO/ReLU*).

ReLU. Spatial dropout is similar to the regular dropout (introduced in Section 2.4). However, instead of randomly dropping out observations, spatial dropout randomly drops an entire dimension. The output sequence of a residual block is first added to the input sequence (of the said residual block) via skip connection, then feed into the next residual block as

input sequence. Bai et al. (2018) showed that TCN was superior to LSTM across a range of sequence modelling tasks. The authors also noted five advantages of TCN (along with two disadvantages) over recurrent networks, most notably, parallelism and stable gradients. Stable gradients allow convolutional networks to access longer history (one year of daily prices is over 250 observations) than recurrent networks. TCN provides another useful tool for modelling time-series data.

2.5.3 Autoencoders

An autoencoder is an unsupervised neural network that learns a latent representation of the input (Goodfellow et al., 2016). First introduced in LeCun (1987); Bourlard and Kamp (1988); Hinton and Zemel (1993), the autoencoder consists of two parts, an *encoder* that encodes the input into a latent representation $\boldsymbol{h} = f(\boldsymbol{x})$ and a decoder that reconstructs the input $\hat{\boldsymbol{x}} = g(\boldsymbol{h})$. Each of encoder and decoder contains one or more hidden layers. Typically, dimensions of $\boldsymbol{h}$ is chosen to be significantly smaller than the dimension of $\boldsymbol{x}$ such that the autoencoder learns a useful representation that summarises the input, as illustrated in Figure 2.11. Autoencoder learns the latent representation of input via hidden layers, and is equivalent to PCA when it has precisely one linear hidden layer (Baldi and Hornik, 1989; Le et al., 2018). Thus, traditional applications of autoencoders are in dimensionality reduction and generative modelling.

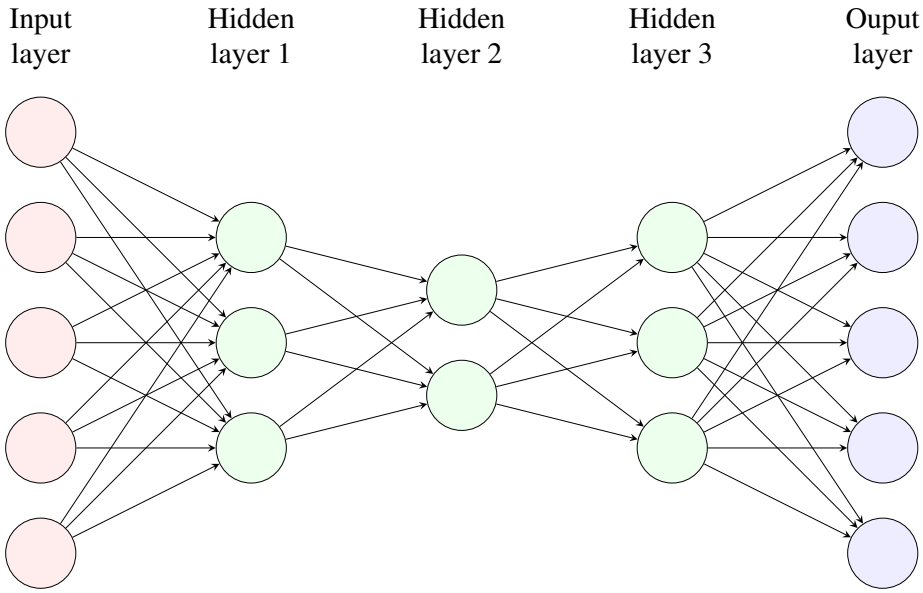


FIGURE 2.11: An illustration of an autoencoder. The input $\boldsymbol{x} \in \mathbb{R}^5$ is converted into a latent representation $\boldsymbol{h} \in \mathbb{R}^2$ using 2 hidden layers. $\boldsymbol{h}$ is then converted back into the (reconstructed) input.

CHAPTER 3

Time-varying neural network for stock return prediction

The motivating application of this chapter is in predicting cross-sectional stock returns in a portfolio context and has been discussed in Section 1.6.1. We propose the OES algorithm and show that a neural network trained using this algorithm can track a function changing with unknown dynamics. We provide a regret-bound for the algorithm and show that the worst-case tracking performance of the algorithm is bound by the time-variations of the DGP. We compare the proposed algorithm to current approaches on predicting monthly U.S. stock returns and show its superiority. Using this algorithm, we also provide evidence that supports recent findings in finance literature that suggest financial markets are time-varying. These contributions have resulted in the following publication:

Steven Y. K. Wong, Jennifer S. K. Chan, Lamiae Azizi, Richard Y. D. Xu, "Time-varying neural network for stock return prediction," *Intelligent Systems in Accounting, Finance and Management*, 29(1), 3–18, 2022.

3.1 Introduction

At every interval, an investor forecasts expected return of assets and performs security selection. This problem is closely related to *asset pricing* (as discussed in Section 1.3 and Appendix A1), cross-sectional predictions and time-varying DGP. To reiterate, literature has found hundreds of factors (Harvey et al., 2016) that can predict stock returns. Literature has also documented evidence of time-variability of the DGP (e.g., Pesaran and Timmermann, 1995; Bossaerts and Hillion, 1999). Both the true functional-form of the DGP and its time-varying dynamics are not known.

To address this, we simultaneously address both challenges by proposing the Online Early Stopping (OES) algorithm, which allows neural networks to adapt to a time-varying function. Our problem is characterised by information release over time and iterative decision making. This shares a remarkably similar setup with *online optimisation*[1]. Optimisation in this context is called *online* as decisions are made with past information but not the future. As noted in Section 2.2, one of the hyperparameters in batch (offline) neural network training is the number of optimisation epochs τ. In OES, we propose to treat τ as a learnable parameter that varies over time (t), as τ_t, and is recursively estimated over time. We provide τ_t with a new meaning — a regularisation parameter that controls the amount of update neural network weights receive as new observations are revealed. Thus, if consecutive cross-sectional observations are very different then we would expect τ_t to be relatively small and the neural network is prevented from overfitting to any one period. Conversely, a slowly changing function will have a high degree of continuity and we would expect the network to fit more tightly to each new observation. Using this training algorithm, a neural network can adapt to changes in the DGP over time. For practitioners, we show that a neural network trained with OES can be a powerful prediction model and a useful tool for understanding the time-varying drivers of returns.

Neural network training is an optimisation problem (Evens et al., 2021). We draw on concepts in online optimisation to provide a performance bound that is related to the variability of each period. We do not assume any time-varying dynamics of the underlying function, a typical approach in online optimisation[2]. The benefit of this approach is that it can track any source of variability in the underlying function, including macroeconomic, arbitrage-induced, market condition-induced, or other unknown sources. For instance, Lev and Srivastava (2019) suggested that the negative return to the value factor was related to diminishing relevance of book equity as an accounting measure. Such drivers would not have been captured by the macroeconomic approach in Gu et al. (2020). Nonetheless, we acknowledge that a limitation of our approach is the difficulty in explaining the source of variability. Attributing the source

[1] We formally describe the context in Section 3.2.1.

[2] Online convex optimisation is typically formulated as a game against an adversary, where solutions are designed to provide worst-case performance guarantees. See (Shalev-Shwartz, 2012) for an overview.

requires measuring interactions between features and sources, of which some may be difficult to quantify. Existing model explanation methods (e.g., Local Interpretable Model-Agnostic Explanations; Ribeiro et al., 2016) attribute model output to input through approximations. Thus, the addition of interaction terms between each feature to each source could lead to a substantial increase in dimensionality.

We provide two evaluations of OES: 1) a simulation study based on a dataset simulated from a non-linear function evolving under a random-walk; 2) an empirical study of U.S. stock returns. The empirical study is based on Gu et al. (2020), who compared several machine learning algorithms for predicting monthly returns of all U.S. stocks. The best performing machine learning model in Gu et al. (2020) as measured by R^2 (this measure is discussed in more detail in Section 3.2.1) was a three-layer fully connected neural network with ReLU activation, dropouts and batch normalisation. The network was trained annually using all available data up to t. The same network is then used for monthly prediction over the next 12 months, upon which an additional 12 months of new data is added to the training set and training is repeated. The majority of the dataset were made available to the public and are used in this chapter. We note that the setup in Gu et al. (2020) is suboptimal for our portfolio selection problem for three reasons. Firstly, (raw) monthly stock returns contain characteristics that complicate the forecasting problem, such as outliers, heavy tails, and volatility clustering (Cont, 2001). These characteristics are likely to impede a predictor's ability to learn. Secondly, the dataset in Gu et al. (2020) contains stocks with very low market capitalisation, are illiquid, and are unlikely to be accessible by institutional investors. Thirdly, at the individual stock level, forecasting stocks' excess returns over risk free rate also encompasses forecasting market excess returns. As practitioners are typically concerned with relative performance between stocks[3], the market return component adds unnecessary noise to the problem of relative performance forecasting. Thus, in addition to comparison with Gu et al. (2020), we also present results based on a more likely use case by practitioners, by excluding stocks with very low capitalisation and forecasting cross-sectionally standardised excess returns. We show that forecasting performance significantly improved based on this

[3]In the simplest form, a long-only investor will hold a portfolio of the top ranked stocks and a long-short investor will buy top ranked stocks and sell short bottom ranked stocks. Thus, relative performance is relevant to practitioners.

re-formulation. We propose to measure performance using the information coefficient (IC), a widely applied performance measure in investment management (Ambachtsheer, 1974; Grinold and Kahn, 1999; Fabozzi et al., 2011b). OES achieves an IC of $4.58\,\%$ on the U.S. equities dataset, compared to $3.82\,\%$ under an expanding window approach in Gu et al. (2020). For context, an IC of $5\,\%$ in predicting cross-sectional returns is considered as "good" by practitioners (Grinold and Kahn, 1999).

A summary of our contributions in this chapter is as follows:

- We propose the OES algorithm which allows a neural network to track a time-varying function. OES can be applied to existing network architectures and requires significantly less time to train than the expanding window approach in Gu et al. (2020). In our tests, OES took 1/7 the time to train and predict as compared to the expanding window approach of expanding window neural network (EWNN)[4]. This finding has a practical implication as practitioners wishing to employ deep learning models have limited time between market close and next day's open to generate features and train new models, which is made worse if an ensemble is required.
- We show that firm features exhibit time-varying importance and that the model changes over time. We find that some prominent features, such as market capitalisation (the size effect) display declining importance over time. This finding is consistent with McLean and Pontiff (2016) and highlights the importance to consider time-varying models.
- We find that firm features, in aggregate, experience a fall in importance in predicting cross-sectional returns during market distress (e.g. Dot-com bubble in 2000–01). The importance of sector dummy variables (e.g., technology and oil stocks) rose over the same period, suggesting the importance of sectors is also time-varying. Our analysis indicates that sectors have an important role in predicting stock returns during market distress. We expect this to be especially true if market stress impacts certain sectors more than others, such as travel and leisure stocks during a pandemic.

[4]Training performed on AMD Ryzen™ 7 3700X, Python 3.7.3, Tensorflow 1.12.0 and Keras 2.2.4.

- Using a subuniverse that is more accessible to institutional investors (by excluding microcap stocks), we show that OES exhibits superior predictive performance. We find that the mean correlation between predictions of OES and EWNN is only $35.9\,\%$ and the monthly correlation is lowest immediately after a shock (e.g., recession). We attribute this to OES adapting to the recovery which manifests as lower drawdown post the Global Financial Crisis.
- We show that an ensemble formed by averaging the standardised predictions of the two models exhibits the highest IC, decile spread and Sharpe ratio. Thus, practitioners may choose to deploy both models in a complementary manner.

In the rest of this chapter, we denote the algorithm of Gu et al. (2020) as expanding window neural network (EWNN) and our proposed Online Early Stopping algorithm as OES. This chapter is organised as follows. Section 3.2 defines our cross-disciplinary problem and provides an overview of online optimisation. Section 3.3 outlines our main contribution of this chapter — the proposed OES algorithm which introduces time-variations to the neural network. Simulation results are presented in Section 3.4, which demonstrates the effectiveness of OES in tracking a time-varying function. An empirical study on U.S. stock returns is outlined in Section 3.5. Finally, Section 3.6 discusses the empirical finance problem and concludes the chapter with some remarks.

3.2 Preliminaries

3.2.1 Problem setup

The setup of the problem in this Chapter largely follows that of the cross-sectional prediction problem defined in Section 1.6.1 and is repeated in here for convenience.

Similar to a classical online learning setup, a player iteratively makes portfolio selection decisions at each period. We call this iterative process *per interval training*. There are N stocks in the market, each with M features, forming input matrix $\boldsymbol{X}_t \in \mathbb{R}^{N\times M}$ at time $t = 1, \ldots, T$. The i-th row in $\boldsymbol{X}_t$ is feature vector $\boldsymbol{x}_{t,i}$ of stock i. To simplify notations, we define

return of stock i as the percentage return over the next period, i.e., $y_{t,i} = (p_{t+1,i} + d_{t+1,i})/p_{t,i} - 1$, where $p_{t,i}$ is price at time t and $d_{t,i}$ is dividend at t if a dividend is paid, and zero otherwise. In other words, the player uses information up to time t (e.g., earnings-to-price ratio at t) to predict a stock's return over t to $t+1$. Player predicts stock returns $\hat{\boldsymbol{y}}_t \in \mathbb{R}^N$ by choosing $\boldsymbol{\theta}_t \in \Theta$, which parameterises prediction function $\{F : \mathbb{R}^{N \times M} \mapsto \mathbb{R}^N; \hat{\boldsymbol{y}}_t = F(\boldsymbol{X}_t; \boldsymbol{\theta}_t)\}$. Market reveals $\boldsymbol{y}_t$ and, for regression purposes, player incurs squared loss,

$$J_t(\boldsymbol{\theta}_t) = \frac{1}{N} \sum_{i=1}^{N} (y_{t,i} - \hat{y}_{t,i})^2.$$

This general iterative portfolio selection setup is shared across Chapter 4, 5 and this chapter. The true function $\Phi_t : \mathbb{R}^{N \times M} \mapsto \mathbb{R}^n$ drifts over time and is approximated by F with time-varying $\boldsymbol{\theta}_t$. Player's objective is to minimise loss incurred by choosing the best $\boldsymbol{\theta}_t$ at time t using observed history up to $t-1$. Both the functional form and time-varying dynamics of Φ_t are not known Hence a neural network is used to model the cross-sectional relationship at each t and the time-variability is formulated as a network weights tracking problem. The loss function J_t verifies the same assumptions adopted in Aydore et al. (2019), which are:

- J_t is bounded: $|J_t| \leq D; D > 0$,
- J_t is L-Lipschitz: $|J_t(\boldsymbol{a}) - J_t(\boldsymbol{b})| \leq L \|\boldsymbol{a} - \boldsymbol{b}\|; L > 0$,
- J_t is β-smooth: $\|\nabla J_t(\boldsymbol{a}) - \nabla J_t(\boldsymbol{b})\| \leq \beta \|\boldsymbol{a} - \boldsymbol{b}\|; \beta > 0$.

We denote the gradient of J_t at $\boldsymbol{\theta}_t$ as $\nabla J_t(\boldsymbol{\theta}_t)$ and stochastic gradient as $\hat{\nabla} J_t(\boldsymbol{\theta}_t) = \mathrm{E}[\nabla J_t(\boldsymbol{\theta}_t)]$, or where the context is obvious, ∇_t and $\hat{\nabla}_t$ respectively.

As performance measure, Gu et al. (2020) used pooled R^2_{oos} without mean adjustment in the denominator,

$$R^2_{oos} = 1 - \frac{\sum_{(t,i) \in \mathcal{D}_{oos}} (y_{t,i} - \hat{y}_{t,i})^2}{\sum_{(t,i) \in \mathcal{D}_{oos}} y^2_{t,i}}, \tag{3.1}$$

where $\mathcal{D}_{oos}$ is the pooled out-of-sample dataset covering January 1987 to December 2016 in the empirical study. There are several shortcomings with this performance measure. The number of stocks in the U.S. equities dataset starts from 1,060 in March 1957, peaks at over 9,100 in 1997 and falls to 5,708 at the end of 2016. A pooled performance metric will place more weight on periods with a higher number of stocks. An investor making iterative portfolio

allocation decisions would be concerned with accuracy *on average over time*. Moreover, asset returns are known to exhibit non-Gaussian characteristics (Cont, 2001). Summary statistics of monthly U.S. stock returns are provided in Table 3.2 (in Section 3.5), which confirms the existence of considerable skewness and time-varying variance. Therefore, we provide three additional metrics. The first metric is the IC, defined as the cross-sectional Pearson's correlation[5] between predictions and actual returns:

$$\text{IC} = \frac{1}{T}\sum_{t=1}^{T}\text{IC}_t, \quad \text{IC}_t = \rho(\boldsymbol{y}_t, \hat{\boldsymbol{y}}_t). \tag{3.2}$$

The time-series of IC is averaged to give the final score, which we refer to as mean IC (Equation (3.2)). Note that even though correlation is a score rather than a percentage, IC is typically expressed as a percentage by practitioners. IC was first proposed by Ambachtsheer (1974) and is widely applied in investment management for measuring predictive power of a forecaster or an investment strategy (Grinold and Kahn, 1999; Fabozzi et al., 2011b). The second metric is the annualised Sharpe ratio, calculated as,

$$\text{SR} = \frac{12 \times \text{E}[P_{t\in\mathcal{D}_{oos}}]}{\sqrt{12 \times \text{Var}[P_{t\in\mathcal{D}_{oos}}]}}, \tag{3.3}$$

where $P_{t\in\mathcal{D}_{oos}}$ is the decile return spread. Decile return spread is computed by first sorting stocks into deciles based on their predicted return at t, where decile 10 (decile 1) contains stocks with the highest (lowest) predicted return. The average return for each decile is then computed. Decile return spread is the difference between monthly returns of decile 10 and decile 1 at t,

$$P_t = \frac{10}{N}\left[\sum_{i=1}^{N} y_{t,i}\delta_{t,i}^{(10)} - \sum_{i=1}^{N} y_{t,i}\delta_{t,i}^{(1)}\right], \tag{3.4}$$

where $\delta_{t,i}^{(d)}$ is the decile indicator and $\hat{y}_t^{(d)}$ is the boundary of the d-th decile of all $\hat{y}_{t,i}$ sorted in ascending order,

$$\delta_{t,i}^{(d)} = \begin{cases} 1 & \text{if } \hat{y}_t^{(d-1)} < \hat{y}_{t,i} \leq \hat{y}_t^{(d)}, \\ 0 & \text{otherwise.} \end{cases}$$

Note that in the experiment results in Section 3.5, we use the label P10-1 for the average decile return spread computed over all t, rather than for a single t in Equation (3.4). The

[5] Rank IC, which uses Spearman's rank correlation instead of Pearson's, is also used in practice.

decile return spread and its associated Sharpe ratio are commonly used to measure economic performance of an investment strategy and are used in Gu et al. (2020). The third metric is the average monthly R^2, where denominator is adjusted by the cross-sectional mean, as a conventional complement to R^2_{oos}.

3.2.2 Neural network training under concept drift

A detailed description of neural networks is provided in Chapter 2, including a description of the standard early stopping algorithm in Section 2.2. In the classical early stopping algorithm, a randomly drawn portion of data is used for validation. Training is stopped when validation loss decreases by lower than a predefined amount. Given optimisation steps τ, the product $\eta\tau$ can be interpreted as the effective capacity which bounds reachable parameter space from starting weights, thus behaving like L_2 regularisation (Goodfellow et al., 2016). A discussion of aspects of neural network training that is relevant to online problems is presented in this section.

For time series problems where chronological ordering is important, popular approaches include expanding window (each new time slice is added to the panel dataset) and rolling window (the oldest time slice is removed as a new time slice is added, Rossi and Inoue, 2012). Instead of randomly splitting training and validation sets, the *out-of-sample* procedure[6] can be used where the end of the series is withheld for evaluation. This is unsatisfactory in the context of stock return prediction for two reasons. First, each period is drawn from a different data distribution $\mathcal{D}$ (hereon denoted by $\mathcal{D}_t$ for dataset drawn at time t, or $\mathcal{D}_{oos}$ for all periods in the out-of-sample dataset). A regression that is fitted on a sliding window of size w effectively assumes that data at $t+1$ is drawn from the average DGP of the past $t-w, \ldots, t$ cross-sections. Secondly, if data is scarce in terms of time periods, estimates for optimal optimisation steps $\hat{\tau}_t$ can have large stochastic error. For instance, monthly data with a window size of 12 months and 3:1 training-validation split. $\hat{\tau}$ is estimated using only 3 months of data. To the best of our knowledge, there is no procedure for adapting early stopping in an online context with time-varying dynamics.

[6] As described in Bergmeir et al. (2018).

Note that even though the problem studied in this chapter contains a time-dimension, the problem itself concerns cross-sectional predictions. Thus, conventional sequential neural networks such as recurrent neural networks (Rumelhart et al., 1986b) and long-short term memory networks (Hochreiter and Schmidhuber, 1997) are not well suited to this problem. However, there have been some recent advances in dealing with concept drifts in time-series problems, such as Liu et al. (2019) who proposed to explicitly model concept drift in a discriminative manner inside an extreme learning machine, and Samanta et al. (2020) who proposed to model time-varying temporality of time-series using a Bayesian approach.

3.2.3 Online optimisation

Optimising network weights to track a function evolving under unknown dynamics is an online optimisation problem. A discussion on relevant concepts in online optimisation is provided in this section. Interested readers are encouraged to read Shalev-Shwartz (2012) for a comprehensive review. In online optimisation literature, iterate is often denoted by x_t and loss function by f_t. We have used $\boldsymbol{\theta}_t$ as iterate to be consistent with our parameter of interest and J_t as loss function to avoid conflict with our use of f as activation function.

Optimality of online optimisation and its variants for this class of problems under various assumptions have been well documented in literature (e.g., Shalev-Shwartz, 2012; Cesa-Bianchi et al., 2012; Dworkin et al., 2014). Thus, online optimisation is also well suited to our iterative portfolio selection problem due to their similarities. Applications of online optimisation in finance first came in the form of the *Universal Portfolios* by Cover (1991). However, most of the early works in online optimisation are focused on the convex case and assume each draw of J_t is from the same distribution (in other words, J_t is stationary). These assumptions are not consistent with our problem. Recently, Hazan et al. (2017) extended online convex optimisation to the non-convex and stationary case. This was further extended by Aydore et al. (2019) to the non-convex and non-stationary[7] case, with the proposed DTS-SGD algorithm. Non-convex optimisation is NP-Hard[8]. Therefore, existing non-convex

[7]Non-stationarity in online optimisation literature refers to time-variability of loss function J_t.

[8]In computer science, NP-Hard refers a class of problems where no known polynomial run-time algorithm exists.

optimisation algorithms focus on finding local minima (Hazan et al., 2017). For this reason, one difference between online *convex* optimisation and online *non-convex* optimisation is that the former focuses on minimising sum of losses relative to a *benchmark* (for instance, the minimiser over all time intervals $\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}\in\Theta} \sum_t J_t(\boldsymbol{\theta})$ is one of the most basic benchmarks), and the latter focuses on minimising sum of gradients (e.g., $\sum_t \nabla J_t(\boldsymbol{\theta}_t)$), without comparison to a benchmark. This sum is called *regret* and the optimisation objective is called *regret minimisation*. In online optimisation, it is desirable to design algorithms that minimises *average regret* over time (e.g., $\frac{\sum_t J_t(\boldsymbol{\theta})}{T}$), as this guarantees that as $T \to \infty$, average regret suffered by the algorithm converges to zero (Shalev-Shwartz, 2012). Readers familiar with time-series analysis might be taken aback by the lack of parameters in a typical online optimisation algorithm. This is due to the game theoretic approach of online optimisation and the focus on worst case performance guarantees, as opposed to the average case performance in statistical learning. Regret bounds are typically functions of properties of the loss function (e.g., convexity and smoothness) and are dependent on environmental assumptions.

At each interval t, DTS-SGD updates network weights using a time-weighted sum of past observed gradients. Time weighting is controlled by a forget factor κ. In analysing DTS-SGD, we note two potential weaknesses. Firstly, neural networks are notoriously difficult to train. Geometry of the loss function is plagued by an abundance of local minima and saddle points (see Chapter 8.2 of Goodfellow et al., 2016). Momentum and learning rate decay strategies (for instance, Sutskever et al., 2013; Kingma and Ba, 2015) have been introduced which require multiple passes over training data, adjusting learning rate each time to better traverse the loss surface. DTS-SGD performs a single weight update at each period which may have difficulties in traversing highly non-convex loss surfaces. Secondly, during our simulation tests, we observed that loss can increase after a weight update. One possibility is that a past gradient is taking the weights further away from the current local minima. This is particularly problematic for our problem as stock returns are very noisy.

3.3 The proposed Online Early Stopping algorithm

3.3.1 Tracking a restricted optimum

We start by providing an informal discussion of the algorithm. Neural networks are universal approximators (Cybenko, 1989; Goodfellow et al., 2016). That is, it can approximate any function up to an arbitrary accuracy. Thus, given a network structure and a time-varying function, network weights trained with data from a single time interval (i.e., a cross-sectional slice of time) neatly summarise the function at that interval. The Euclidean distance between consecutive sets of weights can be interpreted as the amount of variations in the underlying function expressed in weight space. Simply using $\boldsymbol{\theta}_{t-1}$ to predict on t will lead to an overfitted result. To illustrate, suppose $\boldsymbol{\theta}_t \in \mathbb{R}$, $\boldsymbol{\theta}_0 = 0$ and $\boldsymbol{\theta}_{\{t>0\}}$ alternates in a sequence of $\{1, -1, 1, -1, ...\}$[9]. Then, it is clear that using $\boldsymbol{\theta}_1$ to predict on $t = 2$ will lead to a worse outcome than using $\boldsymbol{\theta}_0$. In this scenario, the optimal strategy is to never update weights (or scale updates by zero). Generally, the optimal policy is to regularise updates such that the network is not overfitted to any single period.

In the rest of this section, we present our main theoretical results. Formally, our goal is to track the unobserved minimiser of J_t, a proxy for the true asset pricing model, as closely as possible. In regret analysis, it is desirable to have regret that scales sub-linearly to number of periods T, which leads to asymptotic convergence to the optimal solution[10]. Hazan et al. (2017) demonstrated that in the non-convex case, a sequence of adversarially chosen loss functions can force any algorithm to suffer regret that scales with T as $\Omega\left(\frac{T}{w^2}\right)$[11]. Locally smoothed gradients (over a rolling window of w loss functions) were used to improve *smoothed regret*, with a larger w advocated by Hazan et al. (2017). Aydore et al. (2019) extended this to use rolling weighted average of past gradients which give recent gradients a higher weight to track a dynamic function. Inevitably, smoothing will track a time-varying minimiser with a tracking error that is proportionate to w and the forget factor κ.

[9] This is the worst case scenario for OES which we will revisit at the end of this section.
[10] The sum of regret increases sub-linearly to T. Thus, as $T \to \infty$, average regret $\to 0$.
[11] In computer science, Ω notation refers to the lower bound complexity.

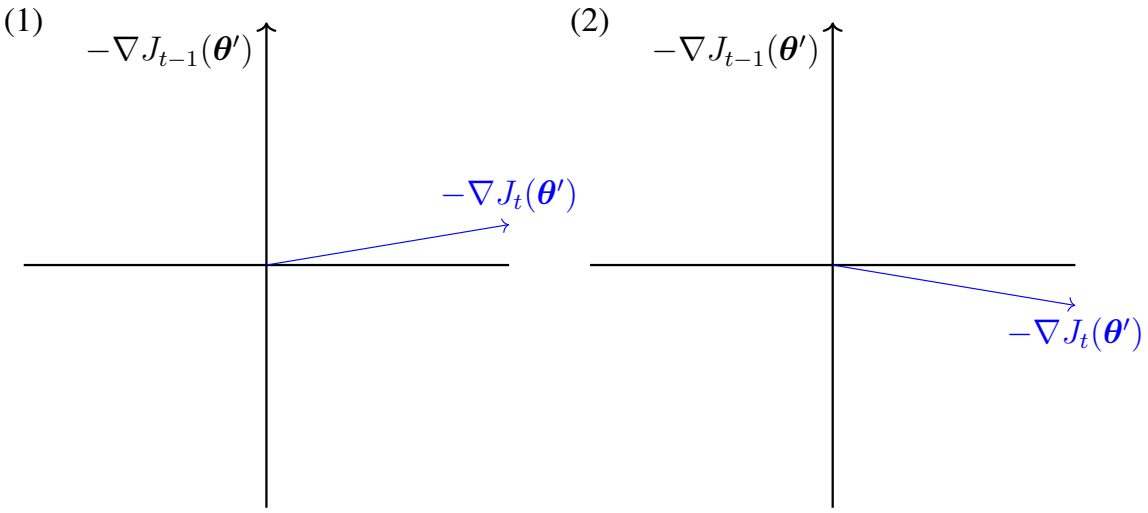


FIGURE 3.1: At each optimisation iteration, weights can be visualised as moving along the direction of $-\nabla J_{t-1}(\boldsymbol{\theta}')$. On the left, optimisation should continue until $-\nabla J_t(\boldsymbol{\theta}')$ is perpendicular to $-\nabla J_{t-1}(\boldsymbol{\theta}')$. On the right, optimisation should terminate.

To address this, we propose a *restricted optimum* (denoted by $\boldsymbol{\theta}_t^*$ at time t) as the tracking target of our algorithm. At time t, the online player selects $\boldsymbol{\theta}_t$ based on observed $\{\nabla_1, \ldots, \nabla_{t-1}\}$. As the network is trained using gradient descent, we propose to restrict the admissible weight set to the path formed from $\boldsymbol{\theta}_{t-1}^*$ and extending along the gradient vector $-\nabla_{t-1}$ (in other words, the path traversed by gradient descent). The point $\boldsymbol{\theta}'$ along this path with the minimum $\|\nabla J_t(\boldsymbol{\theta}')\|$ is the restricted optimum. We argue that the trade-off between restricting the admissible weight space and solving the simplified problem is justified as other points in the weight space are not attainable via gradient descent and is thus unnecessary to consider all possible weight sets in Θ. Without assuming any time-varying dynamics, updating weights using an average of past gradients (similar to Hazan et al., 2017) will induce a tracking error to the time-varying function. To illustrate the restricted optimum concept, let $\boldsymbol{\theta}' = \boldsymbol{\theta}_{t-1}^*$ be our starting point of optimisation, $\boldsymbol{g} = -\nabla J_{t-1}(\boldsymbol{\theta}')$ and $\boldsymbol{g}' = -\nabla J_t(\boldsymbol{\theta}')$. The possible scenarios during training are (also illustrated in Figure 3.1):

(1) If $\left|\cos^{-1} \frac{[\langle \boldsymbol{g}, \boldsymbol{g}' \rangle]}{\|\boldsymbol{g}\| \|\boldsymbol{g}'\|}\right| < \pi/2$, then moving along $\boldsymbol{g}$ will also improve $J_t(\boldsymbol{\theta}')$ until $\boldsymbol{g}$ is perpendicular to $\boldsymbol{g}'$ or $\boldsymbol{\theta}'$ has reached a local minima of J_{t-1}.

(2) If $\left|\cos^{-1}\frac{[\langle \boldsymbol{g},\boldsymbol{g}'\rangle]}{\|\boldsymbol{g}\|\|\boldsymbol{g}'\|}\right| \geq \pi/2$, then following $\boldsymbol{g}$ will not improve $J_t(\boldsymbol{\theta}')$ and training should terminate.

This observation motivates our OES algorithm. In this section, we will use $\boldsymbol{\theta}_t^*$ to denote restricted optimal weights at t and $\boldsymbol{\theta}_t$ to denote the online player's choice of weights. Suppose $\boldsymbol{\theta}_t^*$ evolves under the dynamics of,

$$\boldsymbol{\theta}_t^* = \boldsymbol{\theta}_{t-1}^* - v_{t-1}\nabla J_{t-1}(\boldsymbol{\theta}_{t-1}^*), \tag{3.5}$$

where v_{t-1} is sampled from an unknown distribution. v_{t-1} can be interpreted as a *regulariser* which provides the optimal prediction weights on J_t if we are restricted to travelling along the direction of $-\nabla J_{t-1}(\boldsymbol{\theta}_{t-1}^*)$. In this context, $\|\nabla J_t(\boldsymbol{\theta}_t^*)\|$ is the minimum gradient suffered by the player. Solution to the iterative portfolio selection problem described in Section 3.2.1 contains two loops (one nested within the other). The outer loop recursively updates $\boldsymbol{\theta}_t^*$ for each portfolio selection interval $t = 1, \ldots, T$ (each $\boldsymbol{\theta}_t^*$ in Equation (3.5)). The inner loop relates to the transition between each t, where SGD iteratively updates $\boldsymbol{\theta}_{t-1}^*$ to arrive at $\boldsymbol{\theta}_t^*$ by approximating v_{t-1} in (3.5). In here, let τ_t^* be the optimal number of optimisation steps at time t, τ_t be the estimated number of optimisation steps and k be the k-th SGD optimisation step. At iteration t, we solve optimal optimisation steps τ_{t-2}^*,

$$\tau_{t-2}^* = \underset{\tau' \geq 0}{\operatorname{argmin}}\, J_{t-1}\left[\boldsymbol{\theta}_{t-2}^* - \eta\sum_{k=1}^{\tau'}\nabla J_{t-2}(\boldsymbol{\theta}_{t-2,k}^*)\right]. \tag{3.6}$$

We start from $t-2$ as solving τ_{t-1}^* requires J_t which we are yet to observe. This leads to optimal weights (the restricted optimum) trained on J_{t-2} for prediction on J_{t-1},

$$\boldsymbol{\theta}_{t-1}^* = \boldsymbol{\theta}_{t-2}^* - \eta\sum_{k=1}^{\tau_{t-2}^*}\nabla J_{t-2}(\boldsymbol{\theta}_{t-2,k}^*), \tag{3.7}$$

and can be approximated by,

$$\boldsymbol{\theta}_{t-2}^* - \eta\sum_{k=1}^{\tau_{t-2}^*}\nabla J_{t-2}(\boldsymbol{\theta}_{t-2,k}^*) \approx \boldsymbol{\theta}_{t-2}^* - \eta\tau_{t-2}^*\nabla J_{t-2}(\boldsymbol{\theta}_{t-2}^*),$$

which implies $v_{t-2} \approx \eta\tau_{t-2}^*$. To predict $\hat{\boldsymbol{r}}_t$, we choose $\tau_{t-1} = \frac{1}{t-2}\sum_{q=2}^{t-1}\tau_{t-q}^*$ and train *prediction weights* on J_{t-1} by substituting in $\lfloor \tau_{t-1} + 0.5 \rfloor$ (the rounded up estimate of

optimisation steps),

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}^*_{t-1} - \eta \sum_{k=1}^{\lfloor \tau_{t-1}+0.5 \rfloor} \nabla J_{t-1}(\boldsymbol{\theta}^*_{t-1,k}) \approx \boldsymbol{\theta}^*_{t-1} - \eta \tau_{t-1} \nabla J_{t-1}(\boldsymbol{\theta}^*_{t-1}). \tag{3.8}$$

As η is a constant chosen by hyperparameter search, τ_{t-1} can be interpreted as a proxy to the regulariser v_{t-1}. Using our β-smooth assumption (in Section 3.2.1) and substituting in definitions of $\boldsymbol{\theta}_t$ and $\boldsymbol{\theta}^*_t$ (in Equation 3.8), we obtain total regret,

$$\begin{aligned} \|\nabla J_t(\boldsymbol{\theta}_t) - \nabla J_t(\boldsymbol{\theta}^*_t)\| &\leq \beta \|\boldsymbol{\theta}_t - \boldsymbol{\theta}^*_t\|, \\ \sum_{t=2}^{T} \|\nabla J_t(\boldsymbol{\theta}_t) - \nabla J_t(\boldsymbol{\theta}^*_t)\| &\leq \sum_{t=2}^{T} \beta \|\boldsymbol{\theta}_t - \boldsymbol{\theta}^*_t\|, \\ &\leq \sum_{t=2}^{T} \beta \left\|\eta \tau^*_{t-1} \nabla J_{t-1}(\boldsymbol{\theta}^*_{t-1}) - \eta \tau_{t-1} \nabla J_{t-1}(\boldsymbol{\theta}^*_{t-1})\right\|, \end{aligned} \tag{3.9}$$

where we start from $t = 2$ as our algorithm requires at least 2 cross-sectional observations. The elegance of Equation 3.9 is that it conforms with the conventional notion of regret, with cumulative gradient deficit against an optimal outcome in place of cumulative loss. As τ_{t-1} is the unbiased estimator of τ^*_{t-1}, Equation 3.9 indicates that the cumulative deficit is asymptotically bounded by the variance of τ^*_{t-1}. This concept is illustrated in Figure 3.2. If τ^*_{t-1} is constant, then τ_{t-1} will converge to τ^*_{t-1} and the optimal weights are achieved. Conversely, if τ^*_{t-1} has high variance, then the player will suffer a larger cumulative gradient deficit.

Finally, we discuss the best and worst case scenarios of OES. The best case scenario is if $\boldsymbol{\theta}_t$ is stationary, such that $\tau^*_t = 0$. In this case, from Equation (3.2), regret is 0. The worst case scenario is the example discussed at the beginning of this section. Suppose that $\boldsymbol{\theta}_t \in \mathbb{R}$, $\boldsymbol{\theta}_0 = 0$ and $\boldsymbol{\theta}_{\{t>0\}}$ alternates in a sequence of $\{1, -1, 1, -1, ...\}$. In this case, estimated steps $\tau_t = 0$ and $\boldsymbol{\theta}_t$ never updates. Thus, the upper bound on regret is (from Equation (3.2)),

$$\sum_{t=2}^{T} \|\nabla J_t(\boldsymbol{\theta}_t) - \nabla J_t(\boldsymbol{\theta}^*_t)\| \leq \sum_{t=2}^{T} \beta \left\|\eta \tau^*_{t-1} \nabla J_{t-1}(\boldsymbol{\theta}^*_{t-1}) - 0\right\|,$$

and total regret scales linearly with time, average regret (total regret divided by T) converges a constant and the network always underfit the data. However, as discussed in Section 3.2.3,

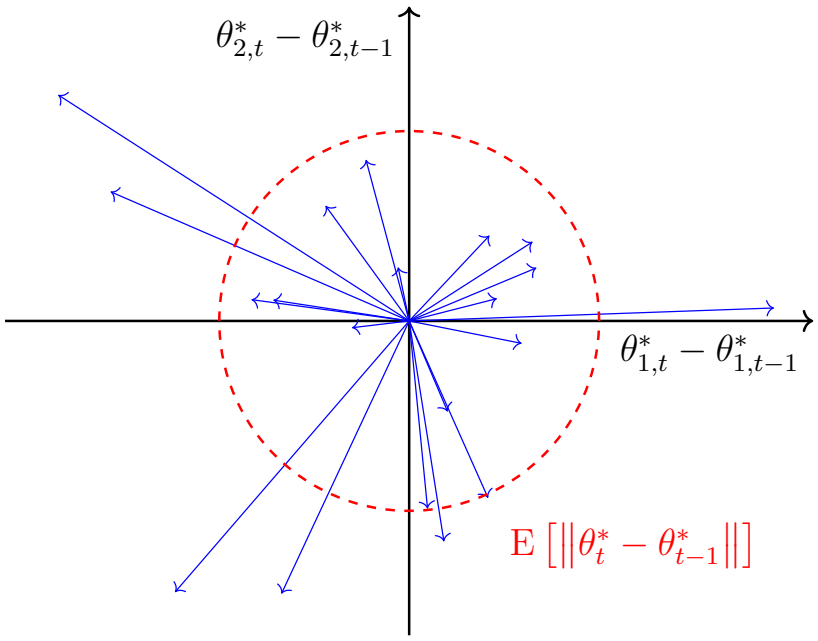


FIGURE 3.2: Illustration of estimating $\mathrm{E}\left[\left\|\boldsymbol{\theta}^*_t - \boldsymbol{\theta}^*_{t-1}\right\|\right]$. Suppose $\boldsymbol{\theta}^*_t = [\theta^*_{1,t} \quad \theta^*_{2,t}]$ is a row vector with two elements. Twenty one random $\boldsymbol{\theta}^*_t$ vectors were drawn with each $\boldsymbol{\theta}^*_t - \boldsymbol{\theta}^*_{t-1}$ pair represented as an arrow. The circle has radius $\frac{1}{20}\sum_{t=2}^{21}\left\|\boldsymbol{\theta}^*_t - \boldsymbol{\theta}^*_{t-1}\right\|$. $\boldsymbol{\theta}_t$ is regularised by limiting how far it can travel from $\boldsymbol{\theta}^*_{t-1}$ which is $\mathrm{E}\left[\left\|\boldsymbol{\theta}^*_t - \boldsymbol{\theta}^*_{t-1}\right\|\right]$.

regret in convex problems is typically compared to a benchmark (e.g., loss suffered by the best hindsight minimiser). In our worst case scenario, the best hindsight minimiser is also $\theta = 0$. Thus, regret suffered by OES converges to the best hindsight minimiser in our worst case. In other words, in the worst case, loss suffered by OES converges to a neural network that is trained on the entire pooled dataset. In Section 3.5, we demonstrate the real world performance of OES on U.S. equities dataset.

3.3.2 Proposed algorithm

Our strategy is to modify the early stopping algorithm to recursively estimate τ_t. An outline is provided below as an introduction to the pseudocode in Algorithm 2:

(1) At t, solve τ^*_{t-2} (Equation 3.6) and $\boldsymbol{\theta}^*_{t-1}$ (Equation 3.7) by training on J_{t-2} and validating against J_{t-1} (step 3 of Algorithm 2).

(2) Recursively estimate τ_{t-1} as the mean of observed $\{\tau^*_1, ..., \tau^*_{t-2}\}$ (line 4).

(3) Start from $\boldsymbol{\theta}^*_{t-1}$ and perform gradient descent for $\lfloor \tau_{t-1}+0.5 \rfloor$ iterations (Equation 3.8). The new weights are $\boldsymbol{\theta}_t$ (line 5–9).

(4) Predict using $\boldsymbol{\theta}_t$ (line 11).

$EarlyStopping$ on line 3 is the classical early stopping procedure as outlined in Algorithm 1 (in Section 2.2). In our implementation of the algorithm, we have used stochastic gradient $\hat{\nabla}_{t-1}$ instead of the full gradient ∇_{t-1}. Validation is performed before the first training step to allow for the case where $\tau_{best} = 0$ (i.e., we start from the optimal weights).

Algorithm 2 General framework for online early stopping. The outer loop recursively estimates τ_{t-1}. See Algorithm 1 for the EarlyStopping function.

Require: data $\boldsymbol{X}_t, \boldsymbol{y}_t \sim p_t$ at interval t; $\boldsymbol{\theta}^*_0$ initialized randomly

1: $\tau' \leftarrow 0$
2: **for** $t = 2, ..., T$ **do**
3: $\quad \tau', \boldsymbol{\theta}^*_{t-1} \leftarrow \text{EARLYSTOPPING}(\boldsymbol{\theta}^*_{t-2}, \boldsymbol{X}_{t-2}, \boldsymbol{y}_{t-2}, \boldsymbol{X}_{t-1}, \boldsymbol{y}_{t-1})$
4: $\quad \tau \leftarrow \frac{\tau(t-2)+\tau'}{t-1}$
5: $\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^*_{t-1}$
6: $\quad$ **for** $i = 1, ..., \lfloor \tau + 0.5 \rfloor$ **do**
7: $\quad\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \hat{\nabla}_{t-1}(\boldsymbol{\theta})$
8: $\quad$ **end for**
9: $\quad \boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}$
10: $\quad$ Receive input $\boldsymbol{X}_t$
11: $\quad$ Predict $\hat{\boldsymbol{r}}_t \leftarrow F(\boldsymbol{X}_t; \boldsymbol{\theta}_t)$
12: $\quad$ Receive output $\boldsymbol{y}_t$
13: **end for**

In the next two sections, we conduct two empirical studies. First is based on simulation data which highlights the use of OES, and the second on predicting U.S. stock returns based on the dataset in Gu et al. (2020) and is presented in Section 3.5.

3.4 Simulation study

3.4.1 Simulation data

For the simulation study, we create the following synthetic dataset:

- $T = 180$ months, each month consists of $N = 200$ stocks.

- Each stock has $M = 100$ features, forming input matrix of $\boldsymbol{X} \in \mathbb{R}^{180\times200\times100}$ and output vector $\boldsymbol{r} \in \mathbb{R}^{180\times200}$.
- Let $x_{t,i,j}$ be the value of feature j of stock i at time t. Each feature value is randomly set to $x_{t,i,j} \sim N(0,1)$.
- Each feature is associated with a latent factor $\psi_{t,j} = 0.95\psi_{t-1,j} + 0.05\delta_{t,j}$, where $\delta_{t,j} \sim \mathrm{N}(0,1)$ and $\psi_{0,j} \sim \mathrm{N}(0,1)$. $\psi_{t,j}$ follows a Wiener process and drifts over time.
- Each output value is $y_{t,i} = \sum_{j=1}^{M} \tanh(x_{t,i,j} \times \psi_{t,j}) + \epsilon_{t,i}$, where $\epsilon_{t,i} \sim \mathrm{N}(0,1)$. Thus, $\boldsymbol{y}_t$ is non-linear with respect to $\boldsymbol{X}_t$ and the relationship changes over time.

We have used the same network setup and hyperparameter ranges as the empirical study on U.S. equities (outlined in Table A.1) but with a batch size of 50. EWNN has the same setup but is re-fitted at every 10-th time intervals. The dataset is split into three 60 interval blocks. Hyperparameters for OES are chosen using a grid search, a procedure called *hyperparameter tuning*. For each hyperparameter combination, the network is trained on the first 60 intervals and validated on the next 60 intervals. Hyperparameters with the minimum MSE in the validation set is used in the remaining 60 intervals as out-of-sample data. Performance metrics are calculated using the out-of-sample set. DGP of the synthetic dataset is designed to be non-linear and time-varying. We expect a slower decay rate to benefit EWNN and a faster decay rate to benefit OES. Size of train, validate and test sets are chosen arbitrarily and is not expected to change the results. DTS-SGD follows the same training scheme as OES, with additional hyperparameters: window period $w \in \{5, 10, 20\}$ and forget factor $\kappa \in \{0.9, 0.8, 0.7\}$. These hyperparameters relate to speed of change of the DGP. A faster changing DGP will lead to smaller window period and forget factor.

3.4.2 Simulation results

Our synthetic data requires the network to adapt to time-varying dynamics. Table 3.1 records results of the simulation. EWNN struggles to learn the time-varying relationships, with mean R^2 of $-8.26\,\%$ and mean rank correlation of $-4.07\,\%$. This is expected as the expanding window approach used in EWNN assumes the relationships at t are best approximated by

TABLE 3.1: Simulation results and selected hyperparameters by hyperparameter search averaged over time and ensemble networks. Values are in percentages unless specified (w refers to number of periods).

%	EWNN	OES	DTS-SGD
Metrics			
Pooled R^2_{oos}	-7.12	50.22	0.13
Mean R^2	-7.77	49.64	-0.33
IC	-4.21	71.24	6.29
Hyperparameters			
Mean L_1 penalty	0.01	0.09	0.04
Mean η	0.55	1.00	0.10
Mean w (periods)			14
Mean κ			83.00

the average relationships in the observed past. OES significantly outperforms the other two methods in this simple simulation, achieving mean R^2 of $49.64\,\%$ and mean rank correlation of $69.63\,\%$. These results demonstrate OES's ability to track a non-linear, time-varying function reasonably closely. There is a preference for higher L_1 regularisation and learning rate. In Aydore et al. (2019), the authors reported issues of exploding gradient with the *static time-smoothed stochastic gradient descent* in Hazan et al. (2017) and that DTS-SGD provided greater stability. In our simulation test, we observe gradient instability with DTS-SGD as well. During training, loss can increase after a weight update. We hypothesise that a past gradient is taking network weights away from the direction of the current local minima and could be an issue with this general class of optimisers. Lastly, we find that mean R^2 tends to be slightly lower than R^2_{oos} (which is reasonable with a smaller denominator of a negative term, see Equation (3.1)).

3.5 Predicting U.S. stock returns

3.5.1 U.S. equities data and model

The U.S. equities dataset in Gu et al. (2020) consists of all stocks listed on NYSE, AMEX, and NASDAQ from March 1957 to December 2016. The average number of stocks exceeds

5,200. Excess returns over risk-free rate are calculated as forward one-month stock returns over Treasury-bill rates. As noted in Section 3.2.1, stock returns exhibit non-Gaussian characteristics. Table 3.2 presents descriptive statistics of excess returns. Monthly excess returns are positively skewed and contain possible outliers that may influence the regression. We follow Gu et al. (2020) in using MSE but note that MSE is not robust against outliers. As noted in Section 3.1, we also provide an alternative setup that excludes microcap stocks. The alternative setup and empirical results are presented in Section 3.5.4.

The feature set includes 94 firm level features, 74 industry dummy variables (based on the first two digits of Standard Industrial Classification code (SIC)), and interaction terms with 8 macroeconomic indicators. The firm features and macroeconomic indicators used in Gu et al. (2020) are based on Green et al. (2017) and Welch and Goyal (2008), respectively. Firm-level features include price-based measures, valuation metrics and accounting ratios. These features are also highlighted in Section 1.5. The purpose of interacting firm-level features with macroeconomic indicators is to capture any time-varying dynamics that are related to (common across all stocks) macroeconomic indicators. For instance, suppose valuation metrics have a stronger relationship with stock returns during periods of high inflation. Then,

TABLE 3.2: Descriptive statistics of monthly excess returns of U.S. equities from April 1957 to December 2016, grouped into 10-Year periods. The numbers in the left column indicate percentiles. Monthly excess returns appear to contain some extreme values, particularly on the positive end. Variance of monthly excess returns varied over time.

%	1957-1966	1967-1976	1977-1986	1987-1996	1997-2006	2007-2016
Mean	0.95	0.25	0.95	0.64	0.90	0.50
Std Dev	9.98	14.89	15.84	18.44	19.93	16.26
Skew	212.44	184.21	365.98	1059.88	502.41	783.70
Min	-76.38	-91.88	-90.14	-99.13	-98.30	-99.90
1	-20.27	-31.41	-33.82	-40.39	-44.61	-38.96
10	-9.26	-14.99	-14.38	-15.61	-17.08	-14.25
25	-4.42	-7.78	-6.54	-6.64	-6.91	-5.76
50	-0.10	-0.65	-0.52	-0.41	0.00	0.24
75	5.14	6.21	6.67	6.18	6.67	5.84
90	11.62	16.23	16.43	16.11	17.57	14.06
99	33.04	49.60	51.99	56.92	65.43	48.08
Max	255.29	432.89	1019.47	2399.66	1266.36	1598.45

this information will be encoded in the interaction term. The aggregated dataset therefore contains $94 \times (8 + 1) + 74 = 920$ features. Each feature has been appropriately lagged to avoid look-forward bias and is cross-sectionally ranked and scaled to $[-1, 1]$. Table A.6 in the Internet Appendix of Gu et al. (2020) contains the full list of firm features.

A subset of the data is available on Dacheng Xiu's website[12] which contains 94 firm-level characteristics and 74 industry classification. Our main result uses $94 + 74 = 168$ firm-level features but results with the full 920 features are also provided as a comparison. At this point, it is useful to remind readers that our goal is to track a time-varying function when the time-varying dynamics are unknown. In other words, we assume that time-varying dynamics between stock returns and features are not well understood or are unobservable. As such, the subset of data without interaction terms is sufficient for our problem. If macroeconomic indicators do encode time-varying dynamics, our network will track changing macroeconomic conditions automatically.

Data is divided into 18 years of training (from 1957 to 1974), 12 years of validation (1975–1986), and 30 years of out-of-sample tests (1987–2016). We use monthly total returns of individual stocks from CRSP. Where stock price is unavailable at the end of month, we use the last available price during the month. Table A.1 (Appendix A3) records test configurations as outlined in Gu et al. (2020) and in our replication. A total of six hyperparameter combinations (L_1 penalty and η in Table A.1) are tested. We use the same training scheme as Gu et al. (2020) to train EWNN. Once hyperparameters are tuned, the same network is used to make predictions in the out-of-sample set for 12 months. Training and validation sets are rolled forward by 12 months at the end of every December and the model is re-fitted. An ensemble of 10 networks is used, where each prediction $\hat{y}_{t,i}$ is the average prediction of 10 networks.

To train OES, we keep the first 18 years (to 1974) as training data, and next 12 years (to 1986) as validation data. For each permutation of hyperparameter set, we have trained an online learner up to 1986. Hyperparameter tuning is only performed once on this period, as opposed to every year in Gu et al. (2020). As the algorithm does not depend on a separate set of data for validation, we simply take the hyperparameter set with the lowest monthly average MSE

[12] Dacheng Xiu's website https://dachxiu.chicagobooth.edu/

over 1975–1986 as the best configuration to use for rest of the dataset. Batch size of 1,000 for OES was chosen arbitrarily.

3.5.2 Predicting U.S. stock returns

In this section, we present our U.S. stock return prediction results. DTS-SGD did not complete training with a reasonable range of hyperparameters due to exploding gradient and is omitted from this section. As an overarching comment, R^2 for both EWNN and OES on U.S. stock returns are very low and are consistent with the findings of Gu et al. (2020). First, results with and without interaction terms are presented in Table 3.3, keeping in mind that our method should be compared against EWNN without interaction terms. Without interaction terms, OES and EWNN achieve IC of $4.53\,\%$ and $3.82\,\%$, respectively. The relatively high correlation of OES (compared to EWNN) indicates that it is better at differentiating relative performance between stocks. This is particularly important in our use case as practitioners build portfolios based on expected relative performance of stocks. For instance, a long-short investor will buy top-ranked stocks and short sell bottom-ranked stocks and earn the difference in relative return between the two baskets of stocks. Mean R^2 are $-12.14\,\%$ and $-9.68\,\%$ for OES and EWNN, respectively. Note that the denominator of mean R^2 is adjusted by the cross-sectional mean of excess returns. Therefore, negative means R^2 of both OES and EWNN indicate that neither method can accurately predict the magnitude of cross-sectional returns. Finally, OES scores $-2.48\,\%$ on R^2_{oos} and EWNN scores $0.22\,\%$. The low values of both methods underscore the difficulty in return forecasting. EWNN achieves higher Sharpe ratio (Equation (3.3)) than OES, at 1.63 and 0.83, respectively. As we will point out in Section 3.5.4, the high Sharpe ratio of EWNN is driven by microcap stocks. Despite the very low R^2, both methods can generate economically meaningful returns. This underscores our argument that R^2 is not the best measure of performance and verifies practitioners' choice of correlation as the preferred measure. We observe similar performance with interaction terms, suggesting that the 8 macroeconomic time series have little interaction effect with the 94 features. In the subsequent results in this section, we only report statistics without interaction terms.

TABLE 3.3: Predictive performance on U.S. equities. Pooled R^2_{oos} is calculated across the entire out-of-sample period as a whole. Mean R^2 and IC are calculated cross-sectionally for each month then averaged across time. P10-1 is the average monthly spread between top and bottom deciles. Sharpe ratio is based on P10-1 return spread and annualised. Mean hyperparameters are calculated over the ensemble of 10 networks and across all periods. *As reported* are results in Gu et al. (2020).

	With Interactions			W/O Interactions	
%	As reported	EWNN	OES	EWNN	OES
Metrics					
Pooled R^2_{oos}	0.4	0.13	-1.93	0.22	-2.48
Mean R^2		-9.89	-11.93	-9.68	-12.17
IC		3.51	4.22	3.82	4.53
P10-1	3.27	1.83	2.10	2.39	2.41
Sharpe ratio	2.36	0.94	0.72	1.63	0.83
Hyperparameters					
Mean L_1 penalty		0.0012	0.0154	0.0024	0.0028
Mean η		0.77	0.10	0.67	0.10

So why do IC and R^2_{oos} diverge? The answer lies in Table 3.4 and Figure 3.3. Here, we form decile portfolios based on predicted returns over the next month and track their respective realised returns. OES predicted values span a wider range than EWNN. This has contributed to a lower R^2, even though OES can better differentiate relative performance between stocks. EWNN used a pooled dataset which will average out time-varying effects. As a result, the average gradient will likely be smaller in magnitude. This is evident from the lower mean L_1 penalty and higher learning rate η chosen by validation. By contrast, OES trains on each time period individually and the norm of the gradient presented to the network at each period is likely to be larger. This led to a lower learning rate chosen by validation. Hence, variance of OES predicted values is higher and potentially requires higher or different forms of regularisation.

In Table 3.4 and Figure 3.3, we observe that the prediction performance of EWNN is concentrated on the extremities, namely P1 and P10, with realised mean returns of $-0.47\,\%$ and $1.92\,\%$ respectively. Stocks between P3 and P7 are not well differentiated. By contrast, OES is better at ranking stocks across the entire spectrum. Realised mean returns of OES are more

evenly spread across the deciles, resulting in higher correlation than EWNN. P10-1 realised portfolio returns are similar across EWNN and OES at 2.39 % and 2.41 %, respectively. However, the difference in mean return spread increases when calculated on a quintile basis (mean return of top 20 % of stocks minus bottom 20 %), to 1.75 % and 1.90 % for EWNN and OES, respectively. This reflects better predictiveness in the middle of the spectrum of OES. An investor holding a well diversified portfolio is more likely to utilise predictions closer to the center of the distribution and experience relative returns that are reminiscent of the quintile spreads (and even tertile spreads) rather than decile spreads. Lastly, forecast dispersion of OES is relatively high compared to EWNN and realised decile returns. We hypothesise that this is due to the small training dataset used by OES on each iteration (consisting of only the cross-section) and suggests additional regularisation may be required.

3.5.3 Time-varying feature importance

So far, our forecasts are predicated on time-varying relationships between features and stock returns. How do features' importance change over time? To examine this, we train the OES

TABLE 3.4: Predicted and realised mean returns by decile where each row represents a decile. *P1* is the mean excess returns of the first decile (0-10% of bottom ranked stocks) and P10-1 is *P10* less *P1* showing the return spread between the best decile relative to the worst decile. *As reported* are original results from Table A.9 in Gu et al. (2020).

	As reported		EWNN		OES	
%	Predicted	realised	Predicted	realised	Predicted	realised
P1	-0.31	-0.92	-0.59	-0.47	-3.53	-0.50
P2	0.22	0.16	0.09	0.15	-1.96	0.03
P3	0.45	0.44	0.37	0.54	-1.07	0.27
P4	0.60	0.66	0.55	0.64	-0.34	0.48
P5	0.73	0.77	0.70	0.73	0.30	0.67
P6	0.85	0.81	0.84	0.78	0.88	0.85
P7	0.97	0.86	0.99	0.85	1.46	1.04
P8	1.12	0.93	1.17	0.96	2.10	1.18
P9	1.38	1.18	1.43	1.26	2.89	1.42
P10	2.28	2.35	2.33	1.92	4.25	1.91
P10-1	2.58	3.27	2.92	2.39	7.78	2.41

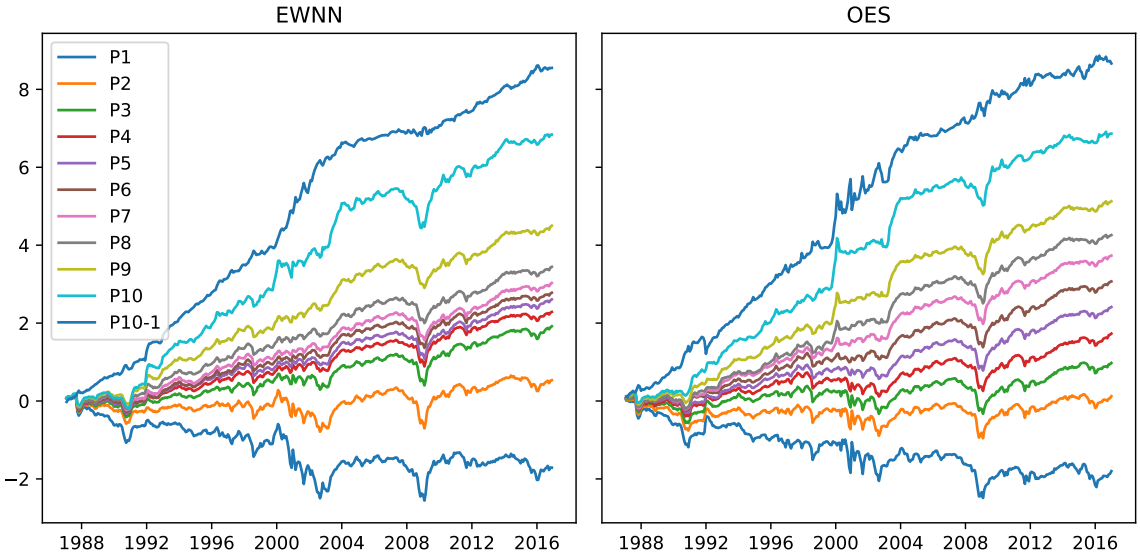


FIGURE 3.3: Cumulative mean excess returns by decile sorted based on predictions by EWNN and OES. Each portfolio follows the same construction as described in Table 3.4. However, cumulative mean excess returns of each portfolio is presented in the chart.

model at every period and make a baseline prediction. For each feature $j = 1, ..., M$, all values of j are set to zero and a new prediction is made. A new R^2 is calculated between the new prediction and the baseline prediction, denoted as $R^2_{t,j}$. The importance of feature j at time t is calculated as $\text{FI}_{t,j} = 1 - R^2_{t,j}$. Our measure tracks features that the network is using. This is different from the procedure in Gu et al. (2020) where R^2 is calculated against actual stock returns, rather than a baseline prediction.

To illustrate the inadequacy of a non-time-varying model, we first track feature importance over January 1987 to December 1991. The top 10 features with the highest feature importance are (in order of decreasing importance): *idiovol* (CAPM residual volatility), *mvel1* (log market capitalisation), *dolvol* (monthly traded value), *retvol* (return volatility), *beta* (CAPM beta), *mom12m* (12-month minus 1-month price momentum), *betasq* (CAPM beta squared), *mom6m* (6-month minus 1-month month price momentum), *ill* (illiquidity), and *maxret* (30-day max daily return). Rolling 12-month averages were calculated to provide a more discernible trend, with the top 5 shown in Figure 3.4. Feature importance exhibits strong time-variability.

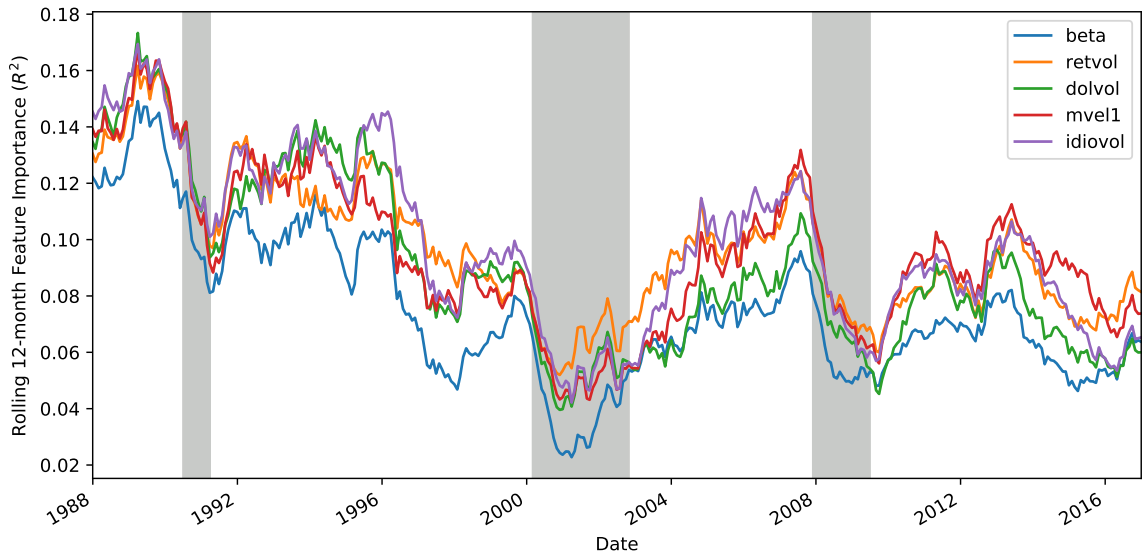


FIGURE 3.4: Top 5 features based on rolling 12-month average feature importance over 1987-1991. Three rapid falls can be seen which coincide with the 1990–91 U.S. recession, Dot-com bubble (2000–03) and the Global Financial Crisis (2007–09). These periods are shaded for reference.

Rolling 12-month average feature importance fell from $14\,\%$ to $16\,\%$ at the start of the out-of-sample period to a trough of $2\,\%$ to $6\,\%$ before rebounding. This indicates that the network would have changed considerably over time. Rapid falls in feature importance can be seen in Figure 3.4, over 1990–91, 2000–01 and 2008–09. These periods correspond to the U.S. recession in early 1990s, the Dot-com bubble and the Global Financial Crisis, respectively. Thus, market distress may explain rapid changes in feature importance.

Next, we examine changes in importance for all features on a yearly basis. Figure 3.5 displays considerable year-to-year variations in feature importance. As there are just a few clusters of features with relatively higher feature importance, the network's predictions can be attributed to a small set of features. This is likely due to the use of L_1 regularisation which encourages sparsity. There is an overall trend towards lower importance over time, consistent with the publication-informed trading hypothesis of McLean and Pontiff (2016). For instance, the importance of market capitalisation (*mvel1*) has decreased over time, as documented in Horowitz et al. (2000). There are periods of visibly lower importance for all features, over 2000–02 and 2008–09, and to a lesser extent 1990 and 1997 (Asian financial

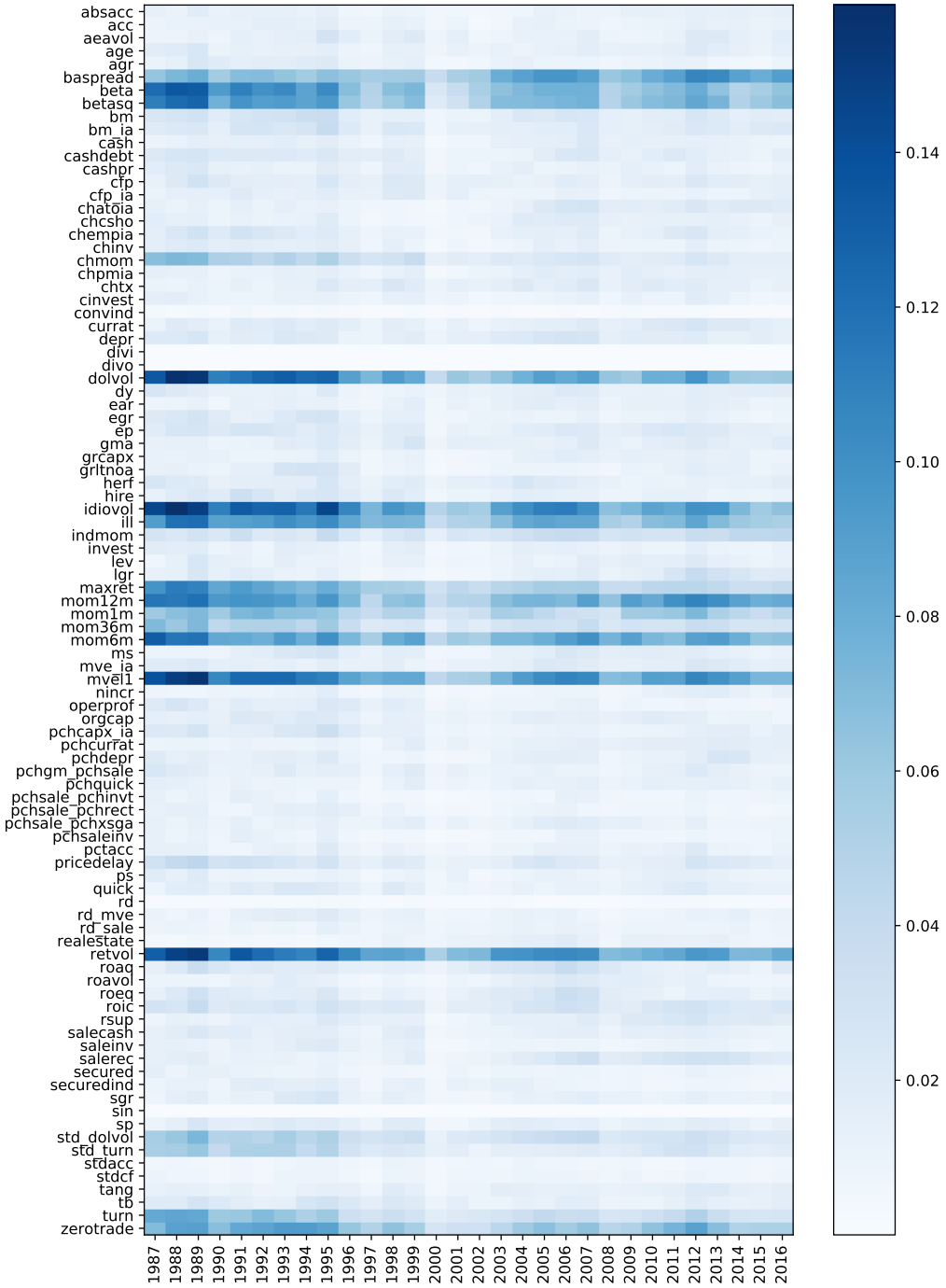


FIGURE 3.5: Yearly average R^2 to baseline predictions (in decimal). The OES network appeared to use only a handful of features. Shades of feature importance are distinctly lighter over 2000–02, 2008–09, and to a lesser extent in 1990 and 1997. Importance of some features have eroded over time (e.g., *dolvol*, *maxret* and *turn*).

crisis). If all features have lower importance during market distress, then what explains stock returns during these periods? To answer this question, we turn to importance of sectors, using SIC 13 (Oil and Gas), 60 (Depository Institutions) and 73 (Business Services) as proxies for oil companies, banks and technology companies, respectively. Figure 3.6 records the rolling 12-month average R^2 to baseline prediction of banks, oil and technology companies. The peak of importance of SIC 73 overlaps with the Dot-com bubble and peak of SIC 60 occurs just after the Global Financial Crisis (which started as a sub-prime mortgage crisis). Importance of SIC 13 peaked in 2016, coinciding with the 2014–16 oil glut which saw oil prices fell from over US$100 per barrel to below US$30 per barrel. This is an example of how an exogenous event that is confined to a specific industry impacts on predictability of stock returns. Thus, a plausible explanation for the observed results is that firm features explain less of cross-sectional returns during market shocks, which becomes increasingly explained by industry groups. This is particularly true if the market shock is industry related. For instance, technology companies during the Dot-com bubble, oil companies during an oil crisis and lodging companies during a pandemic. This underscores the importance to have a dynamic model that adapts to changes in the true model.

In this chapter, we argue that τ_t^* can be interpreted as a measure of variations between consecutive months. Recall that to solve τ_t^*, we train on $\mathcal{D}_t$ and validate on $\mathcal{D}_{t+1}$. Thus, τ_t^* is low if training on $\mathcal{D}_t$ is not beneficial for prediction on $\mathcal{D}_{t+1}$ and τ_t^* is high if $\mathcal{D}_t$ and $\mathcal{D}_{t+1}$ are relatively similar *and* there is a lot of room to update the network before early stopping terminates training. An example of such scenario is when the market is at a turning point, transiting from a risk-averse (risk-seeking) environment to a risk-seeking (risk-averse) environment. Optimal number of iterations τ_t^* is specific to month t and using it to train a network to predict for the next month will lead to overfitting. However, it is still useful to analyse τ_t^* as it provides information on the time-variability of the market as a whole. Figure 3.7 records both τ_t^* and τ_t of OES, including the hyperparameter tuning period (from 1957 to 1986). τ_t converges quickly to approximately 4 iterations and stays relatively stable throughout the approximately 60-year history. Most of the time, τ_t^* fluctuates between 1 and 7, and occasionally jumps to over 10. Periods of U.S. recession and the 2014–16 oil glut have been shaded in grey. Additionally, we have also shaded two market events in green, the

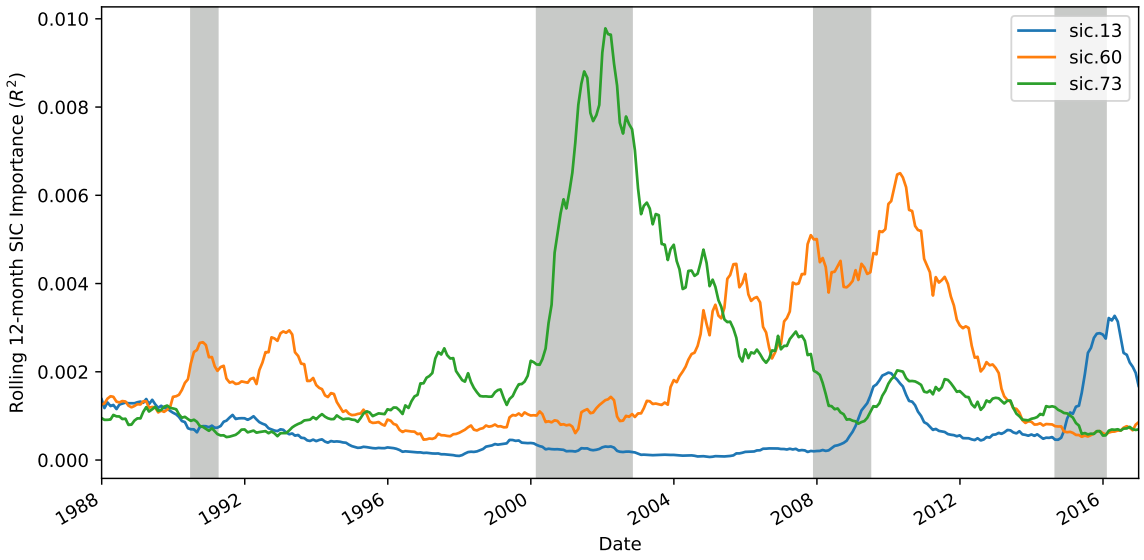


FIGURE 3.6: Rolling 12-month average R^2 to baseline prediction of SIC code 13, 60 and 73, as proxies for oil & gas companies, banks and technology companies, respectively. R^2 of technology companies peaks over 2001–02, banks over 2008–10, and oil companies over 2015–16. Duration of 1990–91 U.S. recession, Dot-com bubble, Global Financial Crisis and the 2014–16 oil glut have been shaded in grey.

Black Monday stock market crash in October 1987, and the collapse of Long-Term Capital Management in August 1998. These events have caused τ_t^* to spike, indicating a sudden change in the underlying DGP. The month with the highest τ_t^* is March 2009, which coincides with the start of a broad market rebound during the depth of the Global Financial Crisis. During these periods ($\tau_t < \tau_t^*$), OES stops training early and prevents overfitting to the large change in DGP.

3.5.4 Investable simulation

As noted in Section 3.1, the dataset in Gu et al. (2020) contains many stocks that are small and illiquid. The U.S. Securities and Exchange Commission (2013) defines "microcap" stocks as companies with market capitalisation below US$250–300 million and "nanocap" stocks as companies with market capitalisation below US$50 million. At the end of 2016, there are over 1,300 stocks with market capitalisation below US$50 million and over 1,800 stocks with

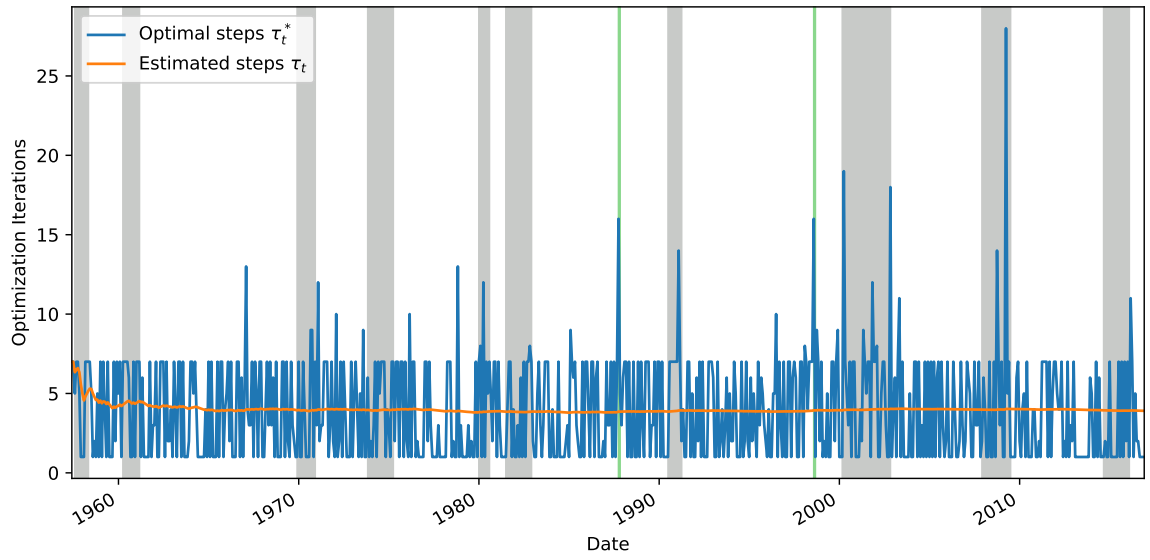


FIGURE 3.7: Optimal and estimated number of optimisation iterations. U.S. recessions and the oil glut (2014–16) have been shaded in grey. Two market shocks — the Black Monday stock market crash in October 1987 and the collapse of Long-Term Capital Management in August 1998, have been shaded in green.

market capitalisation between US$50 million and US$300 million. Together, microcap and nanocap stocks constitute close to half of the dataset as of 2016. Thus, we also provide results excluding these stocks. At the end of every June, we calculate breakpoint based on the 5-th percentile of NYSE listed stocks and exclude stocks with market capitalisation below this value. Once rebalanced, the same set of stocks are carried forward until the next rebalance (unless the stock ceases to exist). This cutoff is chosen to approximately include the larger half of U.S.-listed stocks, with the average number of stocks exceeding 2,600. We label this dataset as the *investable set*. To mitigate the impact of outliers, we also winsorise excess returns at $1\,\%$ and $99\,\%$ for each month (separately). Winsorised returns are then standardised by subtracting the cross-sectional mean and dividing by cross-sectional standard deviation. Standardisation is a common procedure in machine learning and can assist in network training (LeCun et al., 2012). Predicting a dependent variable with zero mean also removes the need to predict market returns which are embedded in stocks' excess returns (over risk-free rate).

This transformation allows the neural network to more easily learn the relationships between relative returns and firm characteristics.

Results based on this investable set are presented in Table 3.5. Both R^2_{oos} and IC improved once microcaps are excluded, with OES scoring $6.05\,\%$ on IC and EWNN on $5.74\,\%$. However, EWNN experienced a significant drop in mean decile spread (to $1.69\,\%$ per month) and Sharpe ratio (0.69), suggesting that microcaps are significant contributors to the results using the full dataset. By contrast, mean decile spread and Sharpe ratio remain stable for OES, at $2.41\,\%$ and 0.82, respectively. This indicates that the predictive performance of OES was not driven by microcap stocks. We believe this is a meaningful result for practitioners as this subset represents a relatively accessible segment of the market for institutional investors. An ensemble based on the average of cross-sectionally standardised predictions of the two models achieved the best IC, decile spread and Sharpe ratio relative to OES and EWNN. Mean monthly correlation between OES and EWNN is only $35.9\,\%$. Thus, an ensemble based on the two methods can effectively reduce variance of the predictions. Monthly correlations between the two models are presented in Figure 3.8. We observe that correlation tends to be

TABLE 3.5: Predictive performance on the investable set. Ensemble is the average of standardised predictions of the two methods. Pooled R^2_{oos} is calculated across the entire out-of-sample period as a whole. Mean R^2 and IC are calculated cross-sectionally for each month then averaged across time. P_t is the average monthly spread between top and bottom deciles. Sharpe ratio is based on P_t and annualised by multiplying $\sqrt{12}$. Mean hyperparameters are calculated over the ensemble of 10 networks and across all periods.

%	EWNN	OES	Ensemble
Metrics			
Pooled R^2_{oos}	0.35	-1.37	
Mean R^2	0.35	-1.37	
IC	5.74	6.05	6.29
P10-1	1.69	2.41	2.60
Sharpe ratio	0.69	0.82	0.96
Hyperparameters			
Mean L_1 penalty	0.0211	0.0046	
Mean η	0.87	0.10	

lowest immediately after a recession or crisis. We hypothesise that OES is quicker to react to economic recovery.

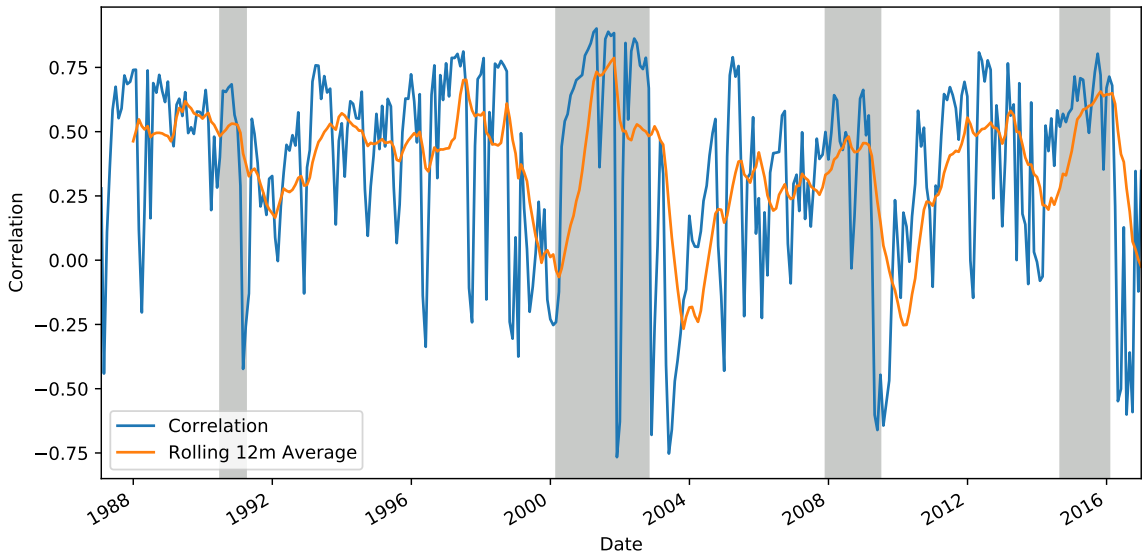


FIGURE 3.8: Monthly and rolling 12-month correlation between predictions of OES and EWNN. Duration of 1990–91 U.S. recession, Dot-com bubble, Global Financial Crisis and the 2014–16 oil glut have been shaded in grey.

Turning to cumulative decile returns presented in Figure 3.9, we observe significant drawdowns for EWNN during recovery phases of the Dot-com bubble and Global Financial Crisis. P1 of EWNN bounced back sharply during these episodes, causing sharp drops in decile spreads and are consistent with *momentum crashes* (Daniel and Moskowitz, 2016). By contrast, decile spreads of OES appear to react to the recovery more quickly. Consistent with prior findings, the spreads between decile 3 to 7 are also better under OES than EWNN in the investable set. Given these favourable characteristics, practitioners are likely to find OES a useful tool to add to the armoury of prediction models.

3.6 Conclusions

Stock return prediction is an arduous task. The true model is noisy, complex and time-varying. Mainstream deep learning research has focused on problems that do not vary over time and,

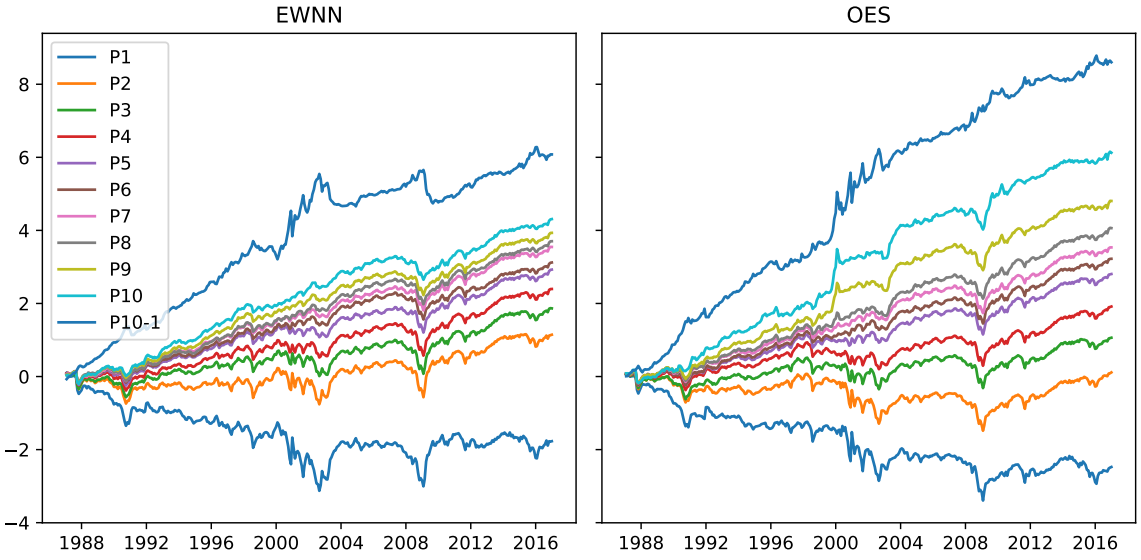


FIGURE 3.9: Cumulative mean excess returns by decile sorted based on predictions by EWNN and OES in the investable set.

arguably, time-varying applications have seen less advancements. In this chapter, we propose an Online Early Stopping algorithm that is easy to implement and can be applied to an existing network setup. We show that a network trained with OES can track a time-varying function and achieve superior performance to DTS-SGD, a recently proposed online non-convex optimisation technique. Our method is also significantly faster, as only two periods of training data are required at each iteration, compared to the pooled method used in Gu et al. (2020) which re-trains the network on the entire dataset annually. In our tests, the pooled method took $5.5\,\text{hours}$ to iterate through the entire dataset (an ensemble of ten networks therefore takes $55\,\text{hours}$)[13]. By contrast, our method took $44.25\,\text{mins}$ for a single pass over the entire dataset (an ensemble of ten networks took $7.4\,\text{hours}$).

Gu et al. (2020) suggested that a small dataset and low signal-to-noise ratio were reasons for the lack of improvement with a deeper network. To this end, we show that only a handful of features contribute to predictive performance. This may be due to correlation between features and the use of L_1 regularisation which encourages sparsity. We also find evidence of

[13]Tests performed on AMD Ryzen™ 7 3700X, Python 3.7.3, Tensorflow 1.12.0 and Keras 2.2.4. Hyperparameter grid search was performed concurrently.

time-varying feature importance. In particular, features such as log market capitalisation (the size effect) and 12-month minus 1-month momentum have seen a gradual decrease to their importance towards the end of our test period, consistent with the publication-informed trading hypothesis of McLean and Pontiff (2016). We find that sectors can also exhibit time-varying importance (for instance, technology stocks during the Dot-com bubble). These results have strong implications for practitioners forecasting stock returns using well known asset pricing anomalies. Excluding microcaps, we find that OES offers superior predictive performance in a subuniverse that is accessible to institutional investors. We find that correlation between OES and EWNN is at its lowest after a recession or crisis. We argue that this is driven by faster reactions of OES in tracking the recovery. An ensemble based on the average prediction of the two models achieves the best IC and Sharpe ratio, suggesting that the two methods may be complementary.

From an academic perspective, recent advances in deep learning such as dropout and *residual connections* (He et al., 2016) may allow deeper networks to be trained, enabling more expressive asset pricing models. Given the higher variance of predictions produced by OES, future work should explore alternative methods of regularisation including dropouts, L_2 penalty or a mixture of regularisation techniques.

In Section 3.3.1, we have discussed the worst case regret of OES which, under adversarial assumptions, can lead to regret that scales linearly with time. We note that our worst case regret converges to the best hindsight minimiser (i.e., the best choice of $\boldsymbol{\theta}$ if the investor is only allowed to pick one $\boldsymbol{\theta}$ for all time periods in hindsight). Thus, this provides users with a guarantee on worst case performance that is no worse than the best hindsight minimiser. The interpretation of this performance guarantee is as follows. In the worst case, our proposed OES algorithm will converge to the performance of a single neural network that is fitted on the entire dataset, as if the problem is stationary. However, this opens up an avenue for future research in advancing online non-convex optimisation algorithms that achieve regret that scales sublinearly with time, such that average regret converges to zero as $T \to \infty$.

In this chapter, we have applied neural networks in a cross-sectional prediction context — inputs into the network are point-in-time attributes of a stock and the problem is treated as a

conventional panel regression problem (with a neural network in place of a linear regression model). The network itself does not learn time-series features of the raw time-series. In Chapter 4, we explore neural networks that can learn from time-series directly.

CHAPTER 4

Supervised temporal autoencoder for stock return time-series forecasting

---

Financial markets are noisy learning environments. We propose an approach that regularises the TCN using a supervised autoencoder, which we term the STAE. We show that the addition of the auxiliary reconstruction task is beneficial to the primary supervised learning task in the context of stock return time-series forecasting. The supervised autoencoder denoises the input and encourages the main network to retain features that are beneficial to both prediction and reconstruction tasks. We show that the supervised temporal autoencoder is able to learn features directly from noisy stock price series, alleviating the need for handcrafted features. These contributions have resulted in the following publication:

Steven Y. K. Wong, Jennifer S. K. Chan, Lamiae Azizi, Richard Y. D. Xu, "Supervised Temporal Autoencoder for Stock Return Time-series Forecasting," *Proceedings of the IEEE 45th Annual Computer Software and Applications Conference*, Madrid, Spain, 2021.

4.1 Introduction

The motivating application of this chapter is in time-series stock return predictions — a problem that can be cast as a pattern recognition task. Using financial time-series forecasting as the motivating application, we focus solely on advancing the existing state-of-the-art deep learning techniques to deal with noise in a regression setting, as discussed in Section 1.6.2. In particular, the noisiness of financial markets exacerbates the difficulty in recognising patterns in stock prices/returns.

Deep learning has become the state-of-the-art in many time-series applications, such as machine translation (Sutskever et al., 2014) and audio generation (van den Oord et al., 2016). Whilst deep learning has achieved tremendous success in sequential applications such as speech and languages, advances in their applications in economics and finance have been relatively modest. Historically, sophisticated machine learning methods have not fared well in forecasting competitions such as the M-competitions[1] (Makridakis and Hibon, 2000; Makridakis et al., 2018; Hyndman, 2020). Empirical results suggest that simpler statistical methods are at least as accurate as sophisticated methods, and models that best fitted training data do not necessarily lead to higher forecasting performance out-of-sample (Makridakis et al., 2018). We argue that time-series in linguistics and speech are *information rich* — where preceding words or sound waves have high information content for the learning task and are naturally well-suited to neural networks with their rich parameterisation. In contrast, time-series in finance and economics are *information deprived* — where signal-to-noise tends to be low (Gu et al., 2020). Thus, good generalisation performance in high-noise environments requires regularisation and certain "scaffolds" to guide the model through the noisy data.

Classical statistical models such as autoregressive-moving-average (ARMA) (Box et al., 1994) models have strong scaffolds. The user picks the orders of autoregressive and moving average terms (through hyperparameter tuning). The two components additively contribute to the forecast in a linear manner. The scaffold "guides" the model to look for autoregressive relationships in the input sequence and residuals, and can be interpreted as a constraint on the functional form of the sequential DGP. However, statistical models such as ARMA are less able to capture more complex or non-linear patterns embedded in the time-series. Recently, Oreshkin et al. (2020) proposed Neural Basis Expansion Analysis for interpretable Time Series (N-BEATS), a hybrid model where basis functions are parameterised by fully-connected blocks. The authors reported state-of-the-art performance on the M4 competition dataset. Users are able to apply domain knowledge when choosing which basis function to use. For instance, if the user knows, *a priori*, that the data exhibits seasonality, then a cyclical basis function can be used. Basis functions provide a "strong form" of scaffold for

[1] M-competitions are forecasting competitions with datasets consisting of mainly business, economics, finance and demographics time-series.

the model. As the overall structure of the model is imposed by the user, the model only needs to fill in the intricacies during learning. This hybrid model approach aids interpretability and generalisation (by preventing an unconstrained neural network from overfitting on noise), but ultimately sacrifices the prized expressiveness of neural networks and requires assumptions on the data generation process. This restrictiveness may be undesirable to some users who want to take a more data-driven approach. At the other end of the spectrum, sequential neural network architectures have taken full advantage of the expressiveness on offer for information-rich applications. LSTM has been a popular choice of neural network architecture for sequential applications (as discussed in Section 2.5.1). More recently, *transformer* models (Vaswani et al., 2017) have achieved breakthrough advances in natural language processing (e.g., Devlin et al., 2019; Brown et al., 2020). The use of "direction-agnostic" attention mechanism allows transformers to perform paired associations in any part of the sequence, and thus solving a limitation of LSTM for language applications (Vaswani et al., 2017; Zeng et al., 2022). However, it remains to be seen if this flexibility afforded by transformers can benefit highly noisy financial time-series. In this chapter, we propose the Supervised Temporal Autoencoder (STAE) as an extension of the TCN (Bai et al., 2018). STAE augments TCN with an auxiliary learning task which acts as a regulariser. We show that STAE improved generalisation of TCN in time-series prediction of monthly returns of a broad set of U.S. stocks. In our proposed dual-objective network, interpretability is achieved through examining the replicated sequence produced by the autoencoder. We show in Section 4.3.2 that the autoencoder has a denoising effect on stock prices. We note that there have been previous attempts in combining an autoencoder with other network architectures (e.g., with LSTM in Heaton et al., 2016 and with CNN in Korczak and Hemes, 2017). However, in these works, the autoencoder is used for dimensionality reduction rather than as a regulariser as proposed in this chapter.

Methods of forecasting stock returns can be classified into two types, by cross-sectional prediction using firm-level features (as described in Section 1.6.1 and Chapter 3); and time-series forecasting[2] (as described in Section 1.6.2 and this chapter). Time-series forecasting implicitly assumes a stock's history contains information about its own future. This assumption stands

[2]Typically, features include a stock's own closing price history and related time-series data such as open, high, low and traded volume.

in contrast to the popular hypothesis within the finance discipline that stock prices evolve under a random walk (Fama, 1965). However, empirical evidence suggests at least a low level of predictability in stock returns. Finance literature has documented one aspect of stock return predictability, termed the MOM12 (Jegadeesh and Titman, 1993), defined as the change in price (adjusted for splits/dividends) from 12 months ago to 1 month ago. MOM12 was found to predict stock returns 1 month ahead. The momentum effect is pervasive and has been found to occur across many asset classes (Asness et al., 2013). Momentum has an intuitive interpretation — stocks that have increased (decreased) in value over (approx.) one year will continue to increase (decrease) in value over the next month. In other words, momentum describes the medium-term trending behaviour of stocks. In this chapter, we show that neural networks can learn this pattern directly from price data. As we will show in this chapter, this simple pattern is in fact very difficult to learn using conventional deep learning models. We argue that this is due to characteristics of stock returns that impedes learning, namely outliers, low signal-to-noise ratio, heavy tails and volatility clustering, as discussed in Section 1.5. Our contributions in this chapter are as follows:

- We propose STAE, a supervised autoencoder augmentation to TCN, a powerful convolutional network for sequential learning. The autoencoder regularises the network and provides a scaffold that can be interpreted as a nonparametric functional-form. This encourages the latent representation to retain information about the input sequence.
- We show that STAE materially improves forecasting performance of TCN and provides an economically meaningful improvement over MOM12, a well-known predictor of returns in financial literature.
- We show that the addition of autoencoder aids interpretability, by allowing the user to inspect the reconstructed input and visualise the features of the original sequence that are retained by the network.
- We establish a benchmark of neural network-based time-series forecasting performance in a large and investable set of U.S. stocks, comparing STAE to TCN, N-BEATS, LSTM and transformer. We show that STAE commands class-leading performance

in this challenging application, and that the reconstruction task is beneficial to forecasting performance even if added at a small weight.

- We provide a precedence on a set of transformations for augmenting raw price series into inputs for neural networks. We show that the network can learn useful features from this transformed series directly, eliminating the need for handcrafted features.

The rest of this chapter is organised as follows. In Section 4.2.1, we outline the problem setup. Section 4.2.2, 4.2.3 and 4.2.4 introduce existing literatures on convolutional neural networks, supervised autoencoders and financial time-series forecasting using deep learning. In Section 4.3, we describe our proposed STAE. Data and experimental setup of our empirical test is outlined in Section 4.3.1 and results in Section 4.3.2. Finally, we provide concluding remarks in Section 4.4

4.2 Preliminaries

4.2.1 Problem setup

We start with the familiar iterative asset return forecasting process of an investor. At every period $t \in \{1, \dots, T\}$, there are N stocks. We define total return index (TRI)[3] $u_{t,i} > 0$ of each stock i at t as the accumulation index, computed as the compounded change in price adjusted by dividends $u_{t,i} = u_{t-1,i}(p_{t,i} + d_{t,i})/p_{t-1,i}$, where $u_{0,i} = 1$, $p_{t,i}$ is price[4] at time t and $d_{t,i}$ is dividend at t if a dividend is paid, and zero otherwise. The input sequence is the log-transformed total return index $\boldsymbol{x}_{t,i} = \{\log u_{t-K+1,i}, \log u_{t-K+2,i}, \dots, \log u_{t,i}\}$, where $K = 250$ is chosen to be the approximate number of trading days per year and is motivated by the momentum effect (i.e., the one-year change in share price exhibiting predictive power on stock returns over the subsequent month). Further pre-processing of the sequence is outlined in Section 4.3.1.

[3] As noted in Section 1.2, total return includes both change in price and dividends. TRI is the compounded accumulation index of total returns. On the day a stock pays a dividend, its share price typically falls by roughly the dividend amount. TRI adds back dividends onto the price series such that stocks that pay dividends are not unfairly penalised.

[4] For simplicity, we assume that price is already adjusted for stock splits.

The dependent variable is forward 1-month return (proxied by 20 trading days), computed as the log-difference in TRI $y_{t,i} = \log u_{t+20,i} - \log u_{t,i}$. Note that this differs from the use of percentage returns in Chapter 3 (chosen to be comparable to Gu et al., 2020). Percentage returns are not normally distributed as the left tail is limited to $-100\,\%$, while the right tail is unlimited. Log-difference of the TRI (also known as *continuously compounded return* or *logarithmic return*) is also not normally distributed due to heavy tails (Peiró, 1994) but is often assumed to be Normal for modelling purposes (Isichenko, 2021). These choices allow our time-series model to be directly compared to the momentum and reversal effects, as documented in Jegadeesh and Titman (1993). Finally, the investor's objective is to find the model $F(\boldsymbol{x}; \boldsymbol{\theta})$ that best forecasts forward 1-month returns, by minimising the expected loss,

$$\min_{F, \boldsymbol{\theta} \in \Omega} \mathrm{E}_{\boldsymbol{x}, y \in \mathcal{D}} \left[\mathcal{L}(F(\boldsymbol{x}; \boldsymbol{\theta}), y) \right].$$

In Section 4.3, we further define the model F and parameterisation $\boldsymbol{\theta}$.

4.2.2 Neural networks for time-series applications

In this section, we provide a discussion on neural network architectures that can be applied to time-series applications.

There are three broad categories of sequential neural network architectures: RNN (and its variants, as discussed in Section 2.5.1), TCN (as discussed in Section 2.5.2) and, more recently, *transformer* models (Vaswani et al., 2017). Bai et al. (2018) argued that RNN suffers from several shortcomings, namely exploding and vanishing gradients, lack of parallelism and difficulty in retaining long term memory. TCN, utilising dilated convolutions, is able to model sequence of arbitrary length by increasing the kernel size and stacking multiple dilated convolution layers. Dilated convolutions provide the network with direct gradient flow to any part of the sequence while still preserving the temporal ordering of the sequence, thereby alleviating the problem of unstable gradients in recurrent networks. Using a benchmark dataset, Bai et al. (2018) demonstrated TCN's superior performance against other popular recurrent networks, including the LSTM. TCN is further validated in other sequential applications, such as speech synthesis (van den Oord et al., 2016), weather forecasting (Yan et al., 2020) and

traffic prediction (Dai et al., 2020). Moreoever, CNNs have achieved tremendous success in conventional image recognition tasks (Krizhevsky et al., 2012; Szegedy et al., 2015; Schroff et al., 2015). Thus, TCN makes for an ideal candidate for pattern recognition in time-series applications.

Sequential neural networks are often used for natural language processing applications. Sentence structure plays an important role in languages. For example, "the cat is brown" and "a brown cat" both have the same semantic meaning and differs in *ordering* of words. However, "the dog barked at the car because it was scared" and "the dog barked at the car because it was fast" contain a simple change of words and the subject association is completely different (dog with scared and car with fast). This context dependency proved challenging for conventional recurrent networks where temporal ordering is preserved and information flow is directional. More advanced LSTM-based language models, such as the model underpinning Google Translate[5] (Wu et al., 2016), employ *bidirectional recurrence* (Schuster and Paliwal, 1997) and *attention* (Luong et al., 2015). Bidrectional recurrence utilises separate LSTMs in both directions, while attention allows a word in the sentence to be associated with any other word in the sentence, regardless of adjacency. Both of these features increase flexibility and reduce scaffolding (i.e., information flow is no longer unidirectional). Transformers take this paradigm one step further by dropping recurrence and relying solely on *self-attention* (Vaswani et al., 2017). For each element of the sequence, self-attention computes association scores with every other element in the sequence. Thus, allowing gradients to flow between any pair of elements within the sequence and is *permutation-invariant* (Zeng et al., 2022). This flexibility proved vital in recent breakthroughs in machine translation applications (Vaswani et al., 2017; Devlin et al., 2019; Brown et al., 2020). However, Zeng et al. (2022) argued that time-series modelling involves extracting information from an ordered set of data points, which runs contrary to the permutation-invariant flexibility that is emblematic of transformers. Thus, transformers are unsuitable for long-term time-series forecasting. Whilst acknowledging that transformers are designed to solve different applications than time-series forecasting, driven by their compelling performance in NLP problems, we include transformers in our benchmark of neural network models for financial time-series forecasting.

[5] https://translate.google.com

Lastly, as discussed in Section 4.1, some recent advances focus on combining statistical constructs with neural networks, such as N-BEATS, which have also shown promising results in time-series applications. Rather than taking a data-driven approach (e.g., LSTM, transformers), N-BEATS allows the user to pre-specify basis functions which form the backbone of the model. The basis functions are parameterised by the outputs of fully connected layers. The choice of basis functions can be interpreted as placing a prior on the functional-form of the time-series. In this chapter, we compare time-series forecasting performance of our proposed STAE architecture, to TCN, LSTM, transformers, N-BEATS and MOM12. The selected models represent two distinct approaches to time-series forecasting — one which emphasises on flexibility and "letting the data speak", and one which imposes functional-form restrictions and thus regularises the model.

4.2.3 Supervised autoencoders

Owing to its vast learning capacity, neural networks can also easily overfit. This is particularly problematic for noisy environments such as financial markets. Advances in improving generalisation of neural networks include dropouts (Srivastava et al., 2014), early stopping (Morgan and Bourlard, 1990) and norm regularisation. Suddarth and Kergosien (1990) first proposed using an auxiliary learning task to assist with network training. More generally, MTL has been shown to improve generalisation performance across a range of tasks, such as facial landmark recognition (Zhang et al., 2014) and natural language processing (Collobert et al., 2011). Simultaneously learning multiple tasks can reduce overfitting through shared representations and by leveraging auxiliary information in secondary tasks. Supervised autoencoder (SAE), first proposed by Le et al. (2018), are a special case of MTL where the auxiliary task is to reconstruct the input used for the supervised learning task via an autoencoder (as discussed in Section 2.5.3; LeCun, 1987; Bourlard and Kamp, 1988; Hinton and Zemel, 1993). Le et al. (2018) and Epstein and Meir (2019) provided the theoretical generalisation bounds of autoencoders and showed that the addition of reconstruction error can improve generalisation of a classifier. The reconstruction task exhibit similar stability to l_2 regularisation but without the negative bias from shrinkage. The SAE learns two contradictory

tasks. The supervised learner only wants to retain features that are relevant for the supervised task, while autoencoder wants to retain all features that are relevant for reconstruction of the original input (Epstein and Meir, 2019). Thus, the autoencoder prevents the supervised learner from discarding too many features of the original sequence. We argue that this is beneficial to learning in a noisy environment (such as financial markets) as the supervised learner may be overfitting on spurious correlations. To date, SAEs have been applied to specific tasks such as classifying biological signals (Thiam et al., 2020; Barlaud and Guyard, 2020) and dialect detection (Parida et al., 2020). In this work, we show that SAE can improve generalisation in financial time-series prediction.

4.2.4 Deep learning in financial time-series prediction

Time-series forecasting using deep learning methods have been an active area of research and has been discussed in Section 1.6.2. Sezer et al. (2020) provided a recent survey of financial time-series forecasting using deep learning. In summary, majority of existing works focus on very short horizon forecasting, such as daily return or next day's closing price, using short sequences (e.g., Li et al., 2017 used previous day's close, high, low and open prices to predict next day's closing price). Very short term strategies are typically difficult to implement in practice due to high turnover, transaction costs (commissions, bid-ask spread and market impact) and overnight slippage[6]. Many existing works are also based on a small set of stocks (e.g., Hiransha et al., 2018 is based on 5 stocks) and/or over a short history (e.g., Chandra and Chand, 2016 used 3 stocks over 2006-10). Some previous methods use neural networks to tune parameters of handcrafted features (e.g., Lim et al., 2019). Our work differs from existing works in three ways. Firstly, we forecast forward 1-month return (proxied by 20 trading days, as opposed to daily returns in many existing works) and compare forecasting performance of our proposed network to a known predictor (the momentum effect) in finance literature. We only consider "pattern recognition on stock prices" a success if the neural network can learn additional patterns from stock prices that is above and beyond the momentum effect (which is

[6]Generally, predicted daily returns (based on the expected change in closing prices of today and tomorrow) are not achievable as the positions can only be initiated at market open the follow day at the earliest. If a stock's price is expected to increase tomorrow, the opening price is also likely to be higher than today's closing price. This *close-to-open* slippage is the overnight slippage.

based on just two data points — start and end stock prices). Secondly, in keeping with the spirit of deep learning, our approach performs feature selection automatically and without the need of any feature engineering (apart from log-transform and standardising daily prices). A common transformation of time-series is to take the first difference. If this is beneficial to the forecasting task, we expect the network to learn this directly from the data. Thirdly, we provide empirical results on the largest 3,000 stocks listed in the U.S. over 1984-2020, a dataset in which existing works have not being tested in.

4.3 Proposed STAE and application to stock return forecasting

We propose to augment TCN by adding a decoder which regularises the predictor subnetwork (convolutional encoder and fully connected predictor, as depicted in Figure 4.1). We term this network the Supervised Temporal Autoencoder (STAE). As per TCN, the convolutional

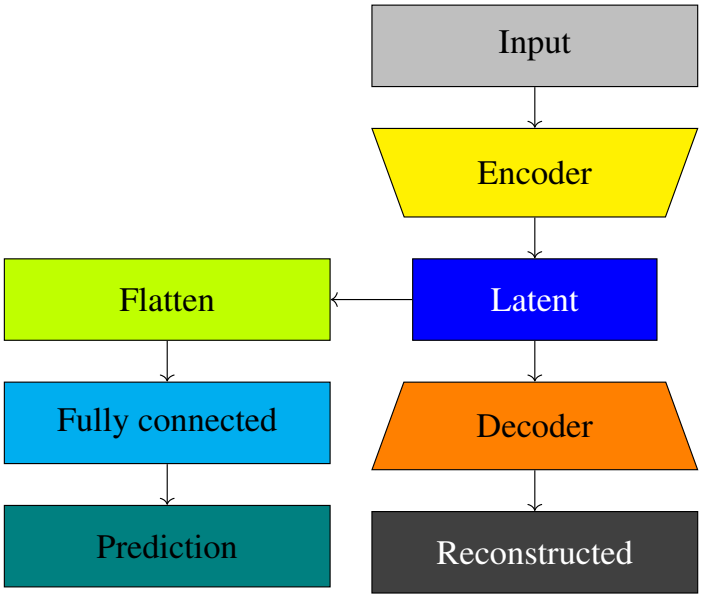
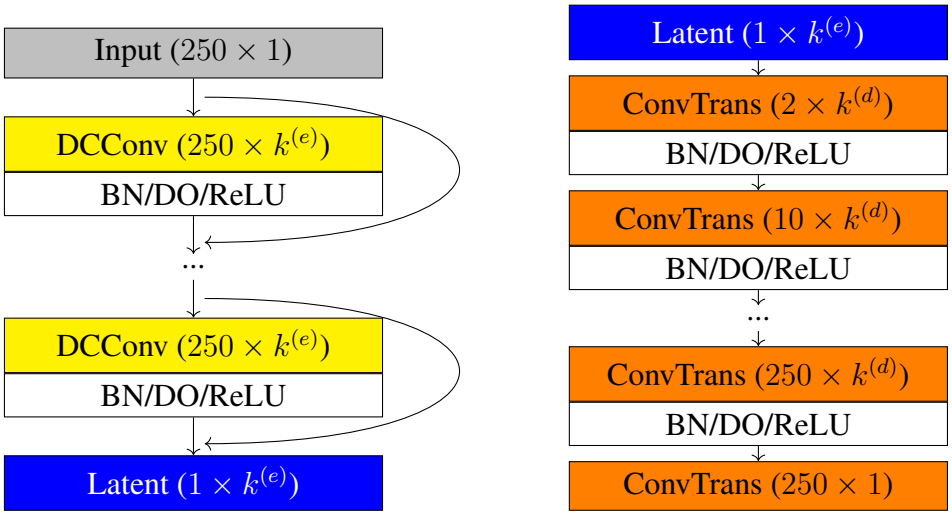


FIGURE 4.1: The Supervised Temporal Autoencoder architecture. A convolutional encoder converts input sequence into a latent representation which is used as input into one or more fully connected layers. *Prediction* produces output for the primary supervised learning task and *Reconstructed* is the reconstructed input sequence, as the auxiliary learning task.



(a) Encoder (b) Decoder

FIGURE 4.2: In 4.2(a), the encoder contains stacks of residual blocks. Each residual block consists of skip connection, dilated causal convolution (abbrev. *DCConv*), batch normalization (Ioffe and Szegedy, 2015), spatial dropout (Tompson et al., 2015) and rectified linear unit activation layers (abbrev. *BN/DO/ReLU*). In 4.2(b), the decoder uses *transposed convolution* layers (abbrev. *ConvTrans*) to reproduce the original sequence from latent representation. $k^{(e)}$ and $k^{(d)}$ are number of filters in convolutional layers of encoder and decoder, respectively. Each convolutional layer may have a different number of filters.

encoder (as depicted in Figure 4.2(a)) is organised into residual blocks (each containing dilated causal convolution, batch normalisation, dropout and ReLU layers) with skip connections between blocks. As the input are time-series, we use 1-D kernels in both the encoder and decoder. We use dilation rates of powers of 2 and allow hyperparameter search to choose between 8, 16 and 32 kernels, and kernel size of 2, 5 and 10, corresponding to daily[7], weekly and fortnightly features, respectively. For each output sequence of the last residual block, we take the last cell of the sequence as the latent representation of the entire sequence (as illustrated in Figure 2.9). The decoder uses *transposed convolutions* (also called *deconvolution*, Long et al., 2015) to recreate the original sequence from the latent representation, as illustrated in Figure 4.2(b). To reduce the hyperparameter search space, both encoder and decoder share the same number of kernels which is kept constant for all dilated convolution layers. In sum,

[7]For kernel size of 2, if the kernel learns values of $\{-1, 1\}$, then the sum product of this kernel with the input corresponds to the difference between the two data points.

STAE differs from conventional autoencoders in using temporal convolutions (as explained in Section 2.5.2 and Figure 2.9), as opposed to variants of recurrent networks such as Seq2Seq in (as discussed in Section 2.5.1; Sutskever et al., 2014) and letting reconstruction to serve only as a secondary task (rather than primary objective).

Next, we define the composite loss used to train STAE. The composite loss is the weighted sum of the prediction loss and reconstruction loss. As preliminaries, B is the size of each minibatch and $K = 250$ is the input sequence length (as defined in Section 4.2.1). Input $\boldsymbol{X} \in \mathbb{R}^{B \times K}$ is a matrix of B price sequences and $\boldsymbol{y} \in \mathbb{R}^{B \times 1}$ is vector of forward 1-month stock returns. $F(\boldsymbol{X}; \boldsymbol{\theta})$ is comprised of three sub-networks, encoder $F^{(e)}(\boldsymbol{X}; \boldsymbol{\theta}^{(e)})$, decoder $F^{(d)}(\boldsymbol{h}; \boldsymbol{\theta}^{(d)})$ and predictor $F^{(p)}(\boldsymbol{h}; \boldsymbol{\theta}^{(p)})$, where $\boldsymbol{h}$ is the latent representation produced by $F^{(e)}$, as depicted in Figure 4.1. For brevity, we use $\boldsymbol{\theta} = \{\boldsymbol{\theta}^{(e)}, \boldsymbol{\theta}^{(d)}, \boldsymbol{\theta}^{(p)}\}$ to denote weights and bias of all three sub-networks as a whole and $\boldsymbol{\theta}^{\{(e),(d),(p)\}}$ to denote weights and biases of encoder, decoder and predictor, respectively. We train network $F(\boldsymbol{X}; \boldsymbol{\theta}^{(e)}, \boldsymbol{\theta}^{(d)}, \boldsymbol{\theta}^{(p)})$ at each t using 10 years of daily prices preceding t. Further details on construction of the training set is provided in Section 4.3.1. The network is trained with *composite loss*:

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{y}; \boldsymbol{\theta}^{(e)}, \boldsymbol{\theta}^{(d)}, \boldsymbol{\theta}^{(p)}) = \ell^{(p)}(\boldsymbol{X}, \boldsymbol{y}; \boldsymbol{\theta}^{(e)}, \boldsymbol{\theta}^{(p)}) + \omega \ell^{(r)}(\boldsymbol{X}; \boldsymbol{\theta}^{(e)}, \boldsymbol{\theta}^{(d)}),$$

where $\ell^{(p)}$ is *prediction loss* (primary learning objective), $\ell^{(r)}$ is *reconstruction loss* (auxiliary learning objective), $\omega \in [0, 1]$ is the weight on $\ell^{(r)}$, and $\boldsymbol{\theta}^{(p)}, \boldsymbol{\theta}^{(e)}, \boldsymbol{\theta}^{(d)}$ are weights for predictor-, encoder- and decoder-part of the network, respectively. For brevity, we denote composite loss, prediction loss and reconstruction loss as simply $\mathcal{L}$, $\ell^{(p)}$ and $\ell^{(r)}$. We use *quadratic loss* for both prediction and reconstruction losses:

$$\ell^{(p)} = \frac{1}{B} \sum_{i=1}^{B} \left[y_i - F^{(p)}(F^{(e)}(\boldsymbol{x}_i; \boldsymbol{\theta}^{(e)}); \boldsymbol{\theta}^{(p)}) \right]^2$$

$$\ell^{(r)} = \frac{1}{B} \frac{1}{K} \sum_{i=1}^{B} \sum_{j=1}^{K} \left[\boldsymbol{x}_{i,j} - \hat{\boldsymbol{x}}_{i,j} \right]^2,$$

where $\boldsymbol{x}_i$ is the i-th row of matrix $\boldsymbol{X}$, $x_{i,j}$ is the j-th entry in the sequence $\boldsymbol{x}_i$, and,

$$\hat{\boldsymbol{x}}_{i,j} = F^{(d)}(F^{(e)}(\boldsymbol{x}_i; \boldsymbol{\theta}^{(e)}); \boldsymbol{\theta}^{(d)}).$$

Thus, $\boldsymbol{\theta}^{(e)}$ is influenced by both the prediction loss and reconstruction loss. With the loss function defined, the network is trained using SGD and early stopping (as discussed in Section 2.2). We use $\ell^{(p)}$ as the early stopping criterion rather than the composite loss as we are concerned with the best prediction performance. As the primary and auxiliary tasks have different convergence rates, using the composite loss as early stopping criterion may cause our predictor to under or overfit. We choose optimal ω as part of the hyperparameter search and expect ω to have similar behaviour to other regularisation techniques. A low ω will lead to under-regularisation and STAE will converge to TCN. A high ω will force the network to place too much focus on reproducing the input sequence and thus the prediction performance will deteriorate.

There are two distinct advantages of STAE in this context. Firstly, STAE can improve generalisation without resorting to function-form constraints in N-BEATS (as discussed in Section 4.1, function-form constraints are introduced into N-BEATS via user-defined basis functions). Secondly, by inspecting the reconstructed input from the decoder, the user can make sense of the features retained by the network and thus provide interpretability.

4.3.1 Data and experimental setup

Our empirical results are based on U.S. stock prices from CRSP. We construct TRI for each stock, adjusted by stock splits/consolidations and inclusive of dividends. We create a proxy of the Russell 3000 index by taking the 3,000 largest stocks in the U.S. at the end of every June. The same set of stocks are tracked for twelve months until the next rebalance (unless they are delisted). This broad universe ensures that there is sufficient breadth for the network to learn from but also excludes stocks with very low capitalisation that are unlikely to be investable by institutional investors. To the best of our knowledge, this universe is also broader than existing literature on time-series stock return prediction. Our dataset spans from 1984 to 2020. We use the initial 10 years, split into 7 years of training and 3 years of validation, for hyperparameter tuning. Then, for every January, we train a new network using 10 years of prices for stocks within the index in every month. The 10-year rolling window provides the network with sufficient data for training and ensures timeliness of the training set (in contrast

to an expanding window approach). The same network is used for prediction throughout the year. This provides us with 26 years of out-of-sample predictions for evaluation. Note that the results presented in this chapter differs to the published paper Wong et al. (2021) as the dataset is extended from 2018 to 2020, the models are re-trained with the addition of LSTM and transformers for comparison.

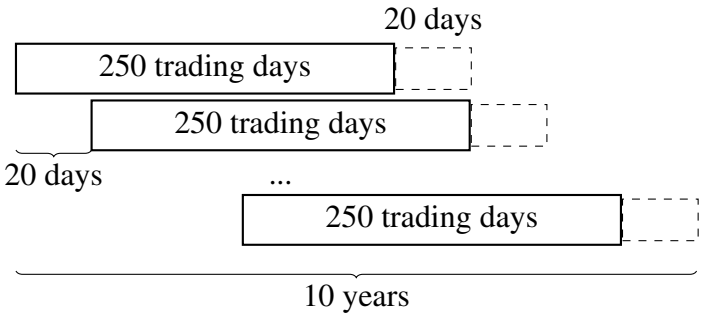


FIGURE 4.3: Training data is created by taking blocks of $250 + 20$ days of TRI.

Next, we will first describe pre-processing procedures for sequences $\boldsymbol{X}$, then expected return $\boldsymbol{y}$. Recall that in Section 4.3, we have defined sequence length $K = 250$ and y is forward 20-day return. In the 10-year rolling window, there are $\lfloor (252 \times 10 - 270)/20 \rfloor = 112$ cross-sections[8]. Let $\mathbf{T} = \{t - 20 \times (i-1) | i = 1, \ldots, 112\}, \mathsf{t} \in \mathbf{T}$ be *period counter* and,

$$\boldsymbol{x}^*_{\mathsf{t}-20,i} = \{\log u_{\mathsf{t}-K-19,i}, \log u_{\mathsf{t}-K-18,i}, \ldots, \log u_{\mathsf{t}-20,i}\}, \tag{4.1}$$

be a K-length price sequence for stock i. Log-transformation is performed in Equation (4.1) to stabilise variance (a common procedure for compounding economic time-series, Box and Jenkins, 1976; Lütkepohl and Xu, 2012). The median value of each sequence is then removed to centre the sequence[9],

$$\mu^{(\text{med})}_{\mathsf{t}-20,i} = \text{med}(\boldsymbol{x}^*_{\mathsf{t}-20,i})$$
$$\boldsymbol{x}'_{\mathsf{t}-20,i} = \boldsymbol{x}^*_{\mathsf{t}-20,i} - \mu^{(\text{med})}_{\mathsf{t}-20,i},$$

[8] Assuming an average of 252 trading days per year.

[9] Otherwise, stocks with high TRI values will denominate the training dataset once the training set is standardised.

where med is the median function. Values of all sequences $\boldsymbol{X}'_t = \{\boldsymbol{x}_{\mathsf{t}-20,i} | (i \in \mathrm{N} : i \leq N) \wedge (\mathsf{t} \in \mathbf{T})\}$ are standardised[10],

$$\boldsymbol{X}_t = \frac{\boldsymbol{X}'_t - \bar{\boldsymbol{X}}'_t}{\sigma(\boldsymbol{X}'_t)},$$

where $\bar{\boldsymbol{X}}'_t$ and $\sigma(\boldsymbol{X}'_t)$ are mean and standard deviation computed over all values of $\boldsymbol{X}'_t$, respectively. Standardised sequences $\boldsymbol{X}_t$ are then used as input into the network. Expected return is the forward 20-day return and cross-sectionally standardised,

$$\begin{aligned} y'_{\mathsf{t},i} &= \log u_{\mathsf{t}} - \log u_{\mathsf{t}-20} \\ \boldsymbol{y}'_{\mathsf{t}} &= \{y'_{\mathsf{t},i}\}_{i=1}^{N} \\ \boldsymbol{y}_{\mathsf{t}} &= \frac{\boldsymbol{y}'_{\mathsf{t}} - \bar{\boldsymbol{y}}'_{\mathsf{t}}}{\sigma(\boldsymbol{y}'_{\mathsf{t}})}. \end{aligned}$$

The training dataset is the pooled dataset, comprising of $\mathcal{D}_t = \{(\boldsymbol{x}_{\mathsf{t}-20,i}, y_{\mathsf{t},i}) | i = 1, \ldots, N \wedge \mathsf{t} \in \mathbf{T}\}$ standardised input-output pairs. This description is illustrated in Figure 4.3. $\boldsymbol{X}_t$ is standardised as a whole (i.e., elementwise) to preserve relative volatility of sequences[11]. For $\boldsymbol{y}_t$, the mean return of the cross-section represents the market return which may be difficult to forecast. $\boldsymbol{y}_{\mathsf{t}}$ is standardised cross-sectionally to remove the market return[12]. This treatment of $\boldsymbol{y}_{\mathsf{t}}$ is consistent with prior works (e.g., Fischer and Krauss, 2018). In effect, the neural network learns to predict a *score* drawn from $\mathrm{N}(0, 1)$. Predictions of the network $\hat{\boldsymbol{y}}_t$ are transformed

[10] Note that before standardisation, values are first winsorised at 1 % to remove outliers. This is applied to both $\boldsymbol{X}'_t$ and $\boldsymbol{y}'_t$.

[11] Consider two sequences, one where daily returns have standard deviation of 10 % and another has standard deviation of 1 %. Standardising $\boldsymbol{X}'_t$ as a whole preserves both the shape of the sequence (e.g., upward or downward trending) and relative volatility, while standardising each sequence individually does not preserve relative volatility and standardising by date does not preserve shape of the sequence.

[12] In our portfolio selection problem, we are only concerned with relative performance between stocks. Thus, it is safe to remove the market return (i.e., the cross-sectional mean). In Section 3.5.4, we have shown that this formulation improves the neural network's ability to predict stock returns.

back into the original distribution (in units of return) by,

$$\mu_t^{(\mathcal{D})} = \frac{1}{|\mathsf{T}|}\sum_{\mathsf{t}\in\mathsf{T}} \bar{\boldsymbol{y}}'_{\mathsf{t}}$$
$$\sigma_t^{(\mathcal{D})} = \frac{1}{|\mathsf{T}|}\sum_{\mathsf{t}\in\mathsf{T}} \sigma(\boldsymbol{y}'_{\mathsf{t}})$$
$$\hat{\boldsymbol{y}}'_t = \hat{\boldsymbol{y}}_t \sigma_t^{(\mathcal{D})} + \mu_t^{(\mathcal{D})}.$$

As noted in Section 4.2.2, we compare STAE to MOM12[13], TCN, LSTM, transformers and N-BEATS. TCN and STAE use mostly identical hyperparameter ranges. For N-BEATS, we use trend and generic models, and search over a range of polynomial dimensions. For LSTM, we search over the number of LSTM layers, followed by a fully-connected layer. For transformers, we search over the complexity of the multi-head attention block, followed by a fully-connected layer. Appendix A4 outlines the full sets of hyperparameters of each model. We train 10 networks for each model and compare the average performance. To gauge performance, we compare average cross-sectional MSE (Equation (2.4)), IC (Equation (3.2)), mean decile return (Equation (3.4)) and Sharpe ratio (Equation (3.3)).

4.3.2 Main empirical results

We start by discussing what the STAE "sees". Figure 4.4 records the input to the models (standardised log TRI of Facebook Inc.) and the reconstructed sequences at various reconstruction loss weights. Note that auxiliary loss weight $\omega = 1$ means both prediction and reconstruction have equal weight and will thus be influenced by what the predictor sees as important. There is a bias towards zero at the beginning of the sequence. This is due to causal padding which appends zeros to the start of the sequence. Thus, with a kernel size of 5, the first convolution involves 4 zeros and the first value of the sequence. Across the various ω, the reconstructed sequence tracks the overall shape of the true sequence reasonably well, with the exception of the local minima during 2018. Based on Figure 4.4, we interpret that STAE sees a general upward sloping trend.

[13] MOM12 is return over 11 months. We convert it into a monthly forecast by dividing by 11.

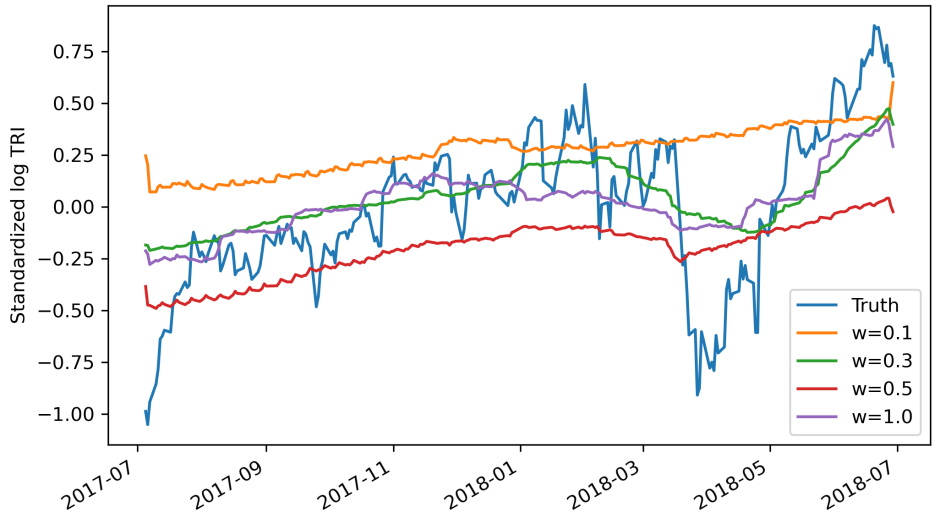


FIGURE 4.4: Standardised log TRI of Facebook Inc. and reconstructed time-series at various ω.

TABLE 4.1: **Main results**: Mean forecasting performance (of 10 networks scored individually) and performance of the ensemble (abbrev. *Ens.*) over the out-of-sample period (1994–2020). Decile return is mean difference in monthly returns of top and bottom deciles based on ensemble forecasts. Sharpe ratio is calculated as annualised decile returns divided by annualised standard deviation of decile returns. Best values in **bold**.

Metric	MOM12	N-BEATS	LSTM	Transformer	TCN	STAE
Mean IC (%)	1.77	1.76	2.11	2.25	1.74	**2.76**
Std Dev of IC (%)		0.32	0.41	0.28	0.91	**0.25**
Mean MSE	0.0205	0.0176	0.0176	0.0176	**0.0175**	0.0176
Ens. IC (%)	1.77	2.20	3.20	2.61	2.92	**3.36**
Ens. MSE	0.0205	0.0176	0.0175	0.0176	**0.0175**	0.0176
Decile Return (%)	0.67	0.57	**1.11**	0.57	0.92	1.05
Sharpe Ratio	0.25	0.25	**0.58**	0.22	0.40	0.45

Next, we turn to forecast accuracy. IC (introduced in Section 3.2.1) is our primary performance measure and is a widely used performance metric in investment management (Grinold and Kahn, 1999; Fabozzi et al., 2011b). We present two types of IC to illustrate the effects of ensembling. First, *mean IC*, given as the average IC of the 10 networks in the ensemble (computed for each network individually, then average is taken). Second, *ensemble IC* (denoted Ens. IC), given as the IC of the ensemble forecasts of the 10 networks (forecasts of the 10 networks are first averaged to produce the ensemble forecasts, then IC is computed).

The IC measure in Chapter 3 corresponds to ensemble IC. Similarly, we also report mean and ensemble MSE for the average of the 10 networks and the ensemble, respective. Consistent with Chapter 3, we also report decile return spread (Equation (3.4)) of the ensemble forecasts and Sharpe ratio of the decile return spread (Equation (3.3)). Table 4.1 records out-of-sample performance over 1994 to 2020. Overall, STAE achieved the highest IC for both the ensemble forecast (3.36 %) and each individual network (on average, at 2.76 %). LSTM is ranked second on IC, at 3.20 % for the ensemble and 2.11 % for the average over 10 networks. Transformer scored second on mean IC at 2.25 % (over 10 networks) but IC of the ensemble is only ranked fourth, at 2.61 %. Mean monthly decile return for LSTM is 1.11 %, marginally higher than STAE at 1.05 %. Sharpe ratio is also marginally higher at 0.58, compared to STAE at 0.45. For practitioners, IC, decile returns and Sharpe ratio are important performance metrics (Sharpe ratio and decile returns are related as Sharpe ratios are derived from decile returns, and are used by Gu et al., 2020). However, decile returns focus on top and bottom 10 % of forecasts without accounting for the middle 80 % of the distribution of forecasts. There are investment strategies that rely on less extreme return forecasts. In general, IC provides a more complete pictures of prediction performance by incorporating all forecasts. The higher IC of STAE reflects better ranking of stocks across the whole distribution, despite having similar decile returns. Both STAE and LSTM are economically meaningfully better than MOM12, with IC of 1.77 %, mean decile return of 0.67 % and Sharpe ratio 0.25. Comparing STAE to TCN, STAE is better on IC, mean decile return and Sharpe ratio. All 5 machine learning models achieve similar MSE of 0.0175–0.0176. Both TCN and LSTM appear to benefit more from ensembling, with IC of the ensemble forecast being 50 % to 60 % higher than the average IC of the individual models. This is compared to an increase of only 22 % for STAE and 16 % for transformer. In the case of STAE, we speculate that by regularising the network using an autoencoder, the networks are slightly more correlated to each other and thus reducing the benefits of ensembling. This is explored in more details in Section 4.3.4. Finally, N-BEATS has not performed well in our test. Due to the complexity of hyperparameter combinations, it is possible that the optimal hyperparameters lie outside of the search range.

Figure 4.5 records cumulative decile returns of MOM12, STAE, TCN, LSTM, transformer and N-BEATS. Cumulative returns of STAE, TCN and LSTM are significantly higher than

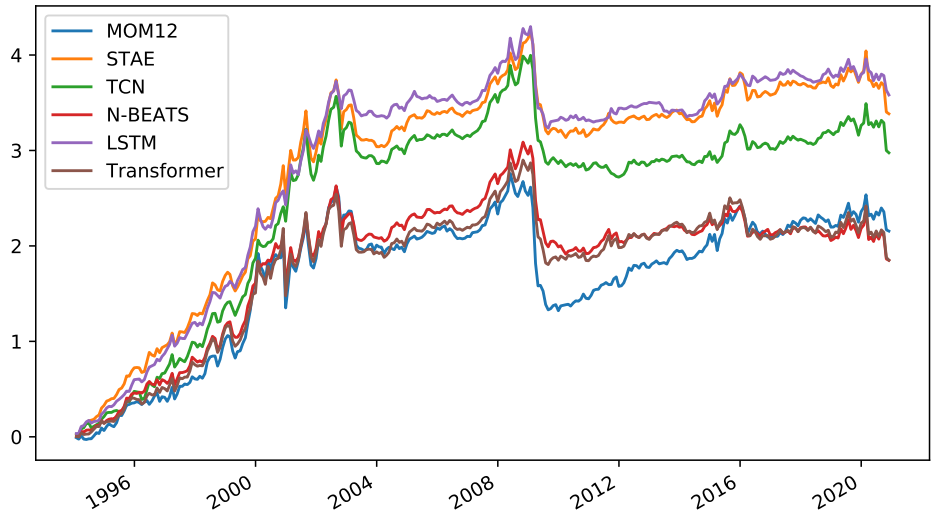


FIGURE 4.5: Cumulative decile returns based on ensemble forecasts of each model. Decile returns are calculated as mean top decile returns less mean bottom decile returns.

MOM12, transformer and N-BEATS. All 6 strategies experienced a "crash" in March 2009, as the U.S. market rebound from the depth of the global financial crisis. This is to be expected, as the input into the models are stocks' own price history and does not include information about the prevailing economic environment. A similar but smaller crash is noted at the end of the Dot-com bubble in 2003. One notable feature is the lower forecast efficacy of all models after the Dot-com bubble[14] We hypothesise that market efficacy has improved following the wide spread adoption of computers since the early 2000s. This constitutes *concept drift* and is discussed in Section 1.5.

TABLE 4.2: **Validation results**: Mean forecasting performance (of 10 networks) over the validation period (1991–1993). MSE is based for standardised returns and are not comparable to Table 4.1. Best values in **bold**.

Metric	**MOM12**	**N-BEATS**	**LSTM**	**Transformer**	**TCN**	**STAE**
Mean IC (%)	5.45	7.07	6.38	7.15	6.99	**8.50**
Mean MSE		0.9953	0.9963	0.9951	0.9955	**0.9929**

Next, we examine forecasting performance on the validation set, recorded in Table 4.2. STAE leads other models on mean IC (over 10 networks) by a large margin, scoring $8.50\,\%$. TCN,

[14]Cumulative returns in Figure 4.5 show relatively steeper inclines until 2003, then a more benign profile after 2003.

LSTM, transformer, N-BEATS and MOM12 scored $6.99\,\%$, $6.38\,\%$, $7.15\,\%$, $6.99\,\%$ and $5.45\,\%$, respectively. MSE is also the lowest for STAE compared to other models. Note that MSE here is computed using cross-sectionally standardised monthly returns (as used in network training and validation), not raw returns in Table 4.1. Comparing IC of MOM12 in the validation set to the out-of-sample set, IC fell from $5.45\,\%$ to $1.77\,\%$. Similarly for STAE, mean IC fell from $8.50\,\%$ to $2.76\,\%$. This indicates a general decline in return predictability and is consistent with our observations in Figure 4.5.

4.3.3 Explaining the predictions of STAE

In this section, we examine the predictions of STAE in relations to two well-known anomalies in finance literature — the *momentum* and *reversal* effects.

Jegadeesh and Titman (1993) found two patterns in U.S. stock prices — a medium term trending effect (termed *momentum*, as discussed in Section 4.1), and a short term reversal effect (MOM1) where stocks that rose (fell) the most over the current month tend to reverse in the subsequent month. If the only pattern that exists in prices is momentum, then we expect predictions produced by the neural network to be highly correlated with MOM12. Conversely, if the only pattern that exists in stock prices is reversal (i.e., stock prices are oscillating within a range), then neural network predictions will be correlated with MOM1. We conjecture that the two patterns can be conceptualised as alternating periods of trending and reversal patterns, as illustrated in Figure 4.6.

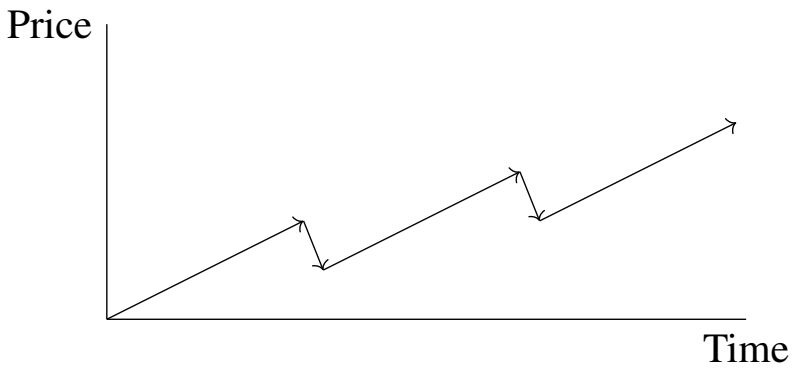


FIGURE 4.6: A hypothetical illustration of momentum and reversal patterns in stocks.

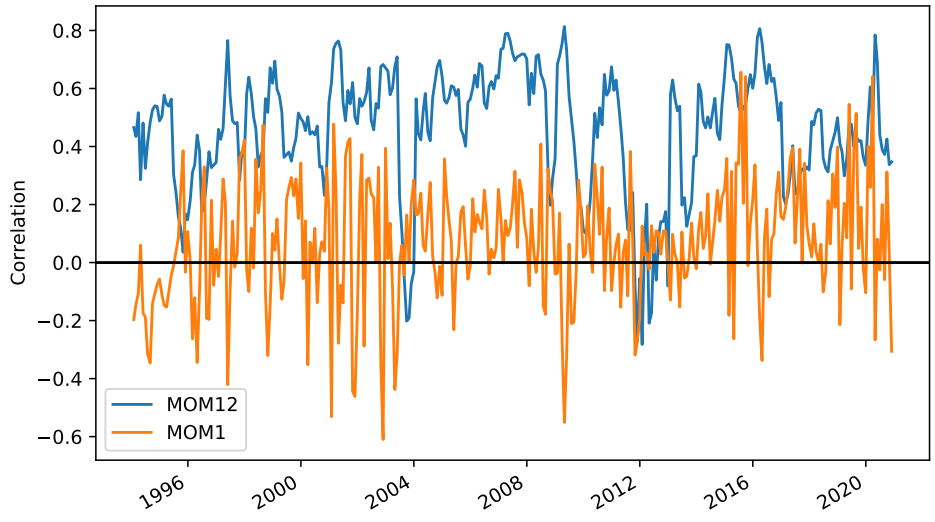


FIGURE 4.7: Cross-sectional correlations of the ensemble prediction of STAE to MOM12 and MOM1.

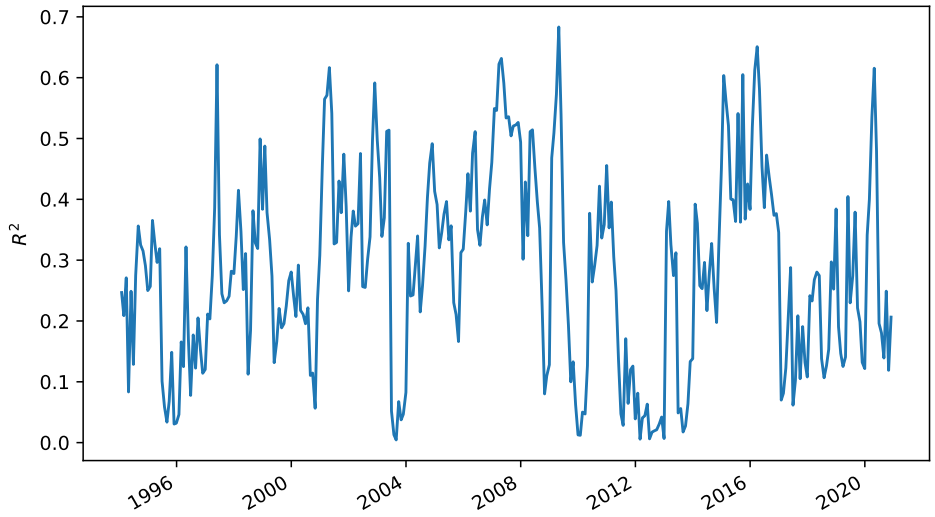


FIGURE 4.8: R^2 of the cross-sectional regression: $\hat{y}_i^{(STAE)} = \beta_0 + \beta_1 MOM12_i + \beta_2 MOM1_i$. Mean R^2 is 28 %.

Momentum and reversal patterns are simple 12-month and 1-month change in price, respectively. We argue that if a neural network were to learn complex, non-linear patterns from stock prices directly, the resultant predictions will be correlated with both MOM12 and MOM1. Moreover, when regressing ensemble predictions of STAE ($\hat{y}_i^{(STAE)}$) on MOM12 and MOM1 scores,

$$\hat{y}_i^{(STAE)} = \beta_0 + \beta_1 MOM12_i + \beta_2 MOM1_i,$$

we would expect the R^2 to be relatively low. Figure 4.7 records the cross-sectional correlations of ensemble predictions of STAE to MOM12 and MOM1. Mean correlations to MOM12 and MOM1 are 0.46 and 0.07, respectively. This suggests that STAE's predictions tend to be driven by trends, as shown by the relatively higher correlation to MOM12 than MOM1. Correlations to both momentum and reversal patterns are highly variable over time. Maximum (minimum) correlations to MOM12 and MOM1 are 0.81 (-0.28) and 0.66 (-0.61), respectively. We note that correlation to MOM12 tends to be lower after a market downturn, namely the end of 2003 (right after the Dot-com bubble) and 2011-12 (after global financial crisis). However, we do not observe a fall in correlation in 2020 during the pandemic. This may be because the 10-year rolling window is chronologically split into 7 years of training data and 3 years of validation data (used for early stopping). Patterns observed during a crisis are only visible to the network 3 years after they occurred. This underlies the reason we are yet to observe a divergence in correlation to MOM12 in 2020. Figure 4.8 records R^2 of regressing ensemble predictions of STAE on MOM12 and MOM1. Mean R^2 is relatively low, at $28\,\%$, indicating that STAE is extracting non-trivial patterns from stock prices that cannot be explained by simple trend and reversal patterns.

4.3.4 Further analysis of the reconstruction task

In this section, we provide further analysis on the regularisation effects of the reconstruction task.

The benefit of ensembling can be illustrated by analysing the expected loss of the ensemble predictor (Goodfellow et al., 2016),

$$\begin{aligned} \mathrm{E}\left[\left(\frac{1}{U}\sum_{i=1}^{U}\epsilon_i\right)^2\right] &= \frac{1}{U^2}\,\mathrm{E}\left[\sum_{i=1}^{U}\left(\epsilon_i^2 + \sum_{j\neq i}\epsilon_i\epsilon_j\right)\right] \\ &= \frac{v}{U} + \frac{U-1}{U}c. \end{aligned} \tag{4.2}$$

where U is number of predictors in the ensemble, $\epsilon_i \sim \mathrm{N}(0, v)$ is the error incurred by model i, which is assumed to be drawn from a $\mathrm{N}(0, v)$ distribution, and c is the expected

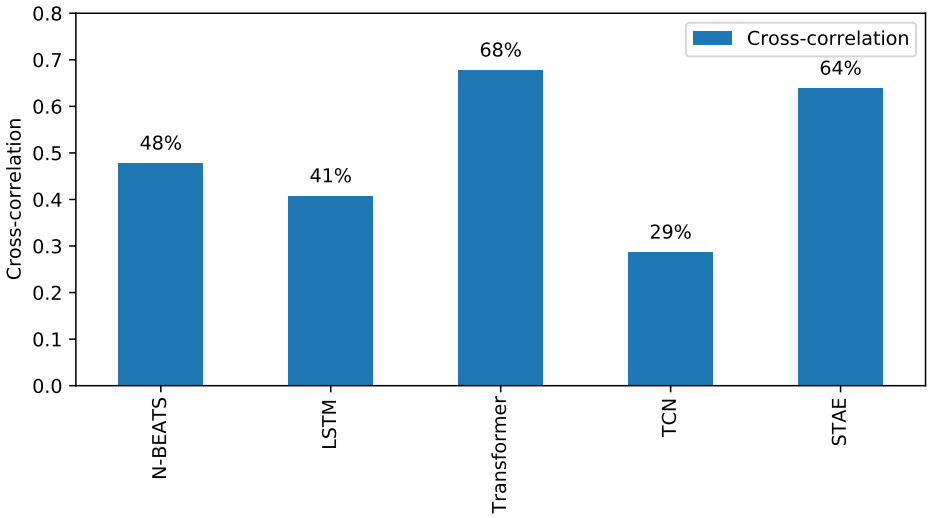


FIGURE 4.9: Mean cross-correlation of every network to every other network in the ensemble (of 10) for each model used in Section 4.3.2.

covariance between any two models of the ensemble. Equation (4.2) shows that expected loss of the ensemble predictor is lower bound by the variance of individual models scaled by size of the ensemble. That is, if each model of the ensemble is independent of all other models, then expected loss of the ensemble predictor decreases logarithmically by size of the ensemble. This "diversification" benefit is offsetted by positive correlations between models in the ensemble. The higher the correlation between models of the ensemble, the higher the expected loss of the ensemble. Due to random weight initialisation and non-convexity, every neural network will be different, even though they are trained on the training set. In Table 4.1, we observe that ensembling has a greater positive impact on TCN and LSTM than STAE and transformer. This is due to higher cross-correlation between networks within each ensemble for STAE and transformer, as shown in Figure 4.9. The mean cross-correlations for STAE and transformer are 0.64 and 0.68, respectively. These are significantly higher than LSTM and TCN, at 0.41 and 0.29. In the case of STAE, we hypothesise that the auxiliary learning task is imposing a non-parametric structure on the representation of the sequence. Thus, predictions by different networks are more correlated.

Next, we investigate whether higher weight assigned to the auxiliary task increases correlation and its impact on forecast accuracy. In this experiment, we fix the encoder kernel size to 2 with

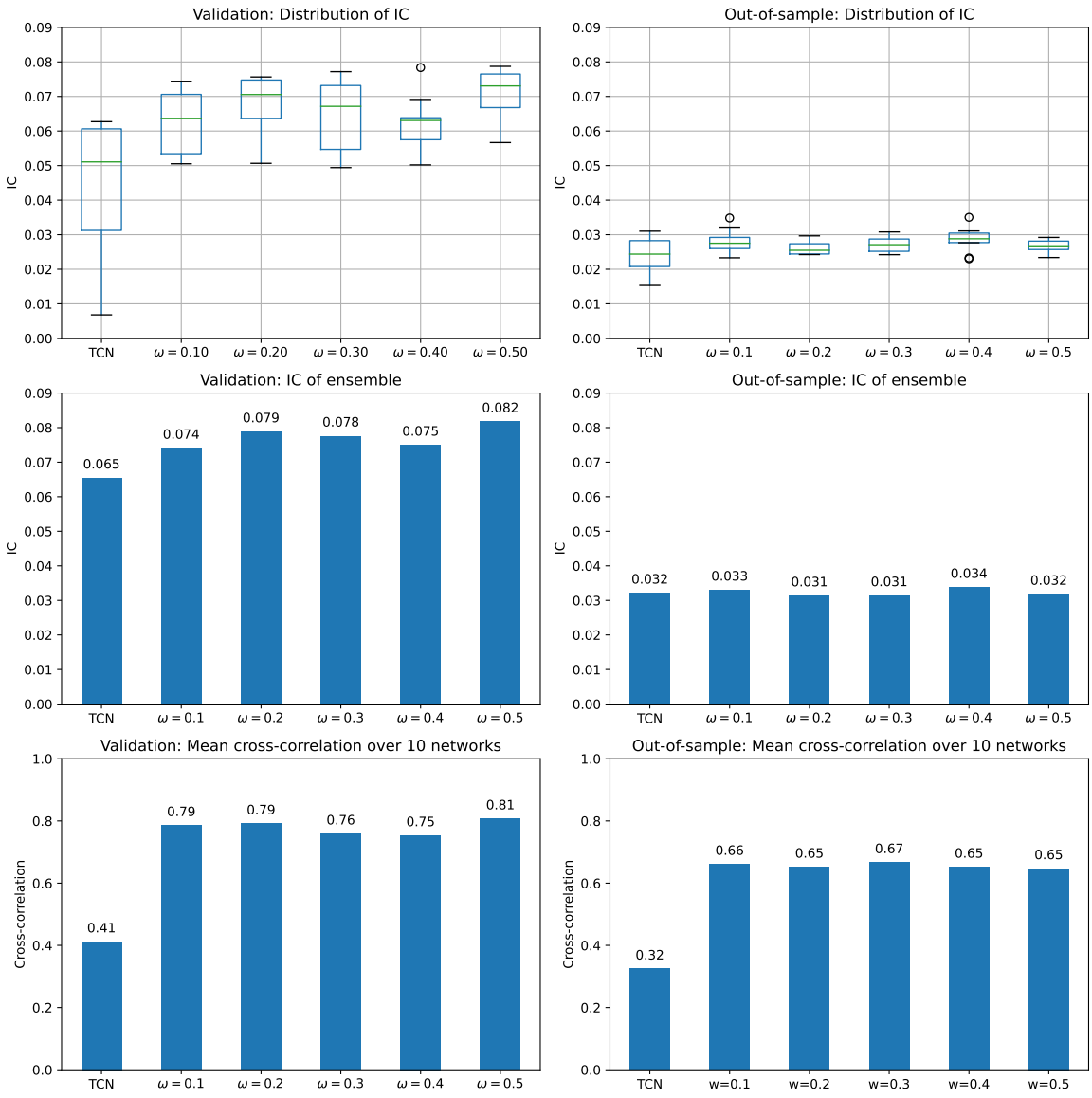


FIGURE 4.10: **Top row**: Distribution of IC for each network of the ensemble for TCN and STAE at different auxiliary loss weights (ω). **Middle row**: IC of the ensemble predictor for TCN and STAE. **Bottom row**: Mean cross-correlation between the predictions of each network of the ensemble.

16 filters, and only vary auxiliary loss weight $\omega \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. We compare STAE at various ω against TCN (i.e., $\omega = 0$) on IC of each network (in an ensemble of 10), IC of the ensemble predictor (i.e., mean prediction of the 10 networks), and average cross-correlation of each network to every other network in the ensemble. This experiment is performed on the validation set and in the out-of-sample set. Figure 4.10 records results of the experiment. In

the top row, we observe that even a small weight to the auxiliary task (at 0.1) is beneficial to IC. All values of ω achieved better median IC (over ensemble of 10) than TCN but the trend between IC and ω is not monotonic. The range of IC varies widely for TCN in both the validation set and the out-of-sample set. As all networks use the same hyperparameters, this indicates that TCN is prone to being stuck in unfavourable local minimas. In the middle row, we observe that all values of ω achieved better mean IC for the ensemble prediction than TCN in the validation set. All models achieved similar IC in the out-of-sample set. As discussed in Section 4.3.2, we conjecture that increasing market efficiency has placed an upper bound on the information content of a stock's own price history. Finally, in the bottom row, we observe that STAE has materially higher cross-correlation between networks in the ensemble than TCN, even at low ω. Cross-correlation does not appear to escalate with ω. Based on these observations, we conclude that the auxiliary task is beneficial to the prediction task, even at a low weight. Finance literature has documented evidence of declining stock return predictability (e.g., Brogaard and Zareei, 2022) — an empirical finding we have also confirmed in Chapter 3 and this chapter. Main stream finance theory (the efficient market hypothesis, as discussed in Section 1.3) relates stock return predictability to the market's information processing efficiency. Hypothesising that market efficiency is improving over time, information content of price history is higher in the validation set (1991–1993) than out-of-sample set (1994–2020), a higher ω appears to be more beneficial when information content is high.

4.4 Conclusion

In this chapter, we propose to use an autoencoder to regularise a TCN for time-series stock return forecasting. We argue that this is beneficial to the supervised learning task as the convolutional filters are constrained to learn features that are useful for reconstruction of the original input and for prediction. Thus, representation sharing will reduce the likelihood of the filters learning spurious features and improve generalisation in noisy environments such as financial markets. We propose STAE, by augmenting TCN with a convolutional decoder for the auxiliary task and show that STAE provides better forecasting performance than TCN

in predicting U.S. stock returns using a time-series of TRI. The reconstructed input by the decoder also assists the user in interpreting the features learnt by the network. We show that neural networks can learn features from (transformed) price series directly, eliminating the need for handcrafted features.

There are two potential extensions to this work. On the topic of improving financial time-series forecasting, in this work, we have demonstrated that STAE outperforms other neural network architectures in forecasting stock returns relying solely on a stock's own price history. We observe a degradation in predictability over time, which we attribute to improving market efficiency and declining information content of prices. Future work can investigate providing the neural network with more information about the stock, such as its size, *OHLCV* (opening price, day's high, day's low, closing price and traded volume), CAPM beta (as discussed in Section 1.3) and measures of business performance[15]. In particular, we observe a sharp fall in decile returns when the market turns (e.g., in March 2009 and April 2020). This is an example of exogenous shock. Informing the neural network with the prevailing market condition may potentially improve its ability to anticipate turning points. To combat concept drift, observations can be time-weighted, or models can be trained in an online manner (such as using the OES algorithm as introduced in Chapter 3). For clarity, in this chapter, we have focused solely on the time-series forecasting in noisy environment problem. We hypothesise that regularising a neural network using an autoencoder has general applicability in other noisy learning environments, outside of financial time-series forecasting. In Section 6.2, we discuss ways of combining the STAE introduced in this chapter with the OES algorithm to train neural networks that can adapt to time-varying DGP and remain robust to noise.

On the topic of improving neural network forecasting in general noisy environments, we have demonstrated that the addition of an auxiliary reconstruction task helped regularise a neural network. We observe that the auxiliary task increased cross-correlation between networks in the ensemble, which decreased the effectiveness of ensembling. Potential ways to decrease cross-correlation are to use *bagging* (Hastie et al., 2020), where both features and time periods are randomly dropped to increase diversity within the ensemble, or different look

[15]Similar to the inputs used in Chapter 3, such as accounting measures of profitability and firm valuation. However, differing to Chapter 3, we can provide a time-series of these metrics instead of just the cross-section.

back windows. The auxiliary task enforces a non-parametric functional form on the latent representation of the sequence, similar to imposing a linear trend shape constraint in linear models. A potential improvement is to combine STAE with attention (Vaswani et al., 2017), where the auxiliary task provides the non-parametric overall trend and attention is applied on deviations from the trend. This decomposition combines a "noise-robust" component with an attention-component that focuses on small intricacies. We also hypothesise that the auxiliary task would also benefit LSTM and transformers. Thus, a positive finding in using a supervised LSTM autoencoder would add to the body of evidence that an auxiliary reconstructon task is beneficial to learning in financial markets.

So far in this thesis, we have examined both cross-sectional and time-series forecasting of stock returns using neural networks. In both applications, outputs of the neural network are point estimates conditional on the input. However, if we were to "bet" on the predictions of a neural network, we need to ask — *how confident are we in the predictions?* In Chapter 5, we will examine methods of incorporating elements of statistical models to provide both the conditional mean and conditional variance of the predictions.

CHAPTER 5

Quantifying neural network uncertainty under volatility clustering

---

Time-series with time-varying variance pose a unique challenge to uncertainty quantification methods. Time-varying variance, such as volatility clustering as seen in financial time-series, can lead to large mismatch between predicted uncertainty and forecast error. Building on recent advances in neural network uncertainty quantification literature, we extend and simplify Deep Evidential Regression and Deep Ensembles into a unified framework to deal with uncertainty quantification under the presence of volatility clustering. We show that a Scale Mixture Distribution is a simpler alternative to the Normal-Inverse-Gamma prior that provides favorable complexity-accuracy trade-off. To illustrate the performance of our proposed approach, we apply it to two sets of financial time-series exhibiting volatility clustering: cryptocurrencies and U.S. equities.

5.1 Introduction

Asset returns are known to exhibit irregular bursts of high volatility that cluster in time (termed *volatility clustering*; Cont, 2001). This poses a challenge to practitioners during portfolio construction which involves the trade-off of return and risk. To motivate the discussion, consider the following simple thought experiment. Suppose an investor has a model that can perfectly forecast next day's asset returns and that the investor's goal is to maximise terminal wealth. Then, on each day, the most rational decision would be to place all of the investor's wealth into the asset with the highest expected return on the next day. Next, suppose that the investor's model is a noisy estimator of future asset returns. Then, the investor may choose to diversify across multiple assets. This intuition serves as the basis

of mean-variance portfolio optimisation (Equation (1.8)) discussed in Section 1.4 and has led to the development of various models for optimal bet allocation that depend on some measures of risk, such as Kelly criterion (where optimal bet size is proportional to expected return divided by variance of expected return[1] which can be replaced by forecast uncertainty; Kelly, 1956; Byrnes and Barnett, 2018) and Bayesian-based portfolio optimisation (Black and Litterman, 1991). Forecast uncertainty can also serve as advanced warning to protect the portfolio from increasing risk. For example, if forecast uncertainty reaches a certain threshold, an investor could purchase portfolio insurance (e.g., put options which allow the investor to sell stocks to the issuer of the options at a pre-agreed price) or liquidate positions to reduce risk.

Forecast uncertainty has an important role in many applications. Such quantity is easy to obtain for statistical models such as linear regression. However, *classical* neural networks for regression problems are typically trained using MSE and provide point estimates for the mean prediction conditional on the input without regards for the conditional variance (see Goodfellow et al., 2016). As a modeller (in our case, an investor), one is concerned with *predictive uncertainty* (Gawlikowski et al., 2021). This is the total uncertainty around a point estimate. Predictive uncertainty can be decomposed into (Gruber et al., 2023): *aleatoric uncertainty*, and *epistemic uncertainty*. Aleatoric uncertainty originates from the stochastic relationship between input variable X taking value x and output variable Y (Gruber et al., 2023). As long as the conditional distribution of $Y|x$ is not degenerate (i.e., Y cannot be perfectly predicted), there will always be aleatoric uncertainty. Aleatoric uncertainty does not typically depend on sample size. By contrast, epistemic uncertainty is attributable to the model and typically scales inversely with sample size (Meinert et al., 2022). Epistemic uncertainty can be further decomposed into *model uncertainty*, which relates to the correct specification of the model, and *parametric uncertainty*, which relates to the correct estimation of model parameters (Sullivan, 2015; Gruber et al., 2023). Epistemic uncertainty refers to the part of predictive uncertainty that is *reducible* through additional information (e.g., more observations and additional variables). In practice, a clear separation between aleatoric and

[1] Note that this differs to Sharpe ratio, which is return divided by standard deviation of return. Kelly criterion is scaled by variance.

epistemic uncertainties is often impossible. To illustrate, consider the (fair) dice rolling experiment, commonly considered to be a process of pure randomness. However, if the initial position and each rotation of the dice can be measured, then it is possible to predict the outcome of each dice roll (Hora, 1996; Gruber et al., 2023). Thus, what is truly aleatoric (i.e., unpredictability of dice roll) and what is epistemic (i.e., initial position and rotation of the dice are merely missing variables) may be difficult to disentangle from a philosophical perspective.

Traditionally, neural network uncertainty quantification requires the use of Bayesian methods or evaluation of the model in unseen data (Meinert et al., 2022). A Bayesian neural network (BNN) is a full probabilistic interpretation of neural network, by placing priors on network weights and inducing a distribution over a parametric set of functions (MacKay, 1992; Neal, 1996; Gal, 2016). Modern BNNs can be trained using MCMC (e.g., the Metropolis-Hastings algorithm; Hastings, 1970) and *Variational Inference* techniques (Jospin et al., 2022). Jospin et al. (2022) notes four advantages of using BNNs over classical neural networks, with two being relevant to uncertainty quantification. First, Bayesian methods provide a natural approach to uncertainty quantification and are better *calibrated* than classical neural networks (Mitros and Namee, 2019; Kristiadi et al., 2020; Ovadia et al., 2019; Jospin et al., 2022). Second, BNN allows distinguishing between epistemic uncertainty and aleatoric uncertainty. However, despite their advantages, MCMC-based methods are computationally expensive (Quiroz et al., 2019). Thus, limiting the applicability of BNNs.

Recent advances (see Gawlikowski et al., 2021 for a recent survey) have focused on predicting the conditional distribution that is most likely to have generated the data and thus bridging the gap between BNNs and classical neural networks. In particular, using a neural network to generate parameters of a conditional distribution that is assumed to have generated the data (Lakshminarayanan et al., 2017; Amini et al., 2020) offers an attractive trade-off between adequately quantifying uncertainty and avoiding the computational cost of a full Bayesian treatment. In Lakshminarayanan et al. (2017) (the *Ensemble* method, also know as *Deep Ensembles*), regression target y is assumed to be drawn from $y \sim \mathrm{N}(\mu, \sigma^2)$, where N is the Normal distribution, μ is the expectation of y and σ^2 models aleatoric uncertainty. In this

setup, σ^2 is incapable of quantifying epistemic uncertainty. Lakshminarayanan et al. (2017) addressed this by using an ensemble of neural networks with randomly initialised weights. Each network settles in a different local minima and produces different μ and σ^2 for the same input. The variance of μ across the ensemble thus provides an estimate of epistemic uncertainty. Addressing this shortcoming, Amini et al. (2020) (the *Evidential* method, also know as *Deep Evidential Regression*) proposed to place an evidential prior[2], the NIG, on μ, σ^2. In this construct, prediction μ is assumed to be drawn from the priors: $\mu \sim \mathrm{N}(\gamma, \sigma^2\nu^{-1})$ and $\sigma^2 \sim \mathrm{InvGam}(\alpha, \beta)$, where σ^2 remains as an estimate of aleatoric uncertainty, InvGam (or IG) is the Inverse-Gamma distribution and $\sigma^2\nu^{-1}$ (with $\sigma^2 \sim \mathrm{InvGam}(\alpha, \beta)$) is estimated epistemic uncertainty. Epistemic uncertainty is linked to aleatoric uncertainty via ν, which is learnt from the data. The marginal distribution of a Normal likelihood with NIG prior is the Student's t-distribution. This mimics a Bayesian setup and circumvents the costly computational burden of MCMC methods by analytically integrating out unobserved variables. Ensemble and Evidential require only minimal modifications to a conventional neural network architecture — requiring only the NLL function of the marginal distribution as loss function and a new output layer. Evidential has been applied to navigation (Liu et al., 2021; Cai et al., 2021; Singh et al., 2022) and medical fields (Soleimany et al., 2021; Li and Liu, 2022), and has been extended into the multi-task learning domain (Oh and Shin, 2022). Multivariate models related to Evidential include the *Natural Posterior Network* which also uses a conjugate prior (NIG for regression problems and Dirichlet for categorical classification problems; Charpentier et al., 2021), and *Regression Prior Networks* which uses a Normal-Wishart prior (Malinin et al., 2020).

However, more recent works have highlighted weaknesses of the Evidential method. *Scoring rules* are a class of loss functions that measure the discrepancy between a predicted distribution and the observed distribution (Gneiting and Raftery, 2007). A scoring rule is *proper* if the score is maximised when the discrepancy is minimised, and is *strictly proper* if the maximum is unique. Thus, strictly proper scoring rules provide attractive loss functions for scoring probabilistic forecasts. Evidential can be interpreted as a hierarchical method with a prior

[2] In contrast to conventional priors in Bayesian inference where the modeller has to specify the parameters of the prior distribution, the evidential prior (e.g., NIG in Evidential) learns these hyperparameters from the data. Note that NIG is a conjugate prior to the Normal distribution (Bernardo and Smith, 2000).

distribution that controls the data distribution. Bengs et al. (2023) argues that in order for hierarchical methods such as Evidential to comply with the requirements of proper scoring rules, rather than training on observable values of y, the predictor must be trained on the imaginary distribution around each observation that depicts its uncertainty, which cannot possibly exist. This requirement stems from the definition of proper scoring, which requires the learner be scored against the "ground truth". As α and β relate to the prior distribution in Evidential, they are not directly observed as data. Thus, hierarchical methods that estimate both the prior and likelihood parameters lack theoretical guarantees on the robustness of their estimated distributions. Similarly, Meinert et al. (2022) argued that unlike aleatoric uncertainty, epistemic uncertainty has no "ground truth" and is difficult to estimate objectively. To motivate this argument, Meinert et al. (2022) used the example of points lined up perfectly in a straight line. If one point is perturbed such that the points no longer form a straight line. Without relying on a-priori assumptions, it is impossible to perform point-wise separation of aleatoric and epistemic uncertainties (i.e., whether the single deviation is due to noise or the correctness of the linear model and its estimated slope). The marginal t-distribution of NIG is overparameterised, which leads to the finding that it is possible to minimise the NLL irrespective of ν (interpreted as "strength of the data" in Amini et al., 2020). As a further critique of the network architecture, we note that all four hyperparameters of Evidential are derived from the same latent representation outputted by the last hidden layer. The four hyperparameters can have vastly different scales (e.g., in our motivating application, γ is in scale of 0.01, while ν is in scale of 10). We consider this feature to be a weakness of these approaches as the latent representation has to provide a sufficiently rich encoding to linearly derive all hyperparameters of the distribution. Nonetheless, successful applications of Evidential on real world datasets has led Meinert et al. (2022) to conclude that Evidential is a heuristic to Bayesian methods and may be appropriate for applications that aim to capture both aleatoric and epistemic uncertainties but do not demand an accurate distinction between them, such as our motivating application.

In this work, we are concerned with neural network uncertainty quantification for time-series that exhibit *time-varying variance*, such as time-series of asset returns. We combine and extend Ensemble and Evidential into a framework (the *Combined* method) for quantifying

predictive uncertainty of this class of time-series. We propose to formulate the problem using the SMD, a simpler alternative to the NIG prior, to address some of the shortcomings highlighted by Meinert et al. (2022) and Bengs et al. (2023). In SMD, a sole Gamma prior is placed on the scaling factor of variance of the Normal distribution, rather than both the mean and variance in Evidential. This is motivated by our asset return forecasting application, where the mean is typically close to zero (in scale of 0.01) and thus uncertainty is negligible, and volatility is significantly larger (standard deviation in scale of 0.1). This is consistent with fitting a return series with models such as Generalised Autoregressive Conditional Heteroskedasticity (GARCH) (Bollerslev, 1986) in which the mean process is typically assumed zero or first order autoregressive (Carroll and Kearney, 2009). Integrating out the scaling factor of SMD results in a marginal t-distribution, of which its variance indicates the predictive uncertainty. Epistemic uncertainty is assumed to be the difference between variance of the marginal t-distribution and variance of the assumed Normal data distribution. This simplification trades off granular attribution of aleatoric and epistemic uncertainties afforded by the NIG prior but allows the reduction of the number of effective parameters by one and resolves the overparameterisation of NIG, as highlighted by Meinert et al. (2022). We also propose a novel architecture to model parameters of the marginal distribution using disjoint subnetworks, rather than a single output layer as in Ensemble and Evidential. We show through an ablation study in Section 5.4.3 that this is crucial to forecasting predictive uncertainty that closely tracks forecast error when the time-series exhibit volatility clustering. As both forecast accuracy and estimation of predictive uncertainty are important to our motivating application, we incorporate model averaging into our Combined method and show that it significantly improves forecast accuracy without significantly changing the estimated predictive uncertainty. This work also provides a template for uncertainty quantification in time-series that exhibit volatility clustering, such as time-series of asset returns.

To illustrate our contributions, we apply our proposed method to cryptocurrency and U.S. equities time-series forecasting. Cryptocurrencies are an emerging class of digital assets. They are highly volatile and frequently exhibit price bubbles (Fry and Cheah, 2016; Hafner, 2018; Chen and Hafner, 2019; Núñez et al., 2019; Petukhina et al., 2021), with large volumes of high frequency data (e.g., prices in hourly intervals) freely available from major exchanges.

This makes cryptocurrencies an ideal testbed for uncertainty quantification methodologies in financial applications. Given the extreme levels of volatility, we view cryptocurrencies as one of the most challenging datasets for this type of application. A comparison in U.S. equities is also provided which illustrates performance in conventional financial time-series. In the rest of this paper, we first describe the setup of our motivating application (asset return forecasting) in Section 5.2.1 and review of related works in Section 5.2.2. We describe our proposed framework in Section 5.3. Data description and empirical results of applying Ensemble, Evidential and Combined on cryptocurrency are presented in Section 5.4.1 and U.S. equities in Section 5.4.2. An ablation study analysing the benefits of each of our proposed enhancements is presented in Section 5.4.3. Whilst this paper is focused on uncertainty quantification in time-series that exhibit volatility clustering, in Appendix A7.1, we also provide a direct comparison to Evidential and Ensemble using the UCI benchmark datasets (non-time-series), as previously analysed in Hernández-Lobato and Adams (2015), Gal and Ghahramani (2016), Lakshminarayanan et al. (2017), and Amini et al. (2020). Finally, concluding remarks are provided in Section 5.5.

5.2 Preliminaries

5.2.1 Problem setup

The basic setup of the problem in this chapter follows that of Chapter 4. At every period $t \in \{1, \ldots, T\}$, an investor observes price history up to t and uses the preceding $\{K \in \mathbb{Z} \,|0 < K < t\}$ period returns to forecast one-step ahead returns. Similar to Chapter 4, we define an asset's return at time t as the log difference in price $r_t = \log p_t - \log p_{t-1}$ and, consistent with empirical findings in finance literature (Pesaran and Timmermann, 1995; Cont, 2001), we assume that the DGP is time-varying:

$$r_t \sim \mathrm{N}(\mu_t, \sigma_t^2). \tag{5.1}$$

Let $\boldsymbol{\zeta}_t = (\mu_t, \sigma_t^2)$ be parameters of the assumed DGP, $\boldsymbol{x}_{t-1} = \{r_{t-K}, r_{t-K+1}, \ldots, r_{t-1}\}$ be a K-length input sequence[3] using returns up to $t-1$ and $y_{t-1} = r_t$ be forward one period return. The training dataset is comprised of $\mathcal{D}_t = \{(\boldsymbol{x}_{q-1}, y_{q-1}) | q \in \mathbb{N} : q \leq t\}$ input-output pairs[4] and is essentially a set of sequences formed with a K-length sliding window and their corresponding regression targets. Our goal is to forecast y_t (which corresponds to r_{t+1}). At each t, the investor's goal is to solve the optimisation problem[5],

$$\boldsymbol{\theta}_t = \underset{\boldsymbol{\theta}^*}{\operatorname{argmin}} - \sum_{q=K}^{t-1} \log \mathrm{p}(y_q | F(\boldsymbol{x}_q; \boldsymbol{\theta}^*)), \tag{5.2}$$

where $F(\boldsymbol{x}; \boldsymbol{\theta})$ is a neural network with input $\boldsymbol{x}$ and parameters $\boldsymbol{\theta}$, $\boldsymbol{\theta} = \bigcup_{\ell=1}^{L} \{\boldsymbol{W}^{(\ell)}, \boldsymbol{b}^{(\ell)}\}$ is the set of network weights and biases and, in this context, $\mathrm{p}(y | F(\boldsymbol{x}; \boldsymbol{\theta}))$ is the likelihood of observing y based on the outputs of neural network $F(\cdot; \cdot)$ and the assumed marginal distribution. In other words, the investor is concerned with recovering the parameters $\hat{\boldsymbol{\zeta}}_t = (\hat{\mu}_t, \hat{\sigma}_t^2) \coloneqq F(\boldsymbol{x}_t; \boldsymbol{\theta}_t)$ that are most likely to have generated the observed data. In this setup, $\hat{\sigma}_t^2$ can be interpreted as an estimate of aleatoric uncertainty and is an estimate of the contemporaneous variance of the DGP at time t.

There are two parts to this problem. The first part concerns uncertainty quantification specifically for time-series that exhibit volatility clustering and is the primary focus of this work. The second part concerns advancing methods of uncertainty quantification across general applications. In Appendix A7.1, we show that our proposed approach can still benefit non-time-series problems in spite of it being designed to deal with a series of data points indexed in time order and exhibiting volatility clustering.

[3] For illustrative purposes, we have stated that the sequence only contains returns r_t. However, as discussed in Section 5.3.2, we also include squared returns r_t^2 as part of the input sequence.

[4] Note that at each portfolio selection period t, the training set can at most contain data up to $t-1$ as we have not yet observed r_{t+1}.

[5] For clarity, the case of a single asset is shown. At each t, there are N assets and the dataset is typically in a $t \times N$ layout. It is easy to see the generalisation of Equation 5.2 over N assets, where the average loss is calculated over $(t-K-1) \times N$ instances.

5.2.2 Related work

Recent advances in neural network uncertainty quantification, such as Ensemble and Evidential, have focused on outputting parameters of the assumed data distribution. As these works were originally proposed for non-time-series problems, in discussing these works, we have left out time index t but note that in our motivating application, variables are indexed by t (e.g., the assumed DGP in Equation (5.1)). The neural networks are trained using procedures similar to *maximum likelihood estimation*. In Ensemble (Lakshminarayanan et al., 2017), regression target y is assumed to be drawn from $y \sim \mathrm{N}(\mu, \sigma^2)$, where μ is the forecast of y and σ^2 models aleatoric uncertainty. The output layer of the neural network is modified to output $\boldsymbol{\zeta} = (\mu, \sigma^2)$, and the network is trained using the Gaussian NLL. As this formulation is incapable of quantifying epistemic uncertainty, Lakshminarayanan et al. (2017) used an ensemble of neural networks with randomly initialised weights to provide an empirical estimate of epistemic uncertainty. Addressing this, Amini et al. (2020) proposed to place an *evidential prior*, the NIG distribution, on the model parameters μ, σ^2 of the Normal data distribution:

$$\begin{aligned} \text{Data} &: y \sim \mathrm{N}(\mu, \sigma^2) \\ \text{NIG prior} &: \mu \sim \mathrm{N}(\gamma, \sigma^2 \nu^{-1}), \quad \sigma^2 \sim \mathrm{InvGam}(\alpha, \beta), \end{aligned} \tag{5.3}$$

where μ is assumed to be drawn from a Normal prior distribution with unknown mean γ and scaled variance $\sigma^2\nu^{-1}$, ν is a scaling factor for σ^2, and shape $\alpha > 1$ and scale $\beta > 0$ parameterise the Inverse-Normal (IG) distribution[6]. We require $\alpha > 1$ to ensure the mean of the marginal distribution is finite.

In this construct, parameters of the posterior distribution of y is $\boldsymbol{\zeta} = (\gamma, \nu, \alpha, \beta)$. Epistemic uncertainty is reflected by the uncertainty in μ, which is assumed be a fraction of σ^2 and is itself assumed to be drawn from an IG distribution. This fraction is controlled by ν, which is learnt from the data and, in an abstract sense, varies according to the amount of information in the data. Parameter ν is interpreted as the number of virtual observations for the mean

[6]Time index t has been omitted for brevity and legibility. Note that variables in this section are indexed by time for each asset: $\{y_t, r_t, \mu_t, \sigma_t^2, \gamma_t, \nu_t, \alpha_t, \beta_t\}$.

parameter μ. In other words, ν virtual instances of μ are assumed to have been observed in determining the prior variance of μ (Jordan, 2009; Amini et al., 2020).

For the NIG prior in 5.3, the marginal distribution of μ after integrating out σ^2 is a non-standardised Student's t-distribution (denoted St; Bernardo and Smith, 2000),

$$\begin{aligned} \mathrm{p}(\mu|\gamma,\nu,\alpha,\beta) &= \int_{\sigma^2=0}^{\infty} \mathrm{p}_{\mathrm{N}}(\mu|\gamma,\sigma^2\nu^{-1})\mathrm{p}_{\mathrm{IG}}(\sigma^2|\alpha,\beta)\,\mathrm{d}\sigma^2 \\ &= \mathrm{St}\left(\gamma,\frac{\beta}{\nu\alpha},2\alpha\right), \end{aligned} \tag{5.4}$$

using the fact that $\sigma^2 \sim \mathrm{InvGam}(\alpha,\beta)$ corresponds to $\sigma^{-2} \sim \mathrm{Gam}(\alpha,\beta)$. Hence, assigning a $\mathrm{Gam}(\alpha,\beta)$ prior to precision σ^{-2} in (Equation (5.3)) gives the Normal-Gamma (NG) prior and is equivalent to assigning $\mathrm{InvGam}(\alpha,\beta)$ to σ^2 which gives the NIG prior. The variance of this t-distribution is $\frac{\beta}{\nu(\alpha-1)}$. Predictions based on the NIG prior can be computed as (Amini et al., 2020),

$$\begin{aligned} \text{Prediction} &: \mathrm{E}[\mu] = \gamma \\ \text{Aleatoric uncertainty} &: \mathrm{E}[\sigma^2] = \tfrac{\beta}{\alpha-1} \\ \text{Epistemic uncertainty} &: \mathrm{Var}[\mu] = \tfrac{\beta}{\nu(\alpha-1)}. \end{aligned} \tag{5.5}$$

The marginal variance $\mathrm{Var}[\mu]$ refers to the variance of the marginal t-distribution in (Equation (5.4)) for the NIG prior. We note that ν can also be interpreted as a factor that attributes uncertainty between aleatoric uncertainty ($\frac{\beta}{\alpha-1}$) and epistemic uncertainty ($\frac{\beta}{\nu(\alpha-1)}$). If $\nu = 1$, then total uncertainty is evenly split between aleatoric and epistemic uncertainties.

Whilst not the focus of Amini et al. (2020), we note that epistemic uncertainty can be further decomposed approximately into uncertainties attributable to parameters μ and σ^2. Parameter $\mu|\sigma^2$ is normally distributed with $\mathrm{Var}[\mu|\sigma^2] = \sigma^2/\nu$ (from Equation (5.3)), and $\mathrm{E}[\sigma^{-2}] = \mathrm{E}[\frac{1}{\sigma^{-2}}] \approx \frac{1}{\mathrm{E}[\sigma^{-2}]} = \alpha/\beta$ (from the Gamma distribution of σ^{-2}). This leads to $\mathrm{Var}[\mu|\sigma^2] \approx \frac{\beta}{\nu\alpha}$,

$$\begin{aligned} \text{Model } \mu \text{ uncertainty} &: \mathrm{Var}[\mu|\sigma^2] \approx \tfrac{\beta}{\nu\alpha} \\ \text{Model } \sigma^2 \text{ uncertainty} &: \mathrm{Var}[\mu] - \mathrm{Var}[\mu|\sigma^2] \approx \tfrac{\beta}{\nu\alpha(\alpha-1)}, \end{aligned} \tag{5.6}$$

where the difference between the marginal and conditional variances of μ gives the uncertainty of σ^2.

In this construct, the marginal distribution of y after integrating out μ and σ^2 is a non-standardised Student's t-distribution (Amini et al., 2020),

$$\begin{aligned}\mathrm{p}(y|\gamma,\nu,\alpha,\beta) &= \int_{\sigma^2=0}^{\infty}\int_{\mu=-\infty}^{\infty} \mathrm{p}_{\mathrm{N}}(y|\mu,\sigma^2)\mathrm{p}_{\mathrm{NIG}}(\mu,\sigma^2|\gamma,\nu,\alpha,\beta)\,\mathrm{d}\mu\,\mathrm{d}\sigma^2 \\ &= \mathrm{St}\left(y;\gamma,\frac{\beta(1+\nu)}{\nu\alpha},2\alpha\right). \end{aligned} \tag{5.7}$$

Variance of this t-distribution is $\frac{\beta(1+\nu)}{\nu(\alpha-1)}$, which corresponds to the sum of epistemic and aleatoric uncertainties,

$$\mathrm{Var}[y] = \frac{\beta}{\alpha-1} + \frac{\beta}{\nu(\alpha-1)} = \frac{\beta(1+\nu)}{\nu(\alpha-1)}. \tag{5.8}$$

The corresponding NLL of Equation (5.7) is (Amini et al., 2020),

$$\begin{aligned}\mathcal{L}_{\mathrm{NIG}}(y|\boldsymbol{\zeta}) = &\tfrac{1}{2}\log\left[\tfrac{\pi}{\nu}\right] - \alpha\log\left[2\beta(1+\nu)\right] \\ &+ (\alpha+\tfrac{1}{2})\log\left[(y-\gamma)^2\nu + 2\beta(1+\nu)\right] + \log\left[\frac{\Gamma(\alpha)}{\Gamma(\alpha+\frac{1}{2})}\right]. \end{aligned} \tag{5.9}$$

Equation (5.9) mimics a Bayesian setup, granting classical neural networks the ability to estimate both epistemic and aleatoric uncertainty, and offers an intuitive interpretation of the model mechanics — due to uncertainty in the model parameters, the tails of the marginal likelihood are heavier than a Normal distribution. This has the effect of regularising the network and provides an avenue of estimating epistemic uncertainty. As the distribution of asset returns has heavy tails (Cont, 2001), we argue that the marginal t-distribution also provides a better fit of the data. The implementation is remarkably simple — Equation (5.9) replaces MSE as the loss function (for a regression problem) and the final layer of the network is replaced with a layer that simultaneously outputs four parameters of the marginal distribution. Clearly, modelling of γ and ν by the neural network is direct as they correspond to mean and degrees of freedom of the t-distribution. By contrast, scale of the t-distribution in Equation (5.7) is modelled through a more complex structure ($\frac{\beta(1+\nu)}{\nu\alpha}$), which reflects the two sources of uncertainty in Equation (5.6) with two additional neural network outputs: α and β.

They are the shape and scale parameters of the prior Gamma distribution for precision σ^{-2} which describe distinct characteristics of epistemic uncertainty.

As discussed in Section 5.1, aleatoric and epistemic uncertainties are difficult to disentangle. More recent works have questioned the accuracy of methods, such as Evidential, that directly estimate epistemic uncertainty through minimising the NLL of an assumed marginal distribution. Bengs et al. (2023) argues that a strictly proper loss function for Evidential involves scoring against the distribution around each observation, which cannot possibly exist. Thus, there is no theoretical guarantee that the estimated epistemic uncertainty by Evidential is reliable. Moreover, Meinert et al. (2022) notes that Equation (5.7) is overparameterised, as it is possible to minimise Equation (5.9) irrespective of ν, by: $\frac{\partial}{\partial \nu}\mathcal{L}_{\text{NIG}} = 0$, if $\beta\nu = \frac{1}{1+\nu^{-1}}$ and sending $\nu \to 0$. This is because Equation (5.7) is, by definition, a projection of the NIG distribution, and thus is unable to unfold all of its degrees of freedom unambiguously (Meinert et al., 2022). Through simulation data, Meinert et al. (2022) showed that over the course of neural network training, the estimated ν was related to speed of convergence. Thus, the estimated ν, which controls the ratio of epistemic uncertainty to aleatoric uncertainty, may not be accurate. We note that this is also evident in Equation (5.7), as ν appears in both the numerator and denominator of the scale parameter of the t-distribution in the form of $1 + \frac{1}{\nu}$. Thus, ν relates ambiguously to the the scale parameter of the t-distribution. Motivated by this observation, we propose a simpler formulation, which we detail in Section 5.3.1.

5.3 Uncertainty quantification under volatility clustering

5.3.1 Modelling forecast uncertainty using a scale mixture distribution

As discussed in Section 5.2.2, Evidential provides the ability to perform granular attribution of uncertainty to various parts of the model (e.g., Equation (5.5) and (5.6)). However, this ability comes at the cost of model complexity and the estimated epistemic uncertainty may not be reliable (as discussed in Section 5.2.2). We sought to propose a simpler formulation of the problem than Evidential while offering the ability to quantify predictive uncertainty,

which is the type of forecast uncertainty that we are most concerned about in our motivating application.

We propose to simplify the model by formulating the problem as a SMD[7] (Andrews and Mallows, 1974),

$$y \sim \mathrm{N}(\gamma, \sigma^2\nu^{-1}), \quad \nu \sim \mathrm{Gam}(\alpha, \beta), \tag{5.10}$$

where $\nu > 0$ is the scaling factor, Gam is the Gamma distribution, and $\alpha > 1$ and $\beta > 0$ are the shape and scale parameters of the Gamma distribution, respectively. Our proposed formulation effectively omits the prior on μ and places a prior on ν, the scaling factor of σ^2. We argue that uncertainty of variance can be modelled through either σ^2 or ν. In here, y is assumed to be drawn from $\mathrm{N}(\gamma, \sigma^2\nu^{-1})$, with mean γ and unknown variance $\sigma^2\nu^{-1}$ where ν is a latent variable that introduces uncertainty into the variance of the assumed Normal distribution of y. This allows flexibility to inflate the variance (by minimising ν without inflating σ^2) so as to capture the extremities of the distribution. Relative to Equation (5.7), σ^2 replaces ν in the parameter set when taking the SMD approach as σ^2 has a richer interpretation — it directly indicates the scale of the conditional data distribution. Note that in Equation (5.10), placing a Gamma prior on ν is equivalent to $\sigma^{-2} \sim \mathrm{Gam}(\alpha, \beta)$ as ν and σ^{-2} are indistinguishable in $\sigma^2\nu^{-1}$. However, this is distinct from using a NG prior as there is no Normal prior on μ in Equation (5.10).

The marginal distribution of a Normal distribution with unknown variance (Equation (5.10)) is a non-standardised t-distribution (derivation is provided in Appendix A6),

$$\begin{aligned} \mathrm{p}(y|\gamma, \sigma^2, \alpha, \beta) &= \int_{\nu=0}^{\infty} \mathrm{p}_{\mathrm{N}}(y|\gamma, \sigma^2\nu^{-1})\mathrm{p}_{\mathrm{G}}(\nu|\alpha, \beta)\,\mathrm{d}\nu \\ &= \mathrm{St}\left(y; \gamma, \frac{\sigma^2\beta}{\alpha}, 2\alpha\right). \end{aligned} \tag{5.11}$$

Analogous to Equation (5.7), the shape parameter of this marginal Student's t-distribution is 2α. Equation (5.11) is similar to Equation (5.4) with y replacing μ, and can be interpreted

[7]Time index t has been omitted for brevity and legibility. Note that variables in this section are indexed by time for each asset: $\{y_t, \gamma_t, \sigma_t^2, \nu_t, \alpha_t, \beta_t\}$. We use the same notations in Equation (5.10) as Equation (5.3) where the symbols have the same meaning to improve comparability.

as the Normal distribution being "stretched out" into a heavier tailed distribution due to the uncertainty in its variance. Jointly, $\boldsymbol{\zeta} = (\gamma, \sigma^2, \alpha, \beta)$ are parameters of the SMD distribution and are outputs of the neural network. This has the effect of regularising the mean estimate (γ) and, similar to NIG, provides the ability to handle heavy tails of the distribution that characterise asset returns.

The corresponding NLL of Equation (5.11) (derivation is provided in Appendix A6) is,

$$\mathcal{L}_{\text{SMD}}(y|\boldsymbol{\zeta}) = \log\left[\frac{\Gamma(\alpha)}{\Gamma(\alpha+\frac{1}{2})}\right] + \tfrac{1}{2}\log[2\pi\sigma^2\beta] + (\alpha+\tfrac{1}{2})\log\left[\frac{(y-\gamma)^2}{2\sigma^2\beta}+1\right]. \tag{5.12}$$

Then, $\mathcal{L}_{\text{SMD}}$ is used in place of the marginal likelihood function in Equation (5.2), in which the neural network learns to output parameters in $\boldsymbol{\zeta}$. In Equation (5.10), conditional on the scaling factor ν, the data is normal with variance given by the scale of the marginal t-distribution ($\frac{\sigma^2\beta}{\alpha}$). This variance gives the uncertainty of the data. Since the predictive uncertainty given by the variance of the marginal t-distribution contains both epistemic and aleatoric uncertainties, the difference between predictive and data uncertainties gives the epistemic uncertainty. This is illustrated in Equation (5.13) below:

$$\begin{aligned}
\text{Prediction} &: \mathrm{E}[y] = \gamma \\
\text{Aleatoric uncertainty} &: \mathrm{E}[\tfrac{\sigma^2}{\nu}] \approx \tfrac{\sigma^2\beta}{\alpha} \\
\text{Predictive uncertainty} &: \mathrm{Var}[y] = \tfrac{\sigma^2\beta}{\alpha}\cdot\tfrac{2\alpha}{2\alpha-2} = \tfrac{\sigma^2\beta}{\alpha-1} \\
\text{Epistemic uncertainty} &: \mathrm{Var}[y] - \mathrm{E}[\tfrac{\sigma^2}{\nu}] \approx \tfrac{\sigma^2\beta}{\alpha-1} - \tfrac{\sigma^2\beta}{\alpha} = \tfrac{\sigma^2\beta}{\alpha(\alpha-1)}.
\end{aligned} \tag{5.13}$$

Recall that the result in Equation (5.11) can be interpreted as a Normal distribution being stretched out into a heavier tailed t-distribution when variance is unknown. Kurtosis of the t-distribution is controlled by the shape parameter (2α). In analysing Equation (5.11) and (5.13), we argue that α is analogous to "virtual observations" (ν) in NIG. Epistemic uncertainty $\frac{\sigma^2\beta}{\alpha(\alpha-1)}$ is smaller than aleatoric uncertainty $\frac{\sigma^2\beta}{\alpha}$ by a factor of $\frac{1}{\alpha-1}$, when $\alpha > 2$. Thus, as α increases, both epistemic uncertainty and scale of the marginal t-distribution monotonically decrease. Importantly, epistemic uncertainty also drops relative to aleatoric uncertainty, as the t-distribution converges to the Normal distribution on increasing α. This

stands in contrast to the model with NIG prior (Equation (5.7)), where increasing evidence ν does not monotonically lead to a decrease in scale of the t-distribution.

Our proposed SMD formulation addresses some of the concerns of Meinert et al. (2022) and Bengs et al. (2023). There are essentially three free parameters in Equation (5.11) as $\sigma^2\beta$ together should be treated as one. Parameter β is a redundant parameter as it exists as a product together with σ^2 in both the marginal NLL (Equation (5.12)) and in all three uncertainty measures (Equation (5.13)). Parameters σ^2 and β indicate scales of the Normal and Gamma distributions, respectively. Together, they contribute to the scale of the marginal t-distribution. The number of parameters can be reduced by either reparameterising $\sigma^2\beta$ as a single parameter, or by setting $\alpha = \beta$, which we consider as the more intuitive choice. SMD encapsulates several well-known distributions as special cases. According to Andrews and Mallows (1974) and Choy and Chan (2008), in the case of $\alpha = \beta$, then Equation (5.10) is a Student's t-distribution with 2α degrees of freedom, and is Cauchy if $\alpha = \beta = 1$. If $\alpha \neq \beta$, Equation (5.10) gives the Pearson Type VII (PTVII) distribution which can be re-expressed as a Student's t-distribution in Equation (5.11). As epistemic uncertainty is estimated by the heavy tails of the t-distribution, we can, without loss of generality, set $\alpha = \beta$ and reformulate Equation (5.11) as,

$$\mathrm{p}(y|\gamma, \sigma^2, \alpha) = \mathrm{St}\left(y; \gamma, \sigma^2, 2\alpha\right), \tag{5.14}$$

and the marginal NLL (Equation (5.12)) as,

$$\mathcal{L}_{\text{PTVII}}(y|\gamma, \sigma^2, \alpha, \alpha) = \log\left[\frac{\Gamma(\alpha)}{\Gamma(\alpha + \frac{1}{2})}\right] + \tfrac{1}{2}\log[2\pi\sigma^2\alpha] + (\alpha + \tfrac{1}{2})\log\left[\frac{(y-\gamma)^2}{2\sigma^2\alpha} + 1\right]. \tag{5.15}$$

Comparing Equation (5.14) to the marginal t-distribution of using a NIG prior (Equation 5.7), parameters of this model relate directly to parameters of the t-distribution instead of hyper-parameters of the prior distribution. Hence, mitigating the concerns of Bengs et al. (2023) on hierarchical models and Meinert et al. (2022) on unresolved degrees of freedom. Thus, we argue that SMD offers an attractive trade-off between model complexity and granularity, occupying the middle ground between Ensemble (no prior) and Evidential (prior on both mean and variance).

5.3.2 Architecture of the neural network

For the main application of this work, uncertainty quantification of financial time-series forecasts, we propose a novel architecture for the modelling of distribution parameters, as illustrated in Figure 5.1. To predict $\hat{y}_t$, time-series inputs of both *returns* $(r_{t-K+1}, \ldots, r_t)$ and *log-transformed squared returns* $(\log[r^2_{t-K+1}], \ldots, \log[r^2_t])$ are fed into one or more LSTM layers (Hochreiter and Schmidhuber, 1997). We log-transform squared returns to reduce skewness. The LSTM layers convert each time-series into a latent representation. The latent representation is then fed into four subnetworks, where each subnetwork is comprised of one or more fully connected layers and applies non-linear transformations on the latent representation. This allows the network to model complex relationships between the parameters in $\boldsymbol{\zeta}$ and the sequence. During training, the four parameters outputted by the network and the observed y are fed into the loss function (Equation (5.12)) to compute loss value and gradients, which are backpropagated through the network for weight updates. As noted in Section 5.3.1, we can set $\alpha = \beta$ and reduce the number of subnetworks to three. In other words, the

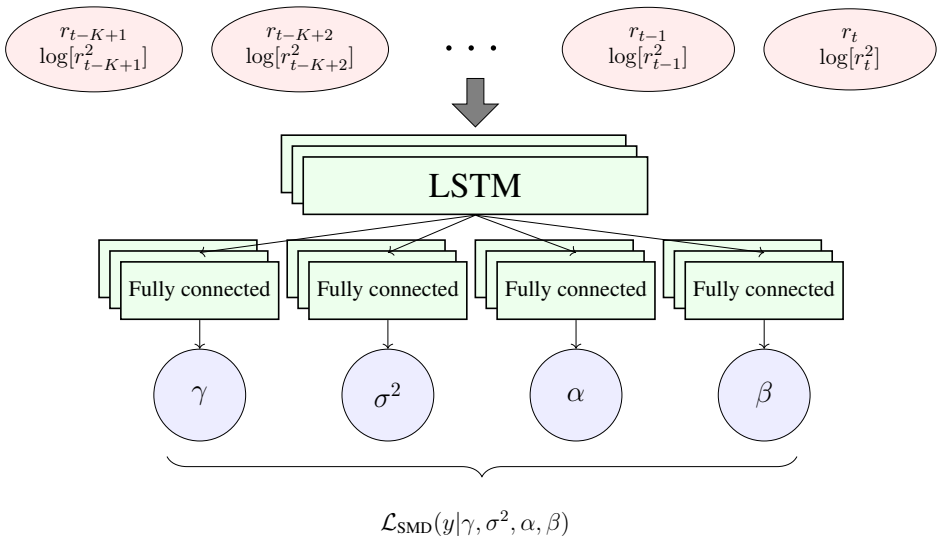


FIGURE 5.1: Input sequence (shaded in red) is passed into one or more LSTM layers. Output from the LSTM layers is then fed into four subnetworks of one or more fully connected layers with ReLU activation. The final layer of each subnetwork is a fully-connected layer with linear activation. Softplus is applied to σ^2, α and β to ensure positivity. During training, the four output values of the neural network together with the observation y are fed into the loss function (Equation (5.12)) to compute loss and gradients.

network architecture illustrated in Figure 5.1 can be modified to output three parameters: $\boldsymbol{\zeta} = (\gamma, \sigma^2, \alpha)$. We have kept β to be comparable to Evidential but provide empirical results in Appendix A8 using the UCI dataset (the same benchmark dataset used in Lakshminarayanan et al., 2017 and Amini et al., 2020, and discussed in Appendix A7.1) to show that the two networks are indeed equivalent. In the following, we explore the proposed design of the architecture in detail.

Lakshminarayanan et al. (2017) and Amini et al. (2020) introduced the `Gaussian` and `NormalInverseGamma` layers as the final layer of a neural network. These final layers output parameters of the posterior distribution. Let $\boldsymbol{a} \in \mathbb{R}^{H^{(I)}}$ be the input vector of the final layer with $H^{(I)}$ dimensions and $H^{(O)}$ be the dimension of the output layer. In the case of the `NormalInverseGamma` layer, $H^{(O)} = 4$. The `NormalInverseGamma` layer outputs,

$$\begin{aligned} \boldsymbol{\zeta} &= \mathrm{O}(\boldsymbol{a}; \boldsymbol{\theta}) = \boldsymbol{a}^{\mathsf{T}} \cdot \boldsymbol{W}^{(O)} + \boldsymbol{b}^{(O)} \\ \gamma = \zeta_1, \quad \nu &= \zeta_2, \quad \alpha = \zeta_3, \quad \beta = \zeta_4, \end{aligned} \tag{5.16}$$

where O denotes the `NormalInverseGamma` output layer, $\{\zeta_{1,\ldots,4}\}$ are 1st, ..., 4th elements of vector $\boldsymbol{\zeta}$, $\boldsymbol{W}^{(O)} \in \mathbb{R}^{H^{(I)} \times H^{(O)}}$ and $\boldsymbol{b}^{(O)} \in \mathbb{R}^{H^{(O)}}$ are weights and bias of the output layer, respectively. Each dimension of $\boldsymbol{\zeta}$ corresponds to each of γ, ν, α and β.

Outputs of the `NormalInverseGamma` layer are linear transformations of a common input $\boldsymbol{a}$ (Equation (5.16)). We argue that this construct is too restrictive for complex applications, such as in quantifying uncertainty of financial time-series forecasts, as detailed in Section 5.4.1. We propose to model each of the four parameters of SMD with its own subnetwork of one or more fully connected layers. This allows for a more expressive modelling of $\boldsymbol{\zeta}$, where each parameter may have complex, non-linear relationships with the input.

Additionally, we enforce constraints on $\sigma^2 > 0$, $\alpha > 1$ and $\beta > 0$ by applying softplus transformation with a constant term, $z' = \log(1 + \exp(z)) + c$, where $z \in \{\sigma^2, \alpha, \beta\}$ and c is the minimum value of the respective parameters. The transformed values constitute the final output of the network: $\boldsymbol{\zeta}' = \{\gamma, (\sigma^2)', \alpha', \beta'\}$. In Section 5.4.1, we show that this modification vastly improves quantification of forecast uncertainty of financial time-series.

For other network architectures, we argue that the same approach can be applied. In the case of a feedforward network, we recommend having at least one common hidden layer that reduces the input to a single latent representation. The latent representation is then passed to individual subnetworks for specialisation. We argue that the common hidden layer allows information sharing across the four parameters, while having no common hidden layer (i.e., if the input is fed into the four disjoint stacks of hidden layers directly) will prevent sharing of information across the stacks.

Machine learning models are typically trained using pooled dataset of historical observations. As such, they learn the average uncertainty within the historical data. However, as noted in Section 5.1, asset returns exhibit time-varying volatility clustering patterns. Thus, we expect predictive uncertainty to be correlated with time-varying variance of the DGP. In other words, predictive uncertainty is high when σ_t^2 of the DGP is high and the model is "surprised" by the volatility. To inform the neural network of the prevailing volatility environment, we propose to include the log of squared returns $\{\log(r_{t-K+1}^2), \ldots, \log(r_t^2)\}$ as part of the input matrix. This follows from the use of squared returns in volatility forecasting literature (Brownlees et al., 2011) and allows the neural network to infer the prevailing volatility environment.

Model averaging, as a special case of ensembling, is a well studied statistical method for improving predictive power of estimators (Breiman, 1996; Goodfellow et al., 2016), and has previously been shown to improve accuracy of financial time-series forecasting (in Chapter 4) and sequential predictions (Raftery et al., 2010). As accuracy of both return forecast accuracy and predictive uncertainty are important in our motivating application, we propose to incorporate model averaging to improve return forecasts at the cost of higher predictive uncertainty estimates. For an ensemble of M models, we compute the ensemble forecast $\tilde{y}$ and predictive variance $\mathrm{Var}[\tilde{y}]$ as,

$$\tilde{y} = \frac{1}{M}\sum_{i=1}^{M}\hat{y}_i, \quad \mathrm{Var}[\tilde{y}] = \frac{1}{M}\sum_{i=1}^{M}(\hat{y}_i^2 + \mathrm{Var}[\hat{y}_i]) - \tilde{y}^2, \tag{5.17}$$

where $\hat{y}_i$ and $\mathrm{Var}[\hat{y}_i]$ are mean and predictive variance of model i, respectively. In Equation (5.17), $\mathrm{E}[\hat{y}^2] > \mathrm{E}[\hat{y}]^2$ (by Jensen's inequality). Thus, predictive uncertainty of the

ensemble will be higher than estimated using the marginal t-distribution alone. In Section 5.4.3, we show that model averaging resulted in significant predictive performance improvement and, despite the higher uncertainty estimates, resulted in the lowest NLL.

Popular tools for modelling time-varying volatility are Autoregressive Conditional Heteroskedasticity (ARCH) (Engle, 1982) and GARCH models. GARCH, when applied to stock returns, assumes the same DGP as Equation (5.1). Time-varying variance σ_t^2 is modelled using an ARMA model (Box et al., 1994). Parameter μ_t can assume a fixed value (e.g., sample mean or 0) or modelled using time-series models such as ARMA (leading to the ARMA-GARCH formulation). In our proposed framework, squared returns are provided as inputs to LSTM in similar spirit to the autoregressive terms of squared returns in GARCH. However, our proposed framework also has few differences to ARMA-GARCH. A neural network offers greater flexibility in modelling and can automatically discover interaction effects between returns and volatility. For example, higher volatility is negatively correlated with future asset returns (known as the *leverage effect*; Cont, 2001). By contrast, modelling of interaction effects in additive models (such as GARCH) requires explicit specification by the user. LSTM can also be interpreted as having dynamic autoregressive orders (as opposed to fixed orders in GARCH). The input and forget gates of LSTM allow the network to control the extent of long-memory depending on features of the time-series. Multi-step ahead forecasting is an iterative process for ARMA-GARCH and forecast errors may compound. LSTM is able to predict multi-step ahead directly. In Section 5.4.1, we apply our framework to forecast forward 1-month U.S. stock returns using daily returns. Nonetheless, we do not directly compare against ARMA-GARCH models for two reasons. First, in this work, we are focused on advancing uncertainty quantification methodologies for neural networks. We argue that several of our advances can be beneficial to both time-series and non-time-series datasets (as demonstrated in Appendix A7.1). Second, we lean on the plethora of literature in comparing LSTM to ARMA-variants (e.g., Siami-Namini et al., 2018) and ARCH-variants (e.g., Liu et al., 2019).

For ease of comparison, we outline the differences of our method to Ensemble (Lakshminarayanan et al., 2017) and Evidential (Amini et al., 2020) in Table 5.1.

TABLE 5.1: A comparison of Combined to Deep Ensemble and Deep Evidential regressions. *Output layer* refers to the structure of output layer(s) of the network that outputs the parameters of the likelihood function.

Method	**Ensemble**	**Evidential**	**Combined**
Prior	None	NIG	Gamma
Ensemble	Yes	No	Yes
Likelihood	Gaussian	Student's t	Student's t
Output layer	Single layer μ, σ^2	Single layer $\gamma, \nu, \alpha, \beta$	Multi-layer $\gamma, \sigma^2, \alpha, \beta$

5.4 Experiments

Our proposed framework is primarily focused on advancing uncertainty quantification in time-series exhibiting volatility clustering. In this chapter, we detail experiment results in our motivating application — time-series forecasting and uncertainty quantification on cryptocurrency and U.S. equities time-series datasets, to illustrate the benefits of our proposed method. Nonetheless, SMD parameterisation, modelling distribution parameters using subnetworks and ensemble predictions can also be applied to general applications of prediction uncertainty quantification. In Appendix A7.1, we also compare our method to Ensemble and Evidential using the UCI benchmark dataset. This is intended to provide readers with a direct comparison to the results published in Lakshminarayanan et al. (2017) and Amini et al. (2020), demonstrating the benefits of our proposed improvements in non-time-series datasets.

5.4.1 Uncertainty quantification in cryptocurrency time-series forecasting

In this section, we will first describe the cryptocurrency dataset, then present empirical results on cryptocurrencies. Further confirmatory experiments on more conventional financial time-series (U.S. equities) is presented in Section 5.4.2. The same neural network architectures are used in the two datasets, with hyperparameters tuned independently. The hyperparameters used are recorded in Appendix A5.

Our cryptocurrency dataset consists of hourly returns downloaded from Binance over July 2018 to December 2021, for 10 of the most liquid, non-*stablecoin*[8] cryptocurrencies. Tickers for these cryptocurrencies are BTC, ETH, BNB, NEO, LTC, ADA, XRP, EOS, TRX and ETC, denominated in USDT[9]. Following Chapter 3 and 4, we use IC (Equation (3.2); cross-sectionally computed for each t for all 10 cryptocurrencies, then averaged over time) as a measure of predictive accuracy, in addition to RMSE and NLL. Data from July 2018 to June 2019 are used for hyperparameter tuning, chronologically split into $70\,\%$ training and $30\,\%$ validation. Data from July 2019 to December 2021 are used for out-of-sample testing. Networks are trained every 30 days using an expanding window of data from July 2018, which is preferred over a rolling window approach used in Chapter 4 due to the small sample size of the cryptocurrency dataset. Each input sequence consists of 10 days of hourly returns r and squared returns $\log(r^2)$ (i.e., each input sequence is a matrix with dimensions 240×2), and are used to predict forward one hour return (i.e., units of analysis and observation are both hourly). Network topology consists of LSTM layers, followed by fully connected layers with ReLU activation and the corresponding output layers of Ensemble and Evidential. For Combined, we use four subnetworks as illustrated in Figure 5.1. As discussed in Section 5.1, we consider uncertainty quantification in cryptocurrencies to be especially challenging due to their high volatility. Note that in this section and Section 5.4.2, "forecast uncertainty" and "uncertainty forecast" refer to estimated predictive uncertainty (i.e., sum of epistemic and aleatoric uncertainties) for simplicity.

At this point, it is useful to remind readers that prior literature have found both datasets to exhibit time-varying variance (e.g., Cont, 2001; Hafner, 2018), which is also visible in Figure 5.2. We start with the main empirical results on cryptocurrency time-series forecasting, recorded in Table 5.2. We observe that Combined has the highest average IC, lowest RMSE and NLL in the cryptocurrency dataset. This indicates that Combined has higher cross-sectional predictive efficacy (as measured by IC) and is able to better forecast uncertainty of

[8]Stablecoins are cryptocurrencies that are pegged to real world assets (e.g., U.S. Dollar). As such, they exhibit lower volatility than other non-pegged cryptocurrencies.

[9]*Tether* (USDT) is a stablecoin that is pegged to USD. It has the highest market capitalisation amongst the USD-linked stablecoins (Lipton, 2021).

the time-series prediction. Evidential has better (higher) IC and (lower) RMSE but worse (higher) NLL than Ensemble.

Next, Figure 5.2, compares predicted uncertainty and actual prediction error of the three methods to actual volatility of Bitcoin (BTC/USDT), the cryptocurrency with the highest market capitalisation, and Cardano Ada (ADA/USDT), a cryptocurrency with relatively smaller market capitalisation and higher volatility. Volatility forecasts are often compared with observed volatility (typically computed over a look back window) to evaluate forecast performance. However, the true instantaneous volatility of an asset (i.e., σ^2 in Equation (5.1)) is unobservable (Ge et al., 2022). Thus, in the top row of Figure 5.2, we use the standard deviation of hourly returns computed over each day as a proxy for σ^2. In rows 2–4, for Bitcoin, we aggregate hourly forecasts to daily data points by computing the daily RMSE of return forecasts $\sqrt{\frac{1}{24}\sum_{k=0}^{23}(y_{t-k}-\hat{y}_{t-k})^2}$ (denoted $\sqrt{(y-\hat{y})^2}$) computed from hourly return forecasts, and the daily root mean predictive uncertainty $\sqrt{\frac{1}{24}\sum_{k=0}^{23}\mathrm{Var}(\hat{y}_{t-k})}$ (denoted $\sqrt{\mathrm{Var}(\hat{y})}$), for each $t = 24, 48, 72, \ldots, T$ (note that t for cryptocurrency is in hourly units). Comparing the top row of Figure 5.2 to the root return forecast error of row 2-4 (blue line), we observe that forecast error spikes when volatility of the asset spikes. This is expected, as the spike in volatility leads to large forecast errors. Comparing the bottom three rows of Figure 5.2, which correspond to Combined, Ensemble and Evidential, respectively. We observe that Combined's predicted uncertainty of $\hat{\mu}$ tracks actual forecast error much more closely than Evidential and Ensemble. This appears to be especially true during periods of elevated volatility (e.g., during March 2020), which are important to investors. Overestimation

TABLE 5.2: Comparing Ensemble, Evidential and Combined on average IC, RMSE and NLL for cryptocurrencies time-series forecasts. Average result and standard deviation over 10 trials for each method. Best method for each dataset is highlighted in **bold**.

Metric	Ensemble	Evidential	Combined
IC (%)	2.78 ± 1.09	3.94 ± 1.84	$\mathbf{9.87 \pm 3.17}$
RMSE (%)	0.874 ± 0.022	0.874 ± 0.003	$\mathbf{0.867 \pm 0.001}$
NLL	-3.74 ± 0.10	-3.24 ± 0.02	$\mathbf{-4.14 \pm 0.01}$

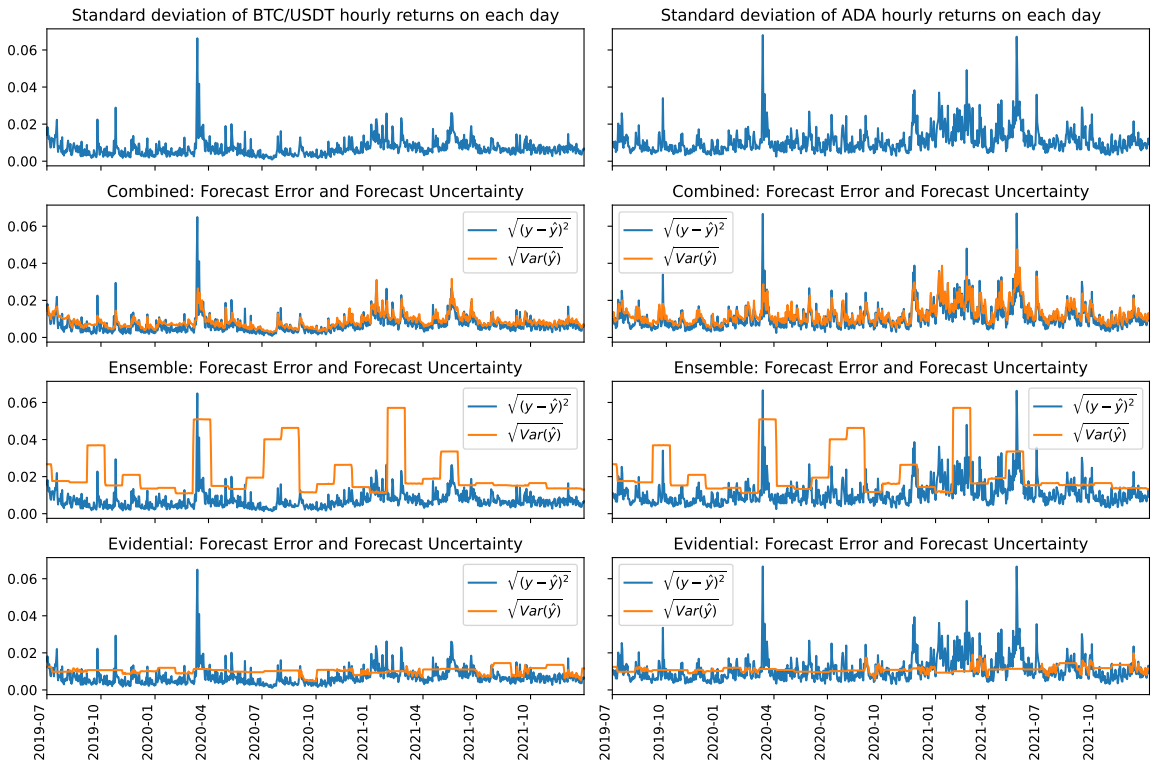


FIGURE 5.2: **First row**: Standard deviation of hourly return of BTC/USDT and ADA/USDT on each day. **Second–fourth rows**: Actual prediction error and predicted uncertainty $\mathrm{Var}(\hat{y})$ of Combined, Ensemble and Evidential for BTC/USDT (left column) and ADA/USDT (right column), respectively. Square root of the average squared error and uncertainty over each day shown.

of predictive uncertainty is severe for Ensemble in Bitcoin, where predictive uncertainty can sometimes be significantly higher than observed forecast error.

Note that the "block-like" appearances of uncertainty forecasts of both Ensemble and Evidential are due to periodic training (monthly for cryptocurrencies and yearly for U.S. equities) and the failure to generalise the prevailing volatility environment. During training, the optimiser updates network weights $\boldsymbol{W}$ and bias $\boldsymbol{b}$ (which is analogous to the intercept in linear models). When the network fails to generalise, it minimises the loss function by updating the bias rather than the weights. Thus, outputting the same constant that do not vary with the input, until the network is re-trained in the following month. This produces the block-like appearances of Ensemble and Evidential, and is indicative of the network setup (e.g., no separate modelling of

hyperparameters) being unsuitable to this class of problems. Lastly, Evidential underestimates forecast error during heightened volatility (e.g., March 2020) and overestimates forecast error under periods of low volatility (e.g., July 2020). In Appendix A9, we investigate the addition of separate modelling of distribution hyperparameters for Evidential, and conclude that both squared returns and separate hyperparameter modelling are required to achieve uncertainty forecasts that closely tracks time-varying volatility. We observe similar visual characteristics in the predicted uncertainty of other cryptocurrencies for all three methods.

5.4.2 Further results on U.S. equities

In this section, we provide empirical results of a further study on a conventional financial time-series dataset, quantifying forecast uncertainty in U.S. equities. Mimicking the S&P 500 index universe, the dataset consists of daily returns downloaded from CRSP over 1984 to 2020, for the 500 largest stocks[10] listed on NASDAQ, NYSE and NYSE American. Data from 1984 to 1993 are used for hyperparameter tuning, while 1994 to 2020 are used for out-of-sample testing. The network is refitted every January using a rolling 10-year window. We retain the same hyperparameter tuning setup to the cryptocurrency dataset, each input sequence consists of 240 trading days (approximately one-year) of daily returns r and squared returns $\log(r^2)$ (rather than 250 trading days in Chapter 4), forecasting forward 20-day (approximately one-month) return and its uncertainty. Given that 240 days cover $95\,\%$ of the 252 trading days per year, we do not expect this choice to have a material impact on the experiment results when compared to Chapter 4. Note that the unit of analysis is monthly and unit of observation is daily. One-month is a popular forecast horizon for U.S. equities in literature (e.g., Gu et al., 2020 and is used in Chapter 3 and 4), which motivated our choice of forecast horizon. The basic setup is similar to the financial time-series forecasting experiment in Chapter 4. The same models as the cryptocurrency experiment are used with separate hyperparameter tuning. Further details on hyperparameters are provided in Appendix A5.

Table 5.3 records the empirical results on U.S. equities. Again, we observe that Combined has the highest IC, and lowest RMSE and NLL out of the three methods. This demonstrates

[10]The list of stocks is refreshed every June, keeping the same stocks until the next rebalance.

TABLE 5.3: Comparing Ensemble, Evidential and Combined on average IC, RMSE and NLL for U.S. equities. Average result and standard deviation over 10 trials for each method. Best method for each dataset is highlighted in **bold**.

Metric	Ensemble	Evidential	Combined
IC (%)	0.40 ± 0.66	0.09 ± 0.93	$\mathbf{1.22 \pm 0.65}$
RMSE (%)	9.426 ± 0.044	9.433 ± 0.033	$\mathbf{9.379 \pm 0.020}$
NLL	-1.65 ± 0.17	-0.82 ± 0.03	$\mathbf{-1.71 \pm 0.01}$

the usefulness of Combined in quantifying forecast uncertainty in both time-series with extreme volatility (e.g., cryptocurrencies) and in conventional financial time-series. IC in U.S. equities are materially lower for all three methods compared to the cryptocurrency dataset. We hypothesise that this is due to both the difference in forecast horizon and maturity of the U.S. market.

Figure 5.3 compares the predicted uncertainty and actual prediction error of the three methods to actual volatility of Chevron Corp., a major U.S. oil producer, and IBM, a major U.S. technology company. As the unit of analysis is monthly, we plot the absolute error between observed monthly returns and predicted returns (denoted $|y - \hat{y}|$) in the top row of Figure 5.3, and square-root of forecast uncertainty (denoted $\sqrt{\text{Var}(\hat{y})}$) in rows 2–4 for Combined, Ensemble and Evidential, respectively. We observe similar results as the cryptocurrency experiment in the bottom three rows of Figure 5.3. Predicted uncertainty of Combined is observed to track actual forecast error more closely than Ensemble, especially during the three market crashes — the Dot-com bubble (2000–01), U.S. recession over 2008–09 and the 2020 pandemic. For Chevron, we observe an additional spike of volatility during the 2015 oil shock. Evidential produced uncertainty forecasts that are visually similar to Combined, but block-like features can still be seen in 1999 and 2012. Ensemble's predicted uncertainty for IBM jumped cover 2000–01, coinciding with a period of elevated volatility for the stock. In Figure 5.3, Ensemble exhibited less block-like appearance than in Figure 5.2. This indicates that Ensemble achieved better generalisation performance on the U.S. equities dataset than on the cryptocurrency dataset. However, Ensemble's predicted uncertainty for Chevron saw the same block-like jump which did not coincide with higher volatility of the stock. We hypothesise that generalisation for Ensemble is still problematic on the U.S. equities dataset

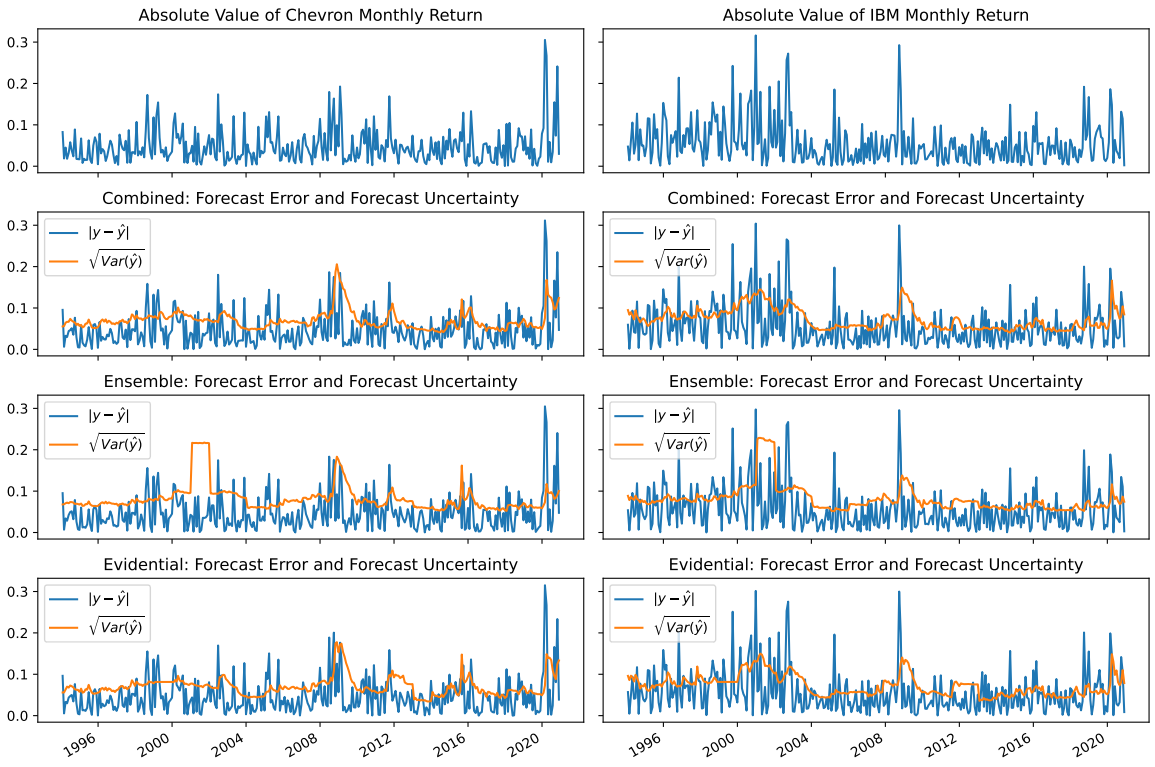


FIGURE 5.3: **First row**: Absolute monthly returns of Chevron (left) and IBM (right). **Second–fourth rows**: Actual prediction error and predicted uncertainty $\mathrm{Var}(\hat{y})$ of Combined, Ensemble and Evidential for Chevron and IBM, respectively. Square root of the monthly forecast error and forecast uncertainty shown.

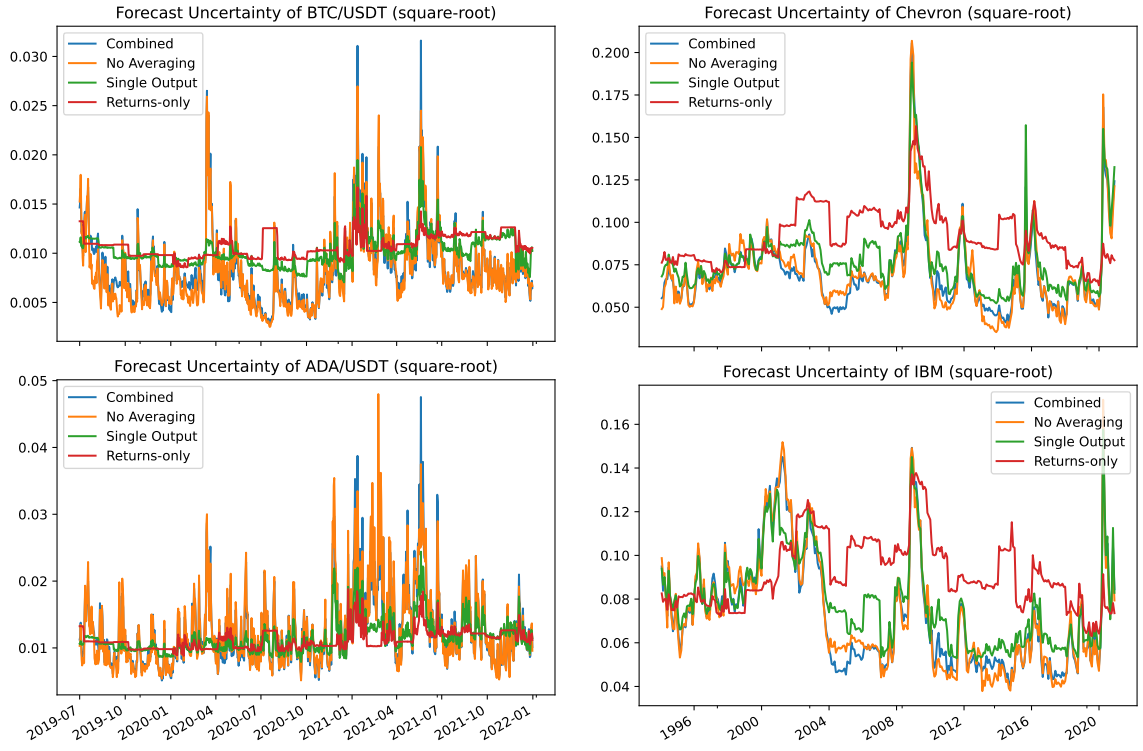
and that Ensemble failed to generalise the impact of heightened volatility environment on different stocks.

5.4.3 Ablation study

Next, we test the effects of removing each of the following for Combined: 1) model averaging; 2) single output layer for all distribution parameters (same as Evidential); 3) using return time-series only (i.e., no squared returns). The results are recorded in Table 5.4 and in Figure 5.4. As discussed in Section 5.3.2, model averaging (Equation (5.17)) will lead to higher predictive uncertainty estimates. Comparing results in Table 5.4 to the main results in Table 5.2 and Table 5.3, we observe that model averaging has a large negative impact on IC and NLL. IC

TABLE 5.4: **Ablation studies**: In each column, we remove model averaging (*No Averaging*), separate modelling of distribution parameters (*Single Output*) and using return time-series only (*Returns-only*) from Combined for cryptocurrencies (left) and U.S. equities (right), respectively. Average result and standard deviation over 10 trials are reported for each method. Note that cryptocurrency returns are hourly and U.S. stock returns are monthly.

	Cryptocurrency			U.S. equities		
Metric	No Averaging	Single Output	Returns Only	No Averaging	Single Output	Returns Only
IC (%)	4.48 ± 2.80	8.23 ± 2.91	10.46 ± 2.04	0.92 ± 0.65	1.87 ± 1.06	1.21 ± 0.73
RMSE (%)	0.868 ± 0.001	0.872 ± 0.002	0.866 ± 0.002	9.392 ± 0.020	9.398 ± 0.029	9.384 ± 0.046
NLL	-3.35 ± 0.01	-4.04 ± 0.02	-3.95 ± 0.02	-0.88 ± 0.01	-1.63 ± 0.04	-1.34 ± 0.04



(a) Uncertainty of BTC/USDT and ADA/USDT

(b) Uncertainty of Chevron and IBM

FIGURE 5.4: Square-root of predicted uncertainty $\mathrm{Var}(\hat{y})$ of Combined, without model averaging (*No Averaging*), single output layer (*Single Output*) and using returns only (*Returns-only*).

is $55\,\%$ and $25\,\%$ lower for cryptocurrencies and U.S. equities, respectively. While NLL is higher by 0.8 in both cases (lower is better), indicating a worse overall fit. However, it

does not appear to impede the network's ability to model time-series forecast uncertainty (as observed in Figure 5.4). Moreover, comparing Combined (with model averaging) to No Averaging (without model averaging) in Figure 5.4, we observe very similar estimated predictive uncertainties with and without model averaging (as the orange and blue lines track each other closely). This indicates a favorable trade-off between significantly improved forecast performance and practically the same predictive uncertainty estimates. Using a single output layer for all distribution parameters leads to marginally worse NLL. IC is lower in cryptocurrencies but marginally higher in U.S. equities. While using returns only leads to marginally higher IC but marginally lower on NLL in cryptocurrency, and lower IC and NLL in U.S. equities. From Figure 5.4, the block-like appearances indicate that both using single output layer and using returns only result in the network failing to closely track time-varying variance of the DGP. This suggests that both squared returns and separate modelling of distribution parameters are required to model time-varying forecast uncertainty.

5.5 Conclusions

Our motivating application of portfolio selection depends on both forecasts and forecast uncertainties. This is a challenging problem due to both the low signal-to-noise ratio in financial markets (Gu et al., 2020) and the presence of volatility clustering. To this end, we present a method for the simultaneous forecasting asset returns and modelling of forecast uncertainty in presence of volatility clustering. Our proposed method extends and simplifies the work of Lakshminarayanan et al. (2017) and Amini et al. (2020). We propose to use a SMD (which uses a Gamma prior for scale uncertainty ν) as a simpler alternative to a NIG prior, in which a Normal prior is placed on μ and an Inverse-Gamma prior on σ^2). Parameters of SMD are modelled using separate subnetworks. Together with ensembling and the use of second order of returns as inputs, we show that our proposed method can successfully model time-varying variance of the DGP, while providing superior forecasting performance than two state-of-the-art neural network uncertainty quantification methods — Evidential and Ensemble. This is illustrated through the successful quantification of forecast uncertainty of two financial time-series datasets: cryptocurrency and U.S. equities. Our proposed SMD

formulation offers an avenue to resolve some of the criticisms of Meinert et al. (2022) and Bengs et al. (2023). In particular, our SMD parameterisation has three effective parameters and thus does not have any unresolved degrees of freedom. We can set $\alpha = \beta$, which leads to a marginal t-distribution where the three distributional parameters $(\gamma, \sigma^2, \alpha)$ relate directly to the location, scale and shape of the t-distribution, without the need of a hierarchical model. In this formulation, epistemic uncertainty is assumed to be the difference between the predictive (t-distributed) and aleatoric (Normal-distributed) uncertainties. This assumption provides for a simpler model but lacks the granular attribution between aleatoric and epistemic uncertainties afforded by the NIG prior in Evidential. However, as Meinert et al. (2022) has pointed out, the granular control comes at the cost of an unresolved degree of freedom. Thus, users are encouraged to weigh the trade-offs in choosing a method to deploy. This also makes for a potential future research direction. We show empirically that our method is able to accurately predict forecast errors, similar to the success Evidential demonstrated in other real world applications (e.g., see Liu et al., 2021; Soleimany et al., 2021; Cai et al., 2021; Singh et al., 2022; Li and Liu, 2022). From a finance application perspective, forecast uncertainty can be used to size bets, or as advanced warning to protect the portfolio from downside risk. For example, if forecast uncertainty reaches a certain threshold, an investor could purchase portfolio insurance (e.g., put options) or liquidate positions to reduce risk. The ability to attribute epistemic and aleatoric uncertainties may also allow for more advanced portfolio optimisation techniques to be developed in future research (e.g., place different risk aversions on the two sources of uncertainties). Lastly, uncertainty quantification in time-series applications is a relatively under-explored area of literature. We believe this work can lead to further advancements of uncertainty quantification in complex time-series.

CHAPTER 6

Conclusion

Neural networks have made tremendous strides across many domains over the past decade, from mastering the game of Go[1], self-driving cars, medical diagnosis, to the personal assistant in smart phones carried by millions of people worldwide. Financial markets have proved to be a challenging problem for econometricians and statisticians. As discussed in Section 1.5, financial markets can suffer from endogenous and exogenous shocks, the distribution of asset returns has heavy tails, signal-to-noise ratio is low and, most inconveniently, the data generation process changes over time. Given the success neural networks have achieved in other domains, this begs the question — can neural networks advance the state-of-the-art in financial market applications? At the conclusion of this thesis, we believe we are a step closer to the opening of the floodgates but with much more still to be done. In the rest of this chapter, we will first outline our contributions to literature, then discuss future research directions on this topic.

6.1 Contributions to machine learning in portfolio management

In this thesis, we have provided an overview of the mechanics of a quantitative investment process, outlined the relevant finance theory that underpins stock return predictability (or the lack of), discussed a number of challenges in applying conventional statistical and machine learning tools in financial markets, identified several potential ways machine learning can be

[1] *Go* is an abstract board game and is considered as the most challenging of classic games for artificial intelligence (Silver et al., 2016).

used to improve the quantitative investment process, and proposed three distinct advances to deep learning that covers three related applications in financial market predictions.

In Chapter 3, we address the time-varying DGP problem in fnancial markets in a cross-sectional prediction application of neural networks. We proposed the Online Early Stopping algorithm for training neural networks online. Online training of a neural network is an online optimisation problem. In classical online optimisation literature, online optimisation algorithms are analysed in terms of *regret* — a measure of performance loss compared to a theoretical (but unattainable) optimum. We provide a worst-case performance bound that conforms with the notion of regret in a non-convex optimisation context, by showing that the OES algorithm can achieve tracking performance no worse than a function of the variance of the DGP. This provides an intuitive interpretation of the worst-case performance of the algorithm, where its ability to track a moving DGP is bounded by the variance of the DGP (i.e., the higher the time variability of the DGP, the more difficult it is to track and the higher the expected loss). We compare OES to a static neural network[2] and the DTS-SGD, a state-of-the-art non-convex online optimisation algorithm, in simulated data and showed superior performance in tracking a time-varying DGP. We highlight the usefulness of OES to practitioners by comparing OES to the static neural network used in Gu et al. (2020), demonstrating competitive performance and the ability to track changes in financial markets over time. In particular, we show that the OES-trained network reacts quicker (relative to the static network) to market downturns and recoveries, such as the global financial crisis and the 2015 oil shock. Finally, we show that the ensemble prediction of the static network and OES-trained network delivered the best prediction performance. Thus, we argue that OES can be a useful tool for practitioners in predicting cross-sectional stock returns.

In Chapter 4, we address the low signal-to-noise problem in financial markets in a time-series prediction application of neural networks. We propose the Supervised Temporal Autoencoder architecture, using a supervised autoencoder to regularise a temporal convolutional network. We argue that due to the low signal-to-noise in financial markets, machine learning should focus on "robust" learning as opposed to "deep" learning. The proposed supervised autoencoder

[2]In this context, a static neural network refers to a network that is trained using all available data and does not vary with time.

imposes a non-parametric functional form on the latent representation of the input sequence. We argue that this is more flexible than a fixed parametric functional form (e.g., a linear time trend). The reconstruction task provides interpretability, allowing users to inspect the smoothed reconstructed sequence to visualise the features retained by the neural network. We provide a template for financial time-series forecasting directly using price series, alleviating the need for handcrafted features. In the application test, we compare STAE to momentum, a prominent stock return predictor documented in finance literature, and showed material improvement in predictive performance — a finding that is economically meaningful to practitioners. We establish a benchmark of sequential neural network architectures in financial time-series forecasting, demonstrating superior predictive performance of STAE over TCN, LSTM and transformers. We show that the addition of the auxiliary task, even at a small weight, is beneficial to the prediction task. We document declining predictive performance of momentum, an asset pricing anomaly in finance literature, and predictions of neural networks. We conjecture that markets are becoming increasingly efficient and that information content of stock prices has decreased over time.

In Chapter 5, we advance the state-of-the-art in incorporating *risk*, in the form of predictive uncertainty, into neural network forecasts. Risk forecasting is of paramount importance in portfolio optimisation and can influence optimal bet sizes. We combine and extend two state-of-the-art methods, Ensemble (Lakshminarayanan et al., 2017) and Evidential (Amini et al., 2020), into a unified framework for quantifying time-series forecast uncertainty of neural networks. The unified framework consists of four improvements. Firstly, we propose to use the SMD instead of the NIG prior used in Evidential, arguing that SMD is simpler and offers superior numerical properties than the NIG prior, which would allow a first-order optimiser (such as SGD) to more easily traverse the loss landscape[3]. Secondly, we argue that in cases where the input has complex relations with the distribution hyperparameters (such as in financial time-series forecasting), it is beneficial to afford the neural network of the flexibility to non-linearly transform the common latent representation of the input sequence generated by the hidden layers (i.e., convolutional or recurrent layers). This is in contrary

[3]Loss landscape refers to the hyperplane spanned by network parameters (Li et al., 2018). A smooth and convex loss landscape can be easily traversed by a first-order optimiser. Conversely, a highly non-convex loss landscape with saddle points and many local minima will be difficult to traverse.

to the output layers used in Ensemble and Evidential, which compute hyperparameters of the distribution as linear combinations of the latent representation. Thirdly, we propose to incorporate ensembling, which was shown to significantly improve forecast accuracy in Chapter 4. Lastly, we propose to incorporate the second moment of returns to inform the network of the prevailing volatility environment, which will directly affect forecast uncertainty. We provide evidence of the benefits of the framework and each of the four improvements using the UCI benchmark datasets, and in cryptocurrencies and U.S. equities forecasts. In particular, using the UCI dataset and an identical network topology, we show that SMD delivers superior uncertainty quantification performance compared to the NIG prior. We believe forecast uncertainty will be a useful input into the portfolio optimisation process, such as for determining optimal bet size (high forecast uncertainty attracts a lower limit) or for scaling the risk model (Equation (1.4)).

6.2 Future research

As discussed in Chapter 1, financial markets represent one of the most challenging areas for the application of machine learning. In this section, we detail several potential advances of machine learning in future research.

Financial markets are endogenous. That is, one's own trading leaves a trail of footprints on asset prices (by incurring market impact and perturbing the share price). The same patterns may also be discovered by other investors which leave the same footprints. Financial markets are also impacted by exogenous shocks such as pandemics, wars and recessions. Thus, time-varying models have an important role in portfolio management. In Chapter 3, we have introduced OES and shown that, in theory, it offers superior predictive performance over a stationary model. We have not investigated realistic performance in a portfolio setting, after accounting for transaction costs and portfolio constraints (e.g., limits on how much the portfolio can bet on any single stock or industry). It is conceivable that actual realisable benefits of a time-varying model are concave with respect to the time-variability of the model due to higher trading. A model with moderate time-variability is likely better than a stationary

model after transaction costs. However, a highly time-varying model may not be better than a moderately time-varying model if transaction costs outstrip further improvements in tracking the time-varying DGP more closely. Thus, future research on this topic can investigate portfolio-level impacts of time-varying models and consider regularisations of OES, such as smoothing of regret in Hazan et al. (2017).

Efficient trading has been discussed in Section 1.6.4. Whilst not addressed in this thesis, this is a worthy topic within the broader domain of machine learning in portfolio management and is an essential component of autonomous trading systems. Stocks exhibit various intraday patterns, such as U-shaped volume distribution (higher at the beginning and end) throughout the trading day (Wood et al., 1985; Jain and Joh, 1988; McInish and Wood, 1992; Eaves and Williams, 2010), which is also associated with similar U-shaped intraday volatility (Lockwood and Linn, 1990; Eaves and Williams, 2010). Volume and volatility patterns form useful inputs to any model that aims to minimise market impact by predicting expected volume during the day. Future work can leverage recent advances in deep reinforcement learning[4] (François-Lavet et al., 2018), combining with new features to extend the work by Nevmyvaka et al. (2006) on using reinforcement learning for optimal trade execution. Another potential direction is to extend the work by Webber (2017), in incorporating concept drifts (Gama et al., 2014) into deep reinforcement learning to address time-varying financial markets (similar to our work in Chapter 3).

Extracting information from text has been discussed in Section 1.6.4. There is a large swathe of text information about companies, such as management's discussion of business performance in the annual report, may contain useful information for predicting future return. A significant portion of a financial analyst's job is to transform textual information about a company, such as the company's strategy and the competitive landscape, into future revenue and earnings expectations (Damodaran, 2006). For example, a biotechnology company developing a life-saving drug may see significant revenue in the future but is currently loss-making. Recently,

[4]Reinforcement learning is the task of learning a policy (i.e., sequence of actions) in an environment in order to maximise cumulative rewards (Bishop, 2006; Murphy, 2012; François-Lavet et al., 2018). This requires estimating future expected reward for each action. Deep reinforcement learning is to use neural networks to estimate future reward.

Araci (2019) used the BERT model (which was trained using the Wikipedia corpus, Devlin et al., 2019) and retrained the final layers using financial news articles to learn a finance-specific language model. I argue that the resultant model understands the grammar used in financial text, but is intrinsically devoid of understanding of the context. In the biotechnology firm example, an article may discuss the drug that the company is developing, but not the financial implications. Such second order effect is inferred from the context. Advances in this domain may combine natural language progressing and concept learning (Mitchell, 1997) in the context of financial markets.

In Chapter 4, we show that it is possible to learn predictive patterns directly from the share price time-series. Potential improvements to both the application of financial time-series forecasting and the method of learning in noisy environments are discussed in Section 4.4. A natural extension of this work is to combine cross-sectional forecasting and time-series forecasting, where the neural network is provided with time-series of all features relating to the company, such as stock prices, company financials and social media sentiment. Learning weak signals from such a large and diverse feature set will pose a significant challenge. However, I am convinced that if the financial industry were to advance towards highly tailored, stock-specific models, the advances will be reminiscent of the model described above. Such model can leverage methods of supervised autoencoding as described in Chapter 4, where the autoencoder performs dimensionality reduction which may assists with processing from a large feature set.

In Chapter 5, we have proposed a framework for quantifying uncertainty in financial time-series predictions. We suggest that forecast uncertainty can be used to determine bet sizes and serves as an input into the portfolio construction process. The quantified uncertainty is a scalar value that is specific to the stock. However, uncertainty may be correlated between stocks. Thus, a natural extension is to produce variance-covariance-style uncertainty that captures uncertainty covariance between a cohort of stocks. The uncertainty covariance matrix can then substitute or supplement the conventional variance-covariance matrix of asset returns used in mean-variance optimisation, as the latter neglects parameter uncertainty in return forecasts. Variance-covariance matrices are subject to the *curse of dimensionality*. For

example, consider the Russell 3000 index used in Chapter 4. Estimating a variance-covariance matrix for this universe involves estimating $3000 \times 3000 = 9\,\text{million}$ values. As discussed in Section 2.5.2, one of the advantages of CNN over fully connected neural network in image recognition problems is *parameter sharing*, which greatly reduces the number of parameters required by representing common patterns with a small number of parameters organised in a kernel. This property of convolution layers may offer a viable avenue to solve the curse of dimensionality problem in uncertainty covariance estimation.

The three advances introduced in this thesis relates to: online learning in a cross-sectional prediction context (Chapter 3), noise-robust learning in a time-series prediction context (Chapter 4), and forecast uncertainty quantification (Chapter 5). All three topics play important roles in quantitative investing. We argue that the three advances can be combined into a unified framework to simultaneously forecast returns, provide predictive uncertainty, and adapt to changes in the DGP of financial markets. We propose the following neural network architecture which can be evaluated in future work. The network is comprised of eight subnetworks:

- Subnetwork 1: Fully connected layers to process cross-sectional firm features (as per Chapter 3).
- Subnetwork 2: An encoder of LSTM layers to process daily stock returns (as per Chapter 5).
- Subnetwork 3: A decoder with fully connected layers which reconstruct all firm features using the latent representation outputted by Subnetwork 1.
- Subnetwork 4: A decoder of LSTM layers which reconstruct the daily return sequence using the latent representation outputted by Subnetwork 2.
- Subnetwork 5: Fully connected layers which combine the output of Subnetwork 1 and 2.
- Subnetwork 6-8: Outputs parameters γ, σ^2 and α of the SMD to simultaneously estimate both returns and predictive uncertainty.

The loss function is the NLL of the marginal t-distribution of SMD, plus reconstruction error of both firm features (output of Subnetwork 3) and time-series of returns (output of Subnetwork 4). To adapt to time-varying DGP, the network can be trained using OES. However, we envisage three potential challenges with this approach. First, as the network is quite large, one may encounter difficulties in training the entire network simultaneously. To solve this, one may employ *transfer learning* in training the two autoencoders for firm features (Subnetwork 1 and 3) and return series (Subnetwork 2 and 4). Transfer learning has been successfully applied in natural language processing, where a language model (typically an encoder-decoder) is trained on a large corpus of text to predict the next word, given the preceding words. The pre-trained language model is then fine tuned on downstream tasks such as sentiment analysis and question-answering (Ruder et al., 2019; Han et al., 2021). In a similar vein, we can first pre-train each encoder-decoder pair (Subnetwork 1 and 3, and Subnetwork 2 and 4) on encoding and decoding firm features and return series, respectively. The pre-trained encoder-decoder pairs can then be used in the final amalgamated network to perform return prediction. Second, the OES algorithm involves training on a the $t-2$ cross-section and validating on the $t-1$ cross-section. For such a large network, one may find that one cross-section contains insufficient data to train the network. To solve this, one may expand the number of periods used to train the network. However, we caution that expanding the look back window will lead to the algorithm to fit the average DGP in the look back window. Thus, losing its ability to closing track the time-varying DGP. Third, the scales of the three components of the loss function are different. Thus, care must be taken during hyperparameter search to determine the optimal weight given reconstructing firm features and return series. The proposed network architecture outputs both the return forecast and predictive uncertainty, which can then be used in downstream portfolio optimisation tasks.

Finally, in this thesis, we have proposed three advances that address various subtopics of applying deep learning to portfolio management, with much more still to be done. Deep learning has contributed to the advances of numerous fields of science. One particular advancement is DeepMind's *AlphaFold*[5], a machine learning system that can predict the

[5]DeepMind is a subsidiary of Alphabet Inc. that focuses on machine learning research. https://www.deepmind.com/research/highlighted-research/alphafold.

structure of over 200 million proteins and promises to speed up drug development. As a trained bioinformatician, I find this development exciting for the field of medical research and sobering for the finance industry. The main impediments to leaps in applying deep learning to financial markets are well discussed in Section 1.5. We, as finance practitioners, can dream that one day deep learning will shine some light on this dark corner of social science.

Bibliography

Balaji Lakshminarayanan, AlexanderPritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In Guyon et al. (2017), pages 6405–6416. ISBN 9781510860964.

Alexander Amini, Wilko Schwarting, Ava Soleimany, and Daniela Rus. Deep evidential regression. In Larochelle et al. (2020), pages 14927–14937.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105, Lake Tahoe, NV, USA, 2012. Curran Associates, Inc.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Bengio and LeCun (2015). URL http://arxiv.org/abs/1409.1556.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In Grauman et al. (2015), pages 1–9. doi: 10.1109/CVPR.2015.7298594.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In Grauman et al. (2015), pages 815–823. doi: 10.1109/CVPR.2015.7298682.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Lourdes Agapito, Tamara Berg, Jana Kosecka, and Lihi Zelnik-Manor, editors, *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR 2016, pages 770–778, Las Vegas, NV, USA, 2016. IEEE.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech*

and Signal Processing, ICASSP 2013, pages 6645–6649. IEEE, 2013. doi: 10.1109/ICASSP.2013.6638947.

Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, ICML'13. JMLR.org, 2013.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML'08, pages 160–167. ACM, 2008. ISBN 9781605582054. doi: 10.1145/1390156.1390177.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In Ghahramani et al. (2014), pages 3104–3112.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. ISSN 1476-4687.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. In *arXiv*, 2016.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alexander Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *arXiv*, 2016. URL https://arxiv.org/abs/1609.03499.

Eunhee Kang, Junhong Min, and Jong Chul Ye. Wavenet: a deep convolutional neural network using directional wavelets for low-dose x-ray ct reconstruction. *Medical Physics*, 44:360–375, 10 2017. doi: 10.1002/mp.12344.

Othmane Mounjid and Charles-Albert Lehalle. Improving reinforcement learning algorithms: towards optimal learning rate policies. In *arXiv*, 2021.

Jeremy D. Turiel and Tomaso Aste. Peer-to-peer loan acceptance and default prediction with artificial intelligence. *Royal Society Open Science*, 7(6):191649, 2020. doi: 10.1098/rsos.191649. URL https://royalsocietypublishing.org/doi/abs/10.1098/rsos.191649.

Richard Grinold and Ronald Kahn. *Active Portfolio Management: A Quantitative Approach for Producing Superior Returns and Controlling Risk*. McGraw-Hill Education, 1999.

Nga Pham. The australian superannuation system. Technical report, Monash University, Victoria, Australia, 2019. URL https://www.monash.edu/__data/assets/pdf_file/0016/2010553/The-Australian-superannuation-system_v3.pdf.

Shihao Gu, Bryan Kelly, and Dacheng Xiu. Empirical asset pricing via machine learning. *The Review of Financial Studies*, 33(5):2223–2273, 02 2020. ISSN 0893-9454. doi: 10.1093/rfs/hhaa009.

Pamela Peterson Drake and Frank J. Fabozzi. *Financial Instruments, Markets, and Intermediaries*, chapter 2, pages 13–35. John Wiley & Sons, Inc., 2010.

Frank J. Fabozzi, Frank J. Jones, Robert R. Johnson, and Pamela P. Drake. *Fundamentals of Common Stock*, chapter 8, pages 207–227. Volume 1 of Fabozzi and Markowitz (2011), 2011a. ISBN 9781118267028.

Yakov Amihud. Illiquidity and stock returns: cross-section and time-series effects. *Journal of Financial Markets*, 5(1):31–56, 2002. ISSN 1386-4181.

Aswath Damodaran. Lecture notes in corporate finance, Feb 2022.

Eugene F. Fama. Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*, 25(2):383–417, 1970.

Harry Markowitz. Portfolio selection. *Journal of Finance*, 7(1):77–91, 1952. ISSN 00221082, 15406261.

William F. Sharpe. Capital asset prices: A theory of market equilibrium under conditions of risk. *Journal of Finance*, 19(3):425–442, 1964. ISSN 00221082, 15406261.

Michael C. Jensen. The performance of mutual funds in the period 1945-1964. *Journal of Finance*, 23(2):389–416, 1968. ISSN 00221082, 15406261. URL http://www.jstor.org/stable/2325404.

Robert Oerter. *The Theory of Almost Everything: The Standard Model, the Unsung Triumph of Modern Physics*. Plume, 1 edition, 2006.

The Nobel Foundation. The sveriges riksbank prize in economic sciences in memory of alfred nobel 1990, Oct 1990. URL https://www.nobelprize.org/prizes/economic-sciences/1990/summary/. 2022-09-01.

Rolf W. Banz. The relationship between return and market value of common stocks. *Journal of Financial Economics*, 9(1):3–18, 1981.

Dennis Stattman. Book values and stock returns. In *The Chicago MBA: A Journal of Selected Papers*, volume 4, pages 25–45, 1980.

Barr Rosenberg, Kenneth Reid, and Ronald Lanstein. Persuasive evidence of market inefficiency. *Journal of Portfolio Management*, 11(3):9–16, Spring 1985. URL http://ezproxy.lib.uts.edu.au/login?url=https://search-proquest-com.ezproxy.lib.uts.edu.au/docview/195568751?accountid=17095. Name - New York Stock Exchange; Copyright - Copyright Euromoney Institutional Investor PLC Spring 1985; Last updated - 2015-05-25.

Campbell R. Harvey, Yan Liu, and Heqing Zhu. ... and the cross-section of expected returns. *The Review of Financial Studies*, 29(1):5–68, 2016.

Eugene F. Fama and James D. MacBeth. Risk, return, and equilibrium: Empirical tests. *Journal of Political Economy*, 81(3):607–637, 1973. doi: 10.1086/260061.

Andrew Alford, Robert Jones, and Terence Lim. *Quantitative Equity Portfolio Management*, chapter 11, pages 287–306. Volume 1 of Fabozzi and Markowitz (2011), 2011. ISBN 9781118267028.

Guofu Zhou and Frank J. Fabozzi. *Factor Models*, chapter 5, pages 103–124. Volume 1 of Fabozzi and Markowitz (2011), 2011. ISBN 9781118267028.

Jeffrey Marc Wooldridge. *Introductory Econometrics: A Modern Approach*. South-Western, 4th edition, 2008. ISBN 9780324581621.

Rama Cont. Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative Finance*, 1:223–236, 2001.

Sara Salinas and Michelle Castillo. Facebook just suffered its worst day ever. *CNBC*, 2018. URL https://www.cnbc.com/2018/07/26/facebook-is-on-pace-for-its-worst-day-ever.html.

Mohamed A. El-Erian. Facebook just suffered its worst day ever. *Bloomberg*, 2021. URL https://www.bloomberg.com/opinion/articles/2021-01-30/gamestop-gme-short-squeeze-who-will-surrender-first.

Anders Johansen and Didier Sornette. Endogenous versus exogenous crashes in financial markets. In *SSRN*, 2002. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=344980.

Yahoo! Finance. Gamestop corp. (gme) stock historical prices & data – yahoo finance, Mar 2022a. URL https://au.finance.yahoo.com/quote/GME/history?p=GME. 2022-03-07.

Ruiqiang Song, Min Shu, and Wei Zhu. The 2020 global stock market crash: Endogenous or exogenous? *Physica A: Statistical Mechanics and its Applications*, 585:126425, 2022. ISSN 0378-4371. doi: https://doi.org/10.1016/j.physa.2021.126425.

Yahoo! Finance. Devon energy corporation (dvn) stock historical prices & data – yahoo finance, Mar 2022b. URL https://au.finance.yahoo.com/quote/DVN/history?p=DVN. 2022-03-07.

Guy P. Nason. Stationary and non-stationary time-series. *Statistics in Volcanology*, 1:129–142, 2006.

João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):44:1–44:37, March 2014. ISSN 0360-0300. doi: 10.1145/2523813.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Jeffrey C. Schlimmer and Richard H. Granger. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, 1986. doi: 10.1007/BF00116895.

Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996. doi: 10.1007/BF00116900.

M. Hashem Pesaran and Allan Timmermann. Predictability of stock returns: Robustness and economic significance. *Journal of Finance*, 50:1201–1228, 1995.

Peter Bossaerts and Pierre Hillion. Implementing statistical criteria to select return forecasting models: What do we learn? *Review of Financial Studies*, 12(2):405–428, 06 1999. ISSN 0893-9454. doi: 10.1093/rfs/12.2.405. URL https://doi.org/10.1093/rfs/12.2.405.

Timotheos Angelidis, Athanasios Sakkas, and Nikolaos Tessaromatis. Stock market dispersion, the business cycle and expected factor returns. *Journal of Banking & Finance*, 59: 265–279, 2015. ISSN 0378-4266. doi: https://doi.org/10.1016/j.jbankfin.2015.04.025.

R. David McLean and Jeffrey Pontiff. Does academic research destroy stock return predictability? *Journal of Finance*, 71(1):5–32, 2016. doi: 10.1111/jofi.12365.

Xi Dong, Qi Liu, Lei Lu, Bo Sun, and Hongjun Yan. Anomaly discovery and arbitrage trading. In *SSRN working paper*, 2020. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2431498.

Narasimhan Jegadeesh and Sheridan Titman. Returns to buying winners and selling losers: Implications for stock market efficiency. *Journal of Finance*, 48(1):65–91, 1993.

Clifford S. Asness, Tobias J. Moskowitz, and Lasse Heje Pedersen. Value and momentum everywhere. *Journal of Finance*, 68(3):929–985, 2013. ISSN 00221082, 15406261.

Melody Y. Huang, Randall R. Rojas, and Patrick D. Convery. Forecasting stock market movements using google trend searches. *Empirical Economics*, 59:2821–2839, 2020. ISSN 1435-8921. doi: https://doi.org/10.1007/s00181-019-01725-1.

Lily Fang and Joel Peress. Media coverage and the cross-section of stock returns. *Journal of Finance*, 64(5):2023–2052, 2009.

Alois Weigand. Machine learning in empirical asset pricing. *Financial Markets and Portfolio Management*, 33:93–104, 2019.

Marcial Messmer. Deep learning and the cross-section of expected returns. In *SSRN*, 2017. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3081555.

Masaya Abe and Hideki Nakayama. Deep learning for forecasting stock returns in the cross-section. In *arXiv*, 2018. URL https://arxiv.org/abs/1801.01777.

Baruch Lev and Anup Srivastava. Explaining the recent failure of value investing. In *SSRN*, 2019. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3442539.

Eugene F. Fama and Kenneth R. French. The cross-section of expected stock returns. *Journal of Finance*, 47(2):427–465, 1992. ISSN 00221082, 15406261. URL http://www.jstor.org/stable/2329112.

Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1996.

Jonathan Brogaard and Abalfazl Zareei. Machine learning and the stock market. In *SSRN*, 2022. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3233119.

Shihao Gu, Bryan Kelly, and Dacheng Xiu. Autoencoder asset pricing models. *Journal of Econometrics*, 222(1):429–450, 2021. ISSN 0304-4076. Annals Issue:Financial Econometrics in the Age of the Digital Economy.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2 edition, 2020.

Luyang Chen, Markus Pelger, and Jason Zhu. Deep learning in asset pricing. In *arXiv*, 2021. URL https://arxiv.org/abs/1904.00745.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Ghahramani et al. (2014), pages 2672–2680.

Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Anomaly discovery and arbitrage trading. In *arXiv*, 2021. URL https://arxiv.org/abs/2106.04560.

Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In Guyon et al. (2017), pages 6232–6240. ISBN 9781510860964.

John F. Kolen and Stefan C. Kremer. *Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies*, pages 237–243. Wiley-IEEE Press, 2001.

David Rolnick, Andreas Veit, Serge Belongie, and Nir Shavit. Deep learning is robust to massive label noise. In *arXiv*, 2018. URL https://arxiv.org/abs/1705.10694.

Milad Moradi, Kathrin Blagec, and Matthias Samwald. Deep learning models are not robust against noise in clinical text. In *arXiv*, 2021. URL https://arxiv.org/abs/2108.12242.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, January 2014. ISSN 1532-4435.

Giorgio Patrini, Alessandro Rozza, Aditya Krishna Menon, Richard Nock, and Lizhen Qu. Making deep neural networks robust to label noise: A loss correction approach. In Yanxi Liu, James M. Rehg, Camillo J. Taylor, and Ying Wu, editors, *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR 2017, pages 2233–2241, Honolulu, HI, USA, 2017. IEEE.

Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Proceedings of the 13th European Conference on Computer Vision*, ECCV 2014, pages 94–108. Springer International Publishing, 2014. ISBN 978-3-319-10599-4.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(76):2493–2537, 2011.

Omer Sezer, Ugur Gudelek, and Murat Ozbayoglu. Financial time series forecasting with deep learning : A systematic literature review: 2005—2019. *Applied Soft Computing*, 90: 106–181, 02 2020. doi: 10.1016/j.asoc.2020.106181.

Kai Chen, Yi Zhou, and Fangyan Dai. A lstm-based method for stock returns prediction: A case study of china stock market. In Maria Prandini, editor, *Proceedings of the 2015 IEEE International Conference on Big Data*, Big Data '15', pages 2823–2824, Santa Clara, CA, USA, 2015. IEEE.

Rosa Altilio, Giorgio Andreasi, and Massimo Panella. *A Classification Approach to Modeling Financial Time Series*, pages 97–106. Springer International Publishing, Cham, Switzerland, 2019.

Yves Meyer. *Wavelets and Operators*, volume 1 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1993.

Hongju Yan and Hongbing Ouyang. Financial time series prediction based on deep learning. *Wireless personal communications*, 102(2):683–700, 2017.

Zhixi Li and Vincent Tam. Combining the real-time wavelet denoising and long-short-term-memory neural network for predicting stock indexes. In *2017 IEEE Symposium Series on Computational Intelligence*, SSCI, pages 1–8, Honolulu, HI, USA, 2017. IEEE.

John L. Kelly. A new interpretation of information rate. *Bell System Technical Journal*, 35(4): 917–926, 1956.

Tim Byrnes and Tristan Barnett. Generalized framework for applying the kelly criterion to stock markets. *International Journal of Theoretical and Applied Finance*, 21(05):1–13, 2018. doi: 10.1142/S0219024918500334.

Fischer Black and Robert Litterman. Global portfolio optimization. *Financial Analysts Journal*, 48(5):28–43, Sep 1992.

John Mitros and Brian Mac Namee. On the validity of bayesian neural networks for uncertainty estimation. In *arXiv*, 2019. URL https://arxiv.org/abs/1912.01530.

Matias Quiroz, Robert Kohn, Mattias Villani, and Minh-Ngoc Tran. Speeding up MCMC by efficient data subsampling. *Journal of the American Statistical Association*, 114(526): 831–843, 2019. doi: 10.1080/01621459.2018.1448827.

Gabriele Ranco, Darko Aleksovski, Guido Caldarelli, Miha Grčar, and Igor Mozetič. The effects of twitter sentiment on stock price returns. *PloS one*, 10:1–21, 09 2015. doi: 10.1371/journal.pone.0138441.

Wataru Souma, Irena Vodenska, and Hideaki Aoyama. Enhanced news sentiment analysis using deep learning methods. *Journal of Computational Social Science*, 2:33–46, 01 2019. doi: 10.1007/s42001-019-00035-x.

Ran El-Yaniv, Amos Fiat, Richard M. Karp, and G. Turpin. Optimal search and one-way trading online algorithms. *Algorithmica*, 30(1):101–139, 05 2001. doi: 10.1007/s00453-001-0003-0.

Lili Dworkin, Michael Kearns, and Yuriy Nevmyvaka. Pursuit-evasion without regret, with an application to trading. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the*

31st International Conference on Machine Learning, volume 37, pages 1521–1529, Bejing, China, 2014. JMLR.org.

Yuriy Nevmyvaka, Yi Feng, and Michael Kearns. Reinforcement learning for optimized trade execution. In William Cohen and Andrew Moore, editors, *Proceedings of the 23rd International Conference on Machine Learning*, volume 2006 of *ACM International Conference Proceeding Series*, pages 673–680, Pittsburgh, PA, USA, 06 2006. ACM.

Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, 11(3–4):1–156, 2018. doi: 10.1561/2200000071.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359—-366, jul 1989. ISSN 0893-6080.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989. ISSN 1435-568X. doi: 10.1007/BF02551274. URL https://doi.org/10.1007/BF02551274.

Alexander Bain. *Mind and Body: The Theories of their Relation*. D. Appleton and Company, New York, NY, USA, 1873.

William James. *The Principles of Psychology*. H. Holt and Company, New York, NY, USA, 1890.

Frank Rosenblatt. The perceptron: A probalistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

Frank Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington D.C., USA, 1961.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, pages 318–362. MIT Press, Cambridge, MA, USA, 1986a. ISBN 026268053X.

Eran Malach, Gilad Yehudai, Shai Shalev-Shwartz, and Ohad Shamir. The connection between approximation, depth separation and learnability in neural networks. In Mikhail Belkin and Samory Kpotufe, editors, *Conference on Learning Theory*, COLT 2021, pages 3265–3295, Boulder, CO, USA, 2021. PMLR.

Shan Sung Liew, Mohamed Khalil-Hani, and Rabia Bakhteri. Bounded activation functions for enhanced training stability of deep neural networks on visual pattern recognition problems. *Neurocomputing*, 216:718–734, 2016. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2016.08.037.

Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, ICCV 2009, pages 2146–2153, Kyoto, Japan, 2009. IEEE.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 27th International Conference on Machine Learning*, ICML'10, pages 807–814. JMLR.org, 2010.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. In *6th International Conference on Learning Representations*, ICLR 2018, Vancouver, BC, Canada, 2018. OpenReview.net.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations*, ICLR 2016, 2016.

Solomon Kullback and Richard Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951. doi: 10.1214/aoms/1177729694.

Kevin Wainwright and Alpha C. Chiang. *Fundamental Methods of Mathematical Economics*. McGraw Hill, fourth edition, 2005.

Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping from saddle points - online stochastic gradient for tensor decomposition. In Peter Grünwald, Elad Hazan, and Satyen Kale, editors, *Proceedings of The 28th Conference on Learning Theory*, COLT 2015, pages 797–842, Paris, France, 2015. JMLR.org.

Chi Jin, Rong Ge, Praneeth Netrapalli, Sham M. Kakade, and Michael I. Jordan. How to escape saddle points efficiently. In Precup and Teh (2017), pages 1724–1732.

Bobby Kleinberg, Yuanzhi Li, and Yang Yuan. An alternative view: When does SGD escape local minima? In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80, pages 2698–2707, Stockholm, Sweden, 2018. PMLR.

Nelson Morgan and Hervé A. Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 630–637. Morgan-Kaufmann, 1990.

Russell Deryl Reed. Pruning algorithms-a survey. *Transactions on Neural Networks*, 4(5): 740–747, 1993. ISSN 1045-9227. doi: 10.1109/72.248452.

Lutz Prechelt. *Early Stopping - But When?* Springer-Verlag, London, UK, 1998. ISBN 3-540-65311-2.

Maren Mahsereci, Lukas Balles, Christoph Lassner, and Philipp Hennig. Early stopping without a validation set. *CoRR*, abs/1703.09580, 2017.

Jonas Sjöberg and Lennart Ljung. Overtraining, regularization and searching for a minimum, with application to neural networks. *International Journal of Control*, 62(6):1391–1407, 1995. doi: 10.1080/00207179508921605.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011. ISSN 1532-4435.

Geoffrey Hinton and Tijmen Tieleman. Lecture 6.5 - rmsprop, coursera: Neural networks for machine learning, 2012. URL http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Bengio and LeCun (2015).

Timothy Dozat. Incorporating nesterov momentum into adam. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations*, ICLR 2016, 2016.

Yurii Nesterov. A method of solving a convex programming problem with convergence rate $o(1/k^2)$. *Soviet Mathematics Doklady*, 27:372–376, 1983.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Dasgupta and McAllester (2013), pages 1139–1147.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the 13th*

International Conference on Artificial Intelligence and Statistics, volume 9 of *AISTATS 2010*, pages 249–256. PMLR, 2010.

David M. Bradley. *Learning in Modular Systems*. PhD thesis, The Robotics Institute, Carnegie Mellon University, 2010.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision*, ICCV, pages 1026–1034. IEEE, 2015. doi: 10.1109/ICCV.2015.123.

Siddharth Krishna Kumar. On weight initialization in deep neural networks. In *arXiv*, 2017. URL https://arxiv.org/abs/1704.08863.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Bach and Blei (2015), pages 448–456.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In Bengio et al. (2018).

Jonas Moritz Kohler, Hadi Daneshmand, Aurélien Lucchi, Thomas Hofmann, Ming Zhou, and Klaus Neymeyr. Exponential convergence rates for batch normalization: The power of length-direction decoupling in non-convex optimization. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, volume 89 of *AISTATS 2019*, pages 806–815. PMLR, 2019.

Tim Salimans and Durk P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 901–909, Red Hook, NY, USA, 2016. Curran Associates, Inc.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, chapter 8, pages 318—-362. MIT Press, Cambridge, MA, USA, 1986b. ISBN 026268053X.

Chung-Cheng Chiu, Tara N. Sainath, Yonghui Wu, Rohit Prabhavalkar, Patrick Nguyen, Zhifeng Chen, Anjuli Kannan, Ron J. Weiss, Kanishka Rao, Ekaterina Gonina, Navdeep Jaitly, Bo Li, Jan Chorowski, and Michiel Bacchiani. State-of-the-art speech recognition

with sequence-to-sequence models. In Dan Schonfeld, Pascale Fung, and Nam Ik Cho, editors, *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4774–4778, Calgary, AB, Canada, 2018. IEEE. doi: 10.1109/ICASSP.2018.8462105.

Yunus Emre Cebeci. A recurrent neural network model for weather forecasting. In *2019 4th International Conference on Computer Science and Engineering*, UBMK, pages 591–595. IEEE, 2019. doi: 10.1109/UBMK.2019.8907196.

Natasha Jaques, Shixiang Gu, Richard E. Turner, and Douglas Eck. Generating music by fine-tuning recurrent neural networks with reinforcement learning. In *5th International Conference on Learning Representations*, ICLR 2017. OpenReview.net, 2017.

Xiumin Li, Lin Yang, Fangzheng Xue, and Hongjun Zhou. Time series prediction of stock price using deep belief networks with intrinsic plasticity. In *Proceedings of the 29th Chinese Control And Decision Conference*, CCDC 2017, pages 1237–1242. IEEE, 2017. doi: 10.1109/CCDC.2017.7978707.

Arzoo Katiyar and Claire Cardie. Nested named entity recognition revisited. In Marilyn A. Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL-HLT 2018, pages 861–871, New Orleans, LA, USA, 2018. Association for Computational Linguistics. doi: 10.18653/v1/n18-1079.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In Dasgupta and McAllester (2013), pages 1310–1318.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. In *arXiv*, 2018. URL https://arxiv.org/abs/1803.01271.

Yoshua Bengio, Patrice Simard, , and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9 (8):1735—-1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735.

Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition.

IEEE Transactions on Pattern Analysis and Machine Intelligence, 31(5):855–868, 2009. doi: 10.1109/TPAMI.2008.137.

Nal Kalchbrenner, Lasse Espeholt, Aäron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. In *arXiv*, 2016. URL https://arxiv.org/abs/1609.03499.

Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In Grauman et al. (2015), pages 648–656.

Yann LeCun. *Modeles connexionnistes de l'apprentissage*. Université de Paris VI, June 1987. PhD thesis.

H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4–5):291–294, September 1988. ISSN 0340-1200. doi: 10.1007/BF00332918.

Geoffrey E. Hinton and Richard S. Zemel. Autoencoders, minimum description length and helmholtz free energy. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, pages 3–10, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

Pierre Baldi and K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2(1):53–58, January 1989. ISSN 0893-6080. doi: 10.1016/0893-6080(89)90014-2.

Lei Le, Andrew Patterson, and Martha White. Supervised autoencoders: Improving generalization performance with unsupervised regularizers. In Bengio et al. (2018), pages 107–117.

Brecht Evens, Puya Latafat, Andreas Themelis, Johan Suykens, and Panagiotis Patrinos. Neural network training as an optimal control problem: An augmented lagrangian approach. In Maria Prandini, editor, *Proceedings of the 60th IEEE Conference on Decision and Control*, CDC, pages 5136–5143, Fairmont, TX, USA, 2021. IEEE.

Shai Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2012. ISSN 1935-8237. doi: 10.1561/2200000018.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144, New York, NY, USA, 2016. ACM.

Keith Ambachtsheer. Profit potential in an almost efficient market. *Journal of Portfolio Management*, 1(1):84, FALL 1974.

Frank J. Fabozzi, James L. Grant, and Raman Vardharaj. *Common Stock Portfolio Management Strategies*, chapter 9, pages 229–270. Volume 1 of Fabozzi and Markowitz (2011), 2011b. ISBN 9781118267028.

Sergul Aydore, Tianhao Zhu, and Dean P. Foster. Dynamic local regret for non-convex online forecasting. In Wallach et al. (2019), pages 7980–7989.

Barbara Rossi and Atsushi Inoue. Out-of-sample forecast tests robust to the choice of window size. *Journal of Business & Economic Statistics*, 30(3):432–453, 2012.

Christoph Bergmeir, Rob Hyndman, and Bonsoo Koo. A note on the validity of cross-validation for evaluating autoregressive time series prediction. *Computational Statistics & Data Analysis*, 120:70–83, 4 2018. doi: 10.1016/j.csda.2017.11.003.

Yujie Liu, Hongbin Dong, Xingmei Wang, and Shuang Han. Time series prediction based on temporal convolutional network. In Simon Xu, Yongbin Wang, Mingyong Shi, Wenqiang Shang, Jiefeng Liu, and Kailong Zhang, editors, *2019 IEEE/ACIS 18th International Conference on Computer and Information Science*, ICIS 2019, pages 300–305, Beijing, China, 2019. IEEE.

Subhrajit Samanta, Mahardhika Pratama, Suresh Sundaram, and Narasimalu Srikanth. Learning elastic memory online for fast time series forecasting. *Neurocomputing*, 390:315–326, 2020. ISSN 0925-2312.

Nicolò Cesa-Bianchi, Pierre Gaillard, Gábor Lugosi, and Gilles Stoltz. A new look at shifting regret. In *arXiv*, 2012.

Thomas M. Cover. Universal portfolios. *Mathematical Finance*, 1(1):1–29, 1991. doi: 10.1111/j.1467-9965.1991.tb00002.x.

Elad Hazan, Karan Singh, and Cyril Zhang. Efficient regret minimization in non-convex games. In Precup and Teh (2017), pages 1433–1441.

Jeremiah Green, John R. M. Hand, and X. Frank Zhang. The characteristics that provide independent information about average U.S. monthly stock returns. *The Review of Financial Studies*, 30(12):4389–4436, 03 2017. ISSN 0893-9454. doi: 10.1093/rfs/hhx019.

Ivo Welch and Amit Goyal. A comprehensive look at the empirical performance of equity premium prediction. *The Review of Financial Studies*, 21(4):1455–1508, 03 2008. ISSN 0893-9454. doi: 10.1093/rfs/hhm014.

Joel L. Horowitz, Tim Loughran, and N.E. Savin. The disappearing size effect. *Research in Economics*, 54(1):83–100, 2000. ISSN 1090-9443.

U.S. Securities and Exchange Commission. Microcap stock: A guide for investors. https://www.sec.gov/reportspubs/investor-publications/investorpubsmicrocapstockhtm.html, Sep 2013. Accessed: 2021-01-03.

Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. *Efficient BackProp*, pages 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

Kent Daniel and Tobias J. Moskowitz. Momentum crashes. *Journal of Financial Economics*, 122(2):221–247, 2016. ISSN 0304-405X.

Spyros Makridakis and Michèle Hibon. The M3-competition: results, conclusions and implications. *International Journal of Forecasting*, 16(4):451–476, 2000. ISSN 0169-2070.

Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. Statistical and machine learning forecasting methods: Concerns and ways forward. *PLOS ONE*, 13(3): 1–26, 03 2018. doi: 10.1371/journal.pone.0194889.

Rob J. Hyndman. A brief history of forecasting competitions. *International Journal of Forecasting*, 36(1):7–14, 2020. ISSN 0169-2070. M4 Competition.

George E. P. Box, Gwilym M. Jenkins, and Gregory C. Reinsel. *Time Series Analysis: Forecasting and Control*. Prentice Hall, Englewood Cliffs, N.J., USA, 3 edition, 1994.

Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-BEATS: neural basis expansion analysis for interpretable time series forecasting. In *8th International Conference on Learning Representations*, ICLR 2020, Addis Ababa, Ethiopia, 2020. OpenReview.net.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Guyon et al. (2017),

pages 6000–6010.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL 2019, pages 4171–4186, Online, 2019. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Larochelle et al. (2020), pages 1877–1901.

Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *arXiv*, 2022. URL https://arxiv.org/abs/2205.13504.

J.B. Heaton, Nick Polson, and Jan Hendrik Witte. Deep learning in finance. In *arXiv*, 2016. URL https://arxiv.org/abs/1602.06561.

Jerzy Korczak and Marcin Hemes. Deep learning for financial time series forecasting in a-trader system. In *2017 Federated Conference on Computer Science and Information Systems*, FedCSIS, pages 905–912, Prague, Czech Republic, 2017. IEEE.

Eugene F. Fama. Random walks in stock market prices. *Financial Analysts Journal*, 21(5): 55–59, 1965. ISSN 0015198X.

Amado Peiró. The distribution of stock returns: international evidence. *Applied Financial Economics*, 4(6):431–439, 1994. doi: 10.1080/758518675.

Michael Isichenko. *Quantitative Portfolio Management: The Art and Science of Statistical Arbitrage*. Wiley, 2021.

Jining Yan, Lin Mu, Lizhe Wang, Rajiv Ranjan, and Albert Y. Zomaya. Temporal convolutional networks for the advance prediction of enso. *Scientific Reports*, 10(1), 2020.

Rui Dai, Shenkun Xu, Qian Gu, Chenguang Ji, and Kaikui Liu. Hybrid spatio-temporal graph convolutional network: Improving traffic prediction with navigation data. In *Proceedings*

of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 3074–3082, Virtual Event, CA, USA, 2020. ACM.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. In *arXiv*, 2016. URL https://arxiv.org/abs/1609.08144.

Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on signal processing*, 45(11):2673–2681, 1997.

Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In Lluís Màrquez, Chris Callison-Burch, and Jian Su, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, EMNLP 2015, pages 1412–1421, Lisbon, Portugal, 2015. Association for Computational Linguistics.

S. C. Suddarth and Y. L. Kergosien. Rule-injection hints as a means of improving network performance and learning time. In *Proceedings of the EURASIP Workshop 1990 on Neural Networks*, pages 120–129, Berlin, Heidelberg, 1990. Springer-Verlag. ISBN 3540522557.

Baruch Epstein and Ron Meir. Generalization bounds for unsupervised and semi-supervised learning with autoencoders. In *arXiv*, 2019. URL https://arxiv.org/abs/1902.01449.

Patrick Thiam, Hans A. Kestler, and Friedhelm Schwenker. Multimodal deep denoising convolutional autoencoders for pain intensity classification based on physiological signals. In Maria De Marsico, Gabriella Sanniti di Baja, and Ana L. N. Fred, editors, *Proceedings of the 9th International Conference on Pattern Recognition Applications and Methods*, ICPRAM 2020, pages 289–296. SCITEPRESS, 2020.

Michel Barlaud and Frederic Guyard. A non-parametric supervised autoencoder for discriminative and generative modeling. In *HAL*, September 2020. URL https:

//hal.archives-ouvertes.fr/hal-02937643. working paper or preprint.

Shantipriya Parida, Esau Villatoro-Tello, Sajit Kumar, Mael Fabien, and Petr Motlicek. Detection of similar languages and dialects using deep supervised autoencoders. In Pushpak Bhattacharyya, Dipti Misra Sharma, and Rajeev Sangal, editors, *Proceedings of the 17th International Conference on Natural Language Processing*, ICON 2020. SCITEPRESS, 2020.

M. Hiransha, E. A. Gopalakrishnan, Vijay Krishna Menon, and K. P. Soman. Nse stock market prediction using deep-learning models. *Procedia Computer Science*, 132:1351–1362, 2018. ISSN 1877-0509. International Conference on Computational Intelligence and Data Science.

Rohitash Chandra and Shelvin Chand. Evaluation of co-evolutionary neural network architectures for time series prediction with mobile application in finance. *Applied Soft Computing*, 49:462–473, 2016. ISSN 1568-4946.

Bryan Lim, Stefan Zohren, and Stephen Roberts. Enhancing time-series momentum strategies using deep neural networks. *Journal of Financial Data Science*, 2019. ISSN 2405-9188. doi: 10.3905/jfds.2019.1.015.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In Grauman et al. (2015), pages 3431–3440.

Steven Y. K. Wong, Jennifer Chan, Lamiae Azizi, and Richard Y. D. Xu. Supervised temporal autoencoder for stock return time-series forecasting. In Wing Kwong Chan, Bill Claycomb, and Hiroki Takakura, editors, *Proceedings of the IEEE 45th Annual Computer Software and Applications Conference (COMPSAC'21)*, Madrid, Spain, 2021. IEEE.

George E. P. Box and Gwilym M. Jenkins. *Time series analysis: forecasting and control*. Holden-Day series in time series analysis and digital processing. Holden-Day, San Francisco, rev. ed. edition, 1976. ISBN 0816211043.

Helmut Lütkepohl and Fang Xu. The role of the log transformation in forecasting economic variables. *Empirical Economics*, 42(3):619–638, 2012.

Thomas Fischer and Christopher Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270 (2):654–669, 2018. ISSN 0377-2217.

Fischer Black and Robert B Litterman. Asset allocation: Combining investor views with market equilibrium. *Journal of Fixed Income*, 1(2):7–18, 1991.

Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseo Lee, Matthias Humt, Jianxiang Feng, Anna M. Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, M. Shahzad, Wen Yang, Richard Bamler, and Xiaoxiang Zhu. A survey of uncertainty in deep neural networks. In *arXiv*, 2021. URL https://arxiv.org/abs/2107.03342.

Cornelia Gruber, Patrick Oliver Schenk, Malte Schierholz, Frauke Kreuter, and Göran Kauermann. Sources of uncertainty in machine learning – a statisticians' view. In *arXiv*, 2023. URL https://arxiv.org/abs/2305.16703.

Nis Meinert, Jakob Gawlikowski, and Alexander Lavin. The unreasonable effectiveness of deep evidential regression. In *arXiv*, 2022. URL https://arxiv.org/abs/2205.10060.

T.J. Sullivan. *Introduction to Uncertainty Quantification*. Number 63 in Texts in Applied Mathematics. Springer International Publishing, Cham, Germany, 1st edition, 2015. ISBN 3-319-23395-5.

S. C. Hora. Aleatory and epistemic uncertainty in probability elicitation with an example from hazardous waste management: Treatment of aleatory and epistemic uncertainty. *Reliability engineering & system safety*, 54(2–3):217–223, 1996. ISSN 0951-8320.

David J. C. MacKay. A practical bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 1992. ISSN 0899-7667. doi: 10.1162/neco.1992.4.3.448.

Radford M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag, Berlin, Heidelberg, 1996. ISBN 0387947248.

Yarin Gal. *Uncertainty in Deep Learning*. University of Cambridge, 2016. PhD thesis.

Wilfred Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

Laurent Valentin Jospin, Hamid Laga, Farid Boussaid, Wray Buntine, and Mohammed Bennamoun. Hands-on bayesian neural networks — a tutorial for deep learning users. *IEEE computational intelligence magazine*, 17(2):29–48, 2022.

Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. Being bayesian, even just a bit, fixes overconfidence in relu networks. In *arXiv*, 2020. URL https://arxiv.org/abs/2002.10118.

Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D. Sculley, Sebastian Nowozin, Joshua V. Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In Wallach et al. (2019), pages 14003–14014.

José M. Bernardo and Adrian F. M. Smith. *Bayesian theory*. John Wiley & Sons Ltd., 2000.

Zhijian Liu, Alexander Amini, Sibo Zhu, Sertac Karaman, Song Han, and Daniela L. Rus. Efficient and robust lidar-based end-to-end navigation. In Yu Sun, editor, *2021 IEEE International Conference on Robotics and Automation*, ICRA, pages 13247–13254, Xi'an, China, 2021. IEEE.

Peide Cai, Hengli Wang, Huaiyang Huang, Yuxuan Liu, and Ming Liu. Vision-based autonomous car racing using deep imitative reinforcement learning. *IEEE Robotics and Automation Letters*, 6(4):7262–7269, 2021. doi: 10.1109/LRA.2021.3097345.

Sandeep Kumar Singh, Jaya Shradha Fowdur, Jakob Gawlikowski, and Daniel Medina. Leveraging graph and deep learning uncertainties to detect anomalous maritime trajectories. *IEEE Transactions on Intelligent Transportation Systems*, 23(12):23488–23502, 2022. doi: 10.1109/TITS.2022.3190834.

Ava P. Soleimany, Alexander Amini, Samuel Goldman, Daniela Rus, Sangeeta N. Bhatia, and Connor W. Coley. Evidential deep learning for guided molecular property prediction and discovery. *ACS central science*, 7(8):1356–1367, 2021. ISSN 2374-7943.

Hao Li and Jianan Liu. 3D high-quality magnetic resonance image restoration in clinics using deep learning. In *arXiv*, 2022. URL https://arxiv.org/abs/2111.14259.

Dongpin Oh and Bonggun Shin. Improving evidential deep learning via multi-task learning. In *Thirty-Sixth AAAI Conference on Artificial Intelligence*, pages 7895–7903. AAAI Press, 2022.

Bertrand Charpentier, Oliver Borchert, Daniel Zügner, Simon Geisler, and Stephan Günnemann. Natural posterior network: Deep bayesian predictive uncertainty for exponential family distributions. In *9th International Conference on Learning Representations (ICLR)*,

online, 2021. OpenReview.net.

Andrey Malinin, Sergey Chervontsev, Ivan Provilkov, and Mark Gales. Regression prior networks. In *arXiv*, 2020. URL https://arxiv.org/abs/2006.11590.

Tilmann Gneiting and Adrian E. Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

Viktor Bengs, Eyke Hüllermeier, and Willem Waegeman. On second-order scoring rules for epistemic uncertainty quantification. In *arXiv*, 2023. URL https://arxiv.org/abs/2301.12736.

Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307–327, 1986. ISSN 0304-4076. doi: https://doi.org/10.1016/0304-4076(86)90063-1.

Rachael Carroll and Colm Kearney. *GARCH Modeling of Stock Market Volatility*, pages 71–90. Chapman & Hall/CRC finance series. CRC Press, New York, NY, USA, 1st edition, 2009.

John Fry and Eng-Tuck Cheah. Negative bubbles and shocks in cryptocurrency markets. *International Review of Financial Analysis*, 47:343–352, 2016. ISSN 1057-5219.

Christian M. Hafner. Testing for Bubbles in Cryptocurrencies with Time-Varying Volatility. *Journal of Financial Econometrics*, 18(2):233–249, 10 2018. ISSN 1479-8409. doi: 10.1093/jjfinec/nby023.

Cathy Yi-Hsuan Chen and Christian M. Hafner. Sentiment-induced bubbles in the cryptocurrency market. *Journal of Risk and Financial Management*, 12(2):1–12, 2019. ISSN 1911-8074.

José Antonio Núñez, Mario I. Contreras-Valdez, and Carlos A. Franco-Ruiz. Statistical analysis of bitcoin during explosive behavior periods. *PLOS ONE*, 14(3):1–22, 03 2019. doi: 10.1371/journal.pone.0213919.

Alla Petukhina, Simon Trimborn, Wolfgang Karl Härdle, and Hermann Elendner. Investing with cryptocurrencies – evaluating their potential for portfolio allocation strategies. *Quantitative Finance*, 21(11):1825–1853, 2021.

José Miguel Hernández-Lobato and Ryan P. Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In Bach and Blei (2015), pages 1861–1869.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, volume 48, pages 1050–1059. JMLR.org, 2016.

Michael Jordan. The exponential family: Conjugate priors, 2009.

David F. Andrews and Colin L. Mallows. Scale mixtures of normal distributions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 36(1):99–102, 1974.

S.T. Boris Choy and Jennifer S.K. Chan. Scale mixtures distributions in statistical modelling. *Australian & New Zealand Journal of Statistics*, 50(2):135–146, 2008. ISSN 1369-1473.

Christian Brownlees, Robert Engle, and Bryan Kelly. A practical guide to volatility forecasting through calm and storm. *Journal of Risk*, 14(2):3–22, 2011.

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996. ISSN 1573-0565.

Adrian E. Raftery, Miroslav Kárný, and Pavel Ettler. Online prediction under model uncertainty via dynamic model averaging: Application to a cold rolling mill. *Technometrics*, 52 (1):52–66, 2010. doi: 10.1198/TECH.2009.08104. PMID: 20607102.

Robert F. Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, 50(4):987–1007, 1982.

Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. A comparison of arima and lstm in forecasting time series. In M. Arif Wani, editor, *Proceedings of the 17th IEEE International Conference on Machine Learning and Applications*, pages 1394–1401, Orlando, FL, USA, 2018. IEEE.

Alexander Lipton. Cryptocurrencies change everything. *Quantitative Finance*, 21(8):1257–1262, 2021. doi: 10.1080/14697688.2021.1944490.

Wenbo Ge, Pooia Lalbakhsh, Leigh Isai, Artem Lenskiy, and Hanna Suominen. Neural network–based financial volatility forecasting: A systematic review. *ACM Computing Surveys*, 55(1), 2022.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In Bengio et al. (2018).

Robert A. Wood, Thomas H. McInish, and J. Keith Ord. An investigation of transactions data for nyse stocks. *Journal of Finance*, 40(3):723–739, 1985.

Prem C. Jain and Gun-Ho Joh. The dependence between hourly prices and trading volume. *Journal of Financial and Quantitative Analysis*, 23(3):269–283, 1988.

Thomas H. McInish and Robert A. Wood. An analysis of intraday patterns in bid/ask spreads for nyse stocks. *Journal of Finance*, 47(2):753–764, 1992.

James Eaves and Jeffrey Williams. Are intraday volume and volatility u-shaped after accounting for public information? *American Journal of Agricultural Economics*, 92(1):212–227, 2010.

Larry J. Lockwood and Scott C. Linn. An examination of stock market return volatility during overnight and intraday periods, 1964-1989. *Journal of Finance*, 45(2):591–601, 1990. ISSN 00221082, 15406261. URL http://www.jstor.org/stable/2328672.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

Frederick C. Webber. *Multi-Objective Reinforcement Learning with Concept Drift*. PhD thesis, Air Force Institute of Technology, 2017.

Aswath Damodaran. *Damodaran on Valuation: Security Analysis for Investment and Corporate Finance*. Wiley, 2 edition, 2006.

Dogu Araci. Finbert: Financial sentiment analysis with pre-trained language models. In *arXiv*, 2019. URL https://arxiv.org/abs/1908.10063.

Tom M. Mitchell. *Machine Learning*. McGraw-Hill Education, 1 edition, 1997.

Sebastian Ruder, Matthew E. Peters, Swabha Swayamdipta, and Thomas Wolf. Transfer learning in natural language processing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*, pages 15–18, Minneapolis, MN, USA, 2019. Association for Computational Linguistics.

Wenjuan Han, Bo Pang, and Ying Nian Wu. Robust transfer learning with pretrained language models through adapters. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 854–861, Online, 2021. Association for Computational Linguistics.

Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors. *Advances in Neural Information Processing*

Systems 30, Long Beach, CA, USA, 2017. Curran Associates, Inc.

Hugo Larochelle, Marc'Aurelio Ranzato, Raia, Hadsell, Maria-Florina Balcan, and Hui Lin, editors. *Advances in Neural Information Processing Systems 33*, NIPS 2020, Vancouver, BC, Canada, 2020. Curran Associates, Inc.

Yoshua Bengio and Yann LeCun, editors. *3rd International Conference on Learning Representations*, ICLR 2015, San Diego, CA, USA, 2015.

Kristen Grauman, Erik Learned-Miller, Antonio Torralba, and Andrew Zisserman, editors. *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR, 2015. IEEE.

Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors. *Advances in Neural Information Processing Systems 27*, NIPS, 2014. Curran Associates, Inc.

Frank J. Fabozzi and Harry M. Markowitz, editors. *The Theory and Practice of Investment Management*, volume 1. John Wiley & Sons, Ltd, 2011. ISBN 9781118267028.

Doina Precup and Yee Whye Teh, editors. *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70, Sydney, NSW, Australia, 2017. JMLR.org.

Sanjoy Dasgupta and David McAllester, editors. *Proceedings of the 30th International Conference on Machine Learning*, ICML'13, 2013. JMLR.org.

Francis R. Bach and David M. Blei, editors. *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 37, 2015. JMLR.org.

Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors. *Advances in Neural Information Processing Systems 31*, NIPS'18, Montréal, QC, Canada, 2018. Curran Associates, Inc.

Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors. *Advances in Neural Information Processing Systems 32*, Vancouver, BC, Canada, 2019. Curran Associates, Inc.

John Lintner. The valuation of risk assets and the selection of risky investments in stock portfolios and capital budgets. *Review of Economics & Statistics*, 47(1):13–37, 1965. ISSN 00346535.

Fischer Black. Capital market equilibrium with restricted borrowing. *Journal of Business*, 45 (3):444–455, 1972. ISSN 00219398, 15375374.

Malcolm Baker, Brendan Bradley, and Jeffrey Wurgler. Benchmarks as limits to arbitrage: Understanding the low-volatility anomaly. *Financial Analysts Journal*, 67(1):40–54, 2011.

Richard Roll. A critique of the asset pricing theory's tests part i: On past and potential testability of the theory. *Journal of Financial Economics*, 4(2):129–176, 1977. ISSN 0304-405X. doi: https://doi.org/10.1016/0304-405X(77)90009-5.

Eugene F. Fama and Kenneth R. French. The capital asset pricing model: Theory and evidence. *Journal of Economic Perspectives*, 18(3):25–46, September 2004. doi: 10.1257/0895330042162430. URL http://www.aeaweb.org/articles?id=10.1257/0895330042162430.

John H. Cochrane. *Asset Pricing*. Princeton University Press, 2005. ISBN 0691121370.

Richard G. Sloan. Do stock prices fully reflect information in accruals and cash flows about future earnings? *Accounting Review*, 71(3):289–315, 1996.

Michael J. Cooper, Huseyin Gulen, and J. Schill Michael. Asset growth and the cross-section of stock returns. *Journal of Finance*, 63(4):1609–1651, 2008.

Andrew Ang, Robert J. Hodrick, Yuhang Xing, and Xiaoyan Zhang. The cross-section of volatility and expected returns. *Journal of Finance*, 61(1):259–299, 2006. doi: 10.1111/j.1540-6261.2006.00836.x.

Robert Novy-Marx. The other side of value: The gross profitability premium. *Journal of Financial Economics*, 108(1):1–28, 2013. ISSN 0304-405X. doi: https://doi.org/10.1016/j.jfineco.2013.01.003.

Stephen A. Ross. The arbitrage theory of capital asset pricing. *Journal of Economic Theory*, 13 (3):341–360, 1976. ISSN 0022-0531. doi: https://doi.org/10.1016/0022-0531(76)90046-6.

Frank J. Fabozzi, Raman Vardharaj, , and Frank J. Jones. *Multifactor Equity Risk Models*, chapter 13, pages 327–343. Volume 1 of Fabozzi and Markowitz (2011), 2011c. ISBN 9781118267028.

CFA Institute. A practitioner's guide to factor models. Technical report, The Research Foundation of The Institute of Chartered Financial Analysts, Charlottesville, VA, USA, 1994. URL https://www.cfainstitute.org/-/media/documents/book/

rf-publication/1994/rf-v1994-n4-4445-pdf.ashx.

Victor Dheur and Souhaib Ben Taieb. A large-scale study of probabilistic calibration in neural network regression. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202, pages 7813–7836. JMLR.org, 2023.

APPENDIX A

Appendix

A1 Supplementary review of asset pricing

A supplementary review of the finance theory relevant for predictions in financial markets is provided in this section. Readers are encouraged to this read this section as preface to Section 1.3. However, this is not a prerequisite to understanding the rest of this thesis.

Consider the following scenario, an investor is making an investment decision between two assets: 1) a government bond that pays a guaranteed coupon; and 2) a highly risky stock. Assuming that the investor is risk averse, it is clear that the stock must offer a higher return than the government bond to compensate for the higher risk. Given an asset that earns an uncertain return, how much should an investor expect in return for holding this asset? In financial theory, this problem is known as *asset pricing*. The seminal work by Markowitz (1952) led to the development of the MPT, proposing to model the first two moments of stock returns. Suppose asset i has *expected return* $\mathrm{E}[r_i]$ and *risk* measured as variance of expected return σ_i^2. Investors are assumed to be risk averse and seek to select portfolios that maximise expected portfolio return,

$$\mathrm{E}[r_p] = \sum_i \mathrm{w}_i \, \mathrm{E}[r_i], \tag{A.1}$$

where r_p is portfolio return, w_i is portfolio holdings of asset i (for simplicity, it is assumed that $\sum_i \mathrm{w}_i = 1$), and minimise portfolio return variance,

$$\sigma_p^2 = \sum_i \mathrm{w}_i^2 \, \sigma_i^2 + \sum_i \sum_{j \neq i} \mathrm{w}_i \, \mathrm{w}_j \, \sigma_i \sigma_j \rho_{ij}, \tag{A.2}$$

where σ_p^2 is portfolio return variance, ρ_{ij} is correlation between expected return of asset i and j. This led to the development of the first asset pricing model, called the CAPM (Sharpe, 1964; Lintner, 1965; Black, 1972). The CAPM offered the first stock return prediction model with a theoretical underpinning,

$$\mathrm{E}[r_i] = r^* + \beta_i(\mathrm{E}[r'] - r^*), \tag{A.3}$$

where r^* is the risk-free rate (e.g., return on government bonds which are assumed to be risk-free), $\mathrm{E}[r']$ is the expected return of the market[1], and $\beta_i = \frac{\mathrm{Cov}(r_i, r')}{\mathrm{Var}(r')}$ is the sensitivity of expected return of asset i to the market. Exclusive to this section, we denote statistics related to the risk free rate with asterisk $*$ and the market with dash $'$. In CAPM, $\mathrm{E}[r']$ can be interpreted as factor m in Equation 1.3. However, it is typically assumed to be the long-term average market return. The intuition for CAPM is as follows. Suppose there are two stocks, both have expected return of $8\,\%$ and return standard deviation of $10\,\%$. Correlation between the two stocks is $50\,\%$. Then, from Equation (A.1) and (A.2), a portfolio with $50\,\%$ weight in each stock has portfolio expected return of $8\,\%$ and standard deviation of $7.9\,\%$. Thus, the diversified portfolio offers a better return/risk trade-off than either of the two stocks on its own. Generalising, there exists a well-diversified portfolio that offers the same level of expected return with equal or less level of risk (due to imperfect correlation between assets), as illustrated on the left of Figure A.1. The red curve on the left of Figure A.1 illustrates the *Efficient Frontier*, which represents the portfolio with the lowest possible risk (x-axis) for each level of expected return (y-axis). Note that all individual stocks (e.g., *C* and *D*) must lie on or within the interior of the efficient frontier. MPT assumes that there exists a well-diversified portfolio which offers the optimal mean-variance trade-off. This portfolio is called the *market portfolio*. The tangent to the red curve which crosses the y-axis at the risk-free rate is called the Security Market Line (SML). Note that SML intersects the Efficient Frontier at the market portfolio and is above the Efficient Frontier for all other levels of risk. Portfolios on this line represent allocations between the risk-free rate and the market portfolio. For example, if the desired level of risk is point **A** in Figure A.1, the level of risk is lower than the market portfolio

[1]The *market portfolio* is the portfolio of all assets weighted by their respective market value. Market value (also called market capitalisation) is the price of each stock multiplied by the number of shares on issue. Excess return of the market $\mathrm{E}[r'] - r^*$ is known as *market risk premium*.

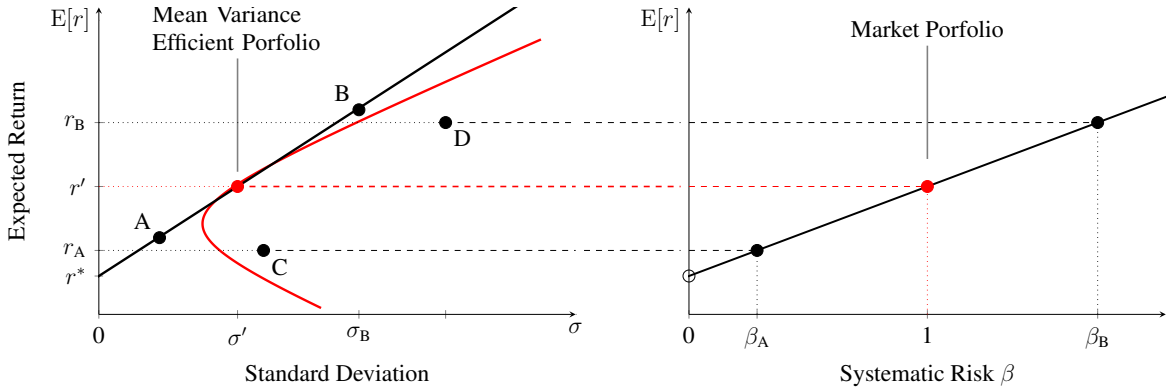


FIGURE A.1: Illustration of the Capital Asset Pricing Model. **Left**: Expected return of assets relative to standard deviation of returns. The red curve is known as the *Efficient Frontier*, where every point on the curve represents a possible portfolio combination. The tangent that passes through r^* (risk free rate) is the *Security Market Line*. **Right**: Expected return relative to systematic risk β. The market portfolio has $\beta' = 1$. A stock that has $\beta < 1$ (point **C**) is expected to earn a lower return than the market, while a stock with $\beta > 1$ (point **D**) is expected to earn a higher return than the market.

(or any risky portfolios on the Efficient Frontier). Point **A** represents a partial allocation to the risk-free rate (i.e., invest in government bonds) with the balance invested in the market portfolio. Conversely, if the desired level of risk is point **B**, the investor borrows at risk-free rate and invests the entire amount in the market portfolio (in MPT, borrowing at the risk-free rate is assumed to be accessible by all investors). In either case, the investor can achieve a better return/risk trade-off by allocating between the risk-free rate and the market portfolio than any of the portfolios on the Efficient Frontier or individual assets. Thus, a corollary of CAPM is that all investors hold the market portfolio and vary their allocation to the risk-free rate to arrive at their risk preference.

Due to the diversification benefits of the market portfolio, CAPM stipulates that the expected return of an individual asset is determined by its correlation to the market portfolio, known as systematic risk[2], and not by the total risk (i.e., variance) of the asset. In other words, only risk that arises due to correlation with the market portfolio is compensated (in the form of higher expected return), as this cannot be further diversified away by holding the market

[2]Denoted as β_i or *beta* in finance literature. Systematic risk is assumed to be non-diversifiable (Sharpe, 1964).

portfolio. Idiosyncratic risk that is specific to the asset is not compensated as there exists a well-diversified portfolio that offers the same level of expected return at a lower risk. Expected return as determined by correlation to the market is illustrated on the right of Figure A.1. In here, the market portfolio has $\beta' = 1$, while stock A (B) with $\beta_A < 1$ ($\beta_B > 1$) is expected to earn return $\mathrm{E}[r_A] < E[r']$ ($\mathrm{E}[r_B] > E[r']$), respectively. Underpinning both MPT and CAPM is the *efficient market hypothesis* (Fama, 1970). In the weak form, the hypothesis postulates that investors cannot outperform the market using publicly known information.

Therefore, one may arrive at the conclusions: 1) predicting stock returns is just a matter of estimating β; and 2) portfolio selection is of little value given that the market portfolio offers the optimal return/risk trade-off. However, empirical evidence proved otherwise. Jensen (1968) was the first to note that CAPM implied a time-series regression,

$$r_{i,t} - r^* = \alpha_i + \beta_i(r'_t - r^*) + \epsilon_{i,t}, \tag{A.4}$$

where $r_{i,t}$ is return of stock i at time t (at a given frequency, e.g. monthly), r'_t is market return at t and $\epsilon_{i,t}$ is stochastic noise. The empirical evidence suggested that α was statistically significantly above zero, which contradicted the predictions of CAPM. A positive α indicates that low β stocks earn a return that is higher than fair compensation for bearing market risk, an anomaly known as the *low beta effect* (Baker et al., 2011). Roll (1977) provided a renown critique of CAPM, noting the that a test of the validity of CAPM is a joint test together with the validity of the market portfolio proxy, and that the true market portfolio is unobservable (as it emcompasses all assets in the world, including private assets). A more recent discussion on CAPM was provided by Fama and French (2004). The authors reviewed published empirical tests of CAPM and concluded that "CAPM's empirical problems probably invalidate its use in applications."

Since the publication of CAPM, numerous other firm features (also known as risk factors or asset pricing *anomalies*[3]) were found to predict cross-sectional stock returns. These include

[3]A variable that is predictive of stock returns is called an *anomaly* as its occurrence contradicted the prediction of CAPM (Cochrane, 2005).

the size effect[4] (Banz, 1981), the value effect[5] (Stattman, 1980; Rosenberg et al., 1985)), accruals[6] (Sloan, 1996), asset growth[7] (Cooper et al., 2008), momentum[8] (Jegadeesh and Titman, 1993), low volatility effect[9] (Ang et al., 2006) and gross profitability[10] (Novy-Marx, 2013). Hundreds of firm characteristics are said to contain information on future stock returns — a survey by Harvey et al. (2016) contained 313 published asset pricing anomalies. The true DGP or stock returns is likely to be significantly more complex than originally suggested by CAPM.

Alternative asset pricing theories have been proposed. One alternative is the APT (Ross, 1976), formally,

$$\mathrm{E}[r_{i,t+1}] = r^* + s_{i,1}\xi_{1,t} + s_{i,2}\xi_{2,t} + ... + s_{i,M}\xi_{M,t}, \tag{A.5}$$

where $s_{i,m}$ is sensitivity (also called exposures in finance literature, e.g., Grinold and Kahn, 1999) of stock i to returns of *systematic risk factor*[11] ξ_m (also called factor returns). APT is a generalisation of CAPM, where systematic risk to the market in CAPM is replaced with any number of factors that determine stock returns. The intuition of the model lies in the assumption that given M non-diversifiable risk factors, one can replicate risk exposures of any stock (i.e., sensitivities of the stock to M predictors) with a portfolio of stocks that offer the same return. If the two do not offer the same return, an investor can buy the asset with a higher return and short[12] the asset with a lower expected return and generate a risk-free profit. Such activity is called *arbitrage* (hence the name Arbitrage Pricing Theory) and leads to the *no arbitrage condition* under efficient markets. This model forms the basis of quantitative investing methods used by practitioners, as described in Grinold and Kahn (1999). However,

[4]Stocks with low market capitalisation tend to outperform stocks with high market capitalisation.

[5]Stocks with high book value to market value ratio (known as value stocks) tend to outperform stocks with low book-to-market ratio (known as growth stocks).

[6]Firms with low operating accruals tend to outperform firms with high accruals. Operating accrual refers to the difference between accrued earnings and cash earnings.

[7]Firms with low year-on-year change in total assets tend to outperform firms with high growth.

[8]Past winners tend to continue to outperform past losers.

[9]Stocks with low idiosyncratic volatility tend to outperform stocks with high idiosyncratic volatility.

[10]Stocks with high gross profit to total assets tend to outperform stocks with low gross profit to total assets.

[11]Systematic risk factors are factors that contribute to expected return. An example is the market portfolio in CAPM. For the purposes of this thesis, this can be interpreted as firm features (e.g., firm profitability, social media sentiment), to the extent that such features are predictive of stock returns.

[12]Shorting refers to the borrowing and selling of an asset that the investor do not own. If the price of the asset falls, the investor can buy it back at a later date and generate a profit.

APT does not stipulate what the risk factors are. The risk factors used are up to the user to define. As concluding remarks on asset pricing theory, the debate on the correct model is one between a highly prescriptive model that specifies a theory-derived return predictor (CAPM β), and a loosely defined model where there could be any number of predictors (APT). The empirical failings of CAPM and the vast number of "anomalies" discovered is providing evidence to the latter.

A2 Supplementary review of forecasting models

A supplementary review of the quantitative investment process is provided in this section. Readers are encouraged to review this section for a better understanding of the quantitative investment process. However, this is not a prerequisite to understanding the rest of this thesis.

A2.1 Forecasting returns

Return forecasting (as illustrated in Figure A.2) requires converting raw data into features. Raw data can comprise of stock price history, company financials, social media sentiment,

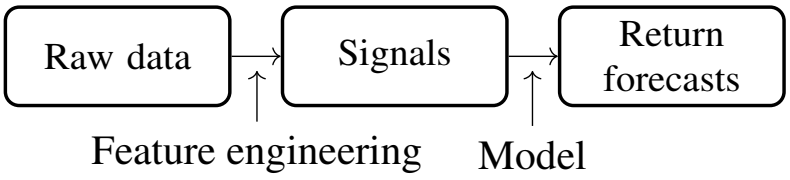


FIGURE A.2: Illustrative steps in converting raw data into forecasts. Raw data (e.g., share price and company financials) are first converted into features, typically crafted using domain knowledge. A forecasting model (e.g., OLS or machine learning models) then combines features into an expected return.

media reports, commodity prices, currency exchange rates or any other information that is predictive of returns. Through feature engineering, these raw data are transformed into signals. For example, the Earnings/Price ratio is calculated as a firm's earnings per share divided by the prevailing share price (Alford et al., 2011). This signal constitutes one of the many features that are used as inputs to the forecasting model. A linear regression, such as the

Fama-MacBeth regression, is a popular choice of F in Equation (1.2) (Zhou and Fabozzi, 2011)). However, as shown by Gu et al. (2020) and through this thesis, machine learning models can improve forecast accuracy and offer alternative approaches to forecasting.

A2.2 Forecasting risk

In the pioneering work of Markowitz (1952), a portfolio is viewed in terms of its first two moments of portfolio returns (mean and variance). This is the industry accepted general measure of risk (for example, see Grinold and Kahn, 1999; and Fabozzi and Markowitz, 2011). However, estimating a full variance-covariance matrix of a large universe of stocks could be onerous. For instance, there are more than 5,000 listed stocks in the U.S. alone. A variance-covariance matrix for the U.S. market could have $5000 \times 5000 = 25$million entries. Therefore, the variance-covariance matrix is typically estimated using a *factor risk model* (Fabozzi et al., 2011c). A factor risk model takes the same form as the return forecasting model (Equation (1.3)) and may include factors that are assumed to explain riskiness of a stock, such as its company size and debt burden. Suppose $M = 10$, the variance-covariance matrix only has $10 \times 10 = 100$ entries — a significantly simpler computation than operating over 25 million entries. The factor risk model is estimated using the Fama-MacBeth procedure as described in the preceding section. At time t, we perform $t-1$ cross-sectional multivariate regressions for every time period. This generates m time-series of factor returns $\hat{\boldsymbol{\xi}}_{t-1,m} = \{\hat{\xi}_{1,m}, \hat{\xi}_{2,m}, \ldots, \hat{\xi}_{t-1,m}\}$, where each value $\hat{\xi}_{t,m}$ in the series is the coefficient of factor[13] m in the multivariate cross-sectional regression at t. The factor covariance matrix is simply the sample variance-covariance matrix of the time-series of factor returns,

$$\hat{\boldsymbol{V}}_t^{(f)} = \begin{bmatrix} \mathrm{Cov}(\hat{\boldsymbol{\xi}}_{t-1,1}, \hat{\boldsymbol{\xi}}_{t-1,1}) & \mathrm{Cov}(\hat{\boldsymbol{\xi}}_{t-1,1}, \hat{\boldsymbol{\xi}}_{t-1,2}) & \cdots & \mathrm{Cov}(\hat{\boldsymbol{\xi}}_{t-1,1}, \hat{\boldsymbol{\xi}}_{t-1,M}) \\ \mathrm{Cov}(\hat{\boldsymbol{\xi}}_{t-1,2}, \hat{\boldsymbol{\xi}}_{t-1,1}) & \mathrm{Cov}(\hat{\boldsymbol{\xi}}_{t-1,2}, \hat{\boldsymbol{\xi}}_{t-1,2}) & \cdots & \mathrm{Cov}(\hat{\boldsymbol{\xi}}_{t-1,2}, \hat{\boldsymbol{\xi}}_{t-1,M}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cov}(\hat{\boldsymbol{\xi}}_{t-1,M}, \hat{\boldsymbol{\xi}}_{t-1,1}) & \mathrm{Cov}(\hat{\boldsymbol{\xi}}_{t-1,M}, \hat{\boldsymbol{\xi}}_{t-1,2}) & \cdots & \mathrm{Cov}(\hat{\boldsymbol{\xi}}_{t-1,M}, \hat{\boldsymbol{\xi}}_{t-1,M}) \end{bmatrix}, \quad \text{(A.6)}$$

[13] As a reminder to readers, signals, factors and firm features are used interchangeably in this thesis.

where $\mathrm{Cov}(\hat{\boldsymbol{\xi}}_{t-1,i}, \hat{\boldsymbol{\xi}}_{t-1,j})$ is the covariance function applied to returns of factor i and j over $t = \{1, \ldots, t-1\}$. Covariance matrix of stocks is then estimated as the sum of *factor risk* and *specific risk* (Fabozzi et al., 2011c),

$$\boldsymbol{V}_t' = \boldsymbol{X}_t \boldsymbol{V}_t^{(f)} \boldsymbol{X}_t^{\mathsf{T}} + \boldsymbol{d}_t \circ \boldsymbol{I}_N, \tag{A.7}$$

where $\boldsymbol{d}_t \circ \boldsymbol{I}_N$ is the Hadamard product of specific variance and the identity matrix of size N. $\boldsymbol{d}_t$ is the element-wise variance of the time-series of regression residuals $\boldsymbol{\epsilon}_t$ in Equation (1.3) (CFA Institute, 1994). This allows newly listed stocks to have reasonable risk forecasts, provided that they have the relevant inputs to the risk model.

A2.3 Forecasting transaction costs

As noted in Section 1.2, the stock market employs a *limit order book* system. In the example illustrated in Table 1.1, the last traded price is \$1.00/share. Suppose the buyer would like to transact 10,000 shares immediately, pushing the last traded price to \$1.02 (second row of the ask queue). The buyer's average price is $\frac{5{,}000 \times 1.01 + 5{,}000 \times 1.02}{10{,}000} = 1.015$. In this scenario, the buyer has incurred a market impact cost of $1.5\,\%$. Market impact comprises of the *bid-ask spread*[14] and a cost for pushing the traded price higher than the last traded price before the trade. Market impact depends on the size of the trade and tends to be higher for low-price and small capitalisation stocks (Zhou and Fabozzi, 2011). Larger trades will consume more of the available volume (termed liquidity) and have a higher price impact. Forecasting market impact is difficult, as market participants only observe completed trades and not intended orders. For example, in the example illustrated in Table 1.1, suppose a buyer wants to trade 100,000 shares at \$1.01. Seeing the limited shares on offer and concerned about potential price impact, the buyer decides to not place the order. However, there could be other potential sellers wanting to sell large parcels but face the same problem. Thus, liquidity can potentially be more than actual traded volumes. Transaction cost forecasting is also an important topic as it determines the economics of a trade. For instance, suppose the expected return of a stock over a week is $2\,\%$ and this opportunity is available every week. This results in an attractive

[14] *Crossing the spread* is when a buyer pays the ask price or a seller takes the bid price for a trade. Bid-ask spread is typically measured relative to the mid price and is calculated as $\frac{2 \times (1.01 - 1)}{1.01 + 1} = 1\,\%$.

compounded pre-cost return of $180\,\%$ in one year. However, suppose transaction cost is $1\,\%$ each way (i.e., $1\,\%$ is paid on acquisition and on disposal). Then, this seemingly profitable strategy is no longer profitable. Additionally, each trade incurs a commission to the broker (facilitator of the trade, usually a brokerage firm).

A3 Hyperparameters used in Chapter 3

In this section, we provide a list of hyperparameter search ranges and mean hyperparameters used to train the neural networks in Section 3.5. Hyperparameter search was performed on all combinations of hyperparameters. Table A.2 records mean hyperparameters over 10 network trainings.

TABLE A.1: Disclosed model parameters in Gu et al. (2020) and in our replication. We fill missing values with the cross-sectional median or zero if median is unavailable. 'H' is hidden layer activation. 'O' is output layer activation. ADAM is the optimiser proposed by Kingma and Ba (2015).

Parameter	Gu et al. (2020)	Chapter 3
Preprocessing	Rank [-1, 1]; Fill median	Rank [-1, 1]; Fill median/0
Hidden layers	32-16-8	32-16-8
Activation	H: ReLU / O: Linear	H: ReLU / O: Linear
Batch size	10,000	DNN 10,000 / OES 1,000
Batch normalisation	Yes	Yes
L_1 penalty	$[10^{-5}, 10^{-3}]$	$\{10^{-5}, 10^{-4}, 10^{-3}\}$
Early stopping	Patience 5	Patience 5 / Tolerance 0.001
Learning rate η	$[0.001, 0.01]$	$\{0.001, 0.01\}$
Optimiser	ADAM	ADAM
Loss function	MSE	MSE
Ensemble	Average over 10	Average over 10

TABLE A.2: Mean hyperparameters are calculated over the ensemble of 10 networks and across all periods.

	With Interactions		W/O Interactions	
%	DNN	OES	DNN	OES
Mean L_1 penalty	0.0012	0.0154	0.0024	0.0028
Mean η	0.77	0.10	0.67	0.10

A4 Hyperparameters used in Chapter 4

In this section, we provide a list of hyperparameter search ranges and mean hyperparameters used to train the neural networks in Section 4.3.2. Hyperparameter search was performed on all combinations of hyperparameters. Table A.3 records common hyperparameters used in all models. Table A.4, A.5, A.6 and A.7 record hyperparameter search range and average hyperparameters of 10 networks for STAE & TCN, N-BEATS, LSTM and transformer, respectively.

TABLE A.3: Common hyperparameters for all networks used in Section 4.3.2. These are fixed values and are not subject to hyperparameter tuning. ADAM is the optimiser proposed by Kingma and Ba (2015).

Parameter	Common hyperparameters
Hidden layers	8
Activation	ReLU
Batch size	5,000
Batch normalisation	Yes
Early stopping	Patience 5 / Tolerance 0.0001
Learning rate η	0.01
Optimiser	ADAM

TABLE A.4: STAE and TCN specific hyperparameter ranges used in Section 4.3.2. Values enclosed by $\{\cdot\}$ are choices within the set. Values enclosed by $[\cdot]$ are a single list where each value indicates a layer. Mean hyperparameters are average chosen hyperparameters in an ensemble of 10.

Parameter	STAE	TCN
Search range		
Convolutional filters	$\{8, 16, 32\}$	$\{8, 16, 32\}$
Encoder kernel size	$\{2, 5, 10\}$	$\{2, 5, 10\}$
Decoder kernel size	$[2, 5, 5, 5]$	
Spatial dropout	$\{0.2, 0.4\}$	$\{0.2, 0.4\}$
Auxiliary loss ω	$\{0.2, 0.4\}$	
Mean hyperparameters		
Convolutional filters	27.2	18.4
Encoder kernel size	2.9	6.9
Spatial dropout	0.28	0.34
Auxiliary loss ω	0.34	

TABLE A.5: N-BEATS specific hyperparameter ranges used in Section 4.3.2. Values enclosed by $\{\cdot\}$ are choices within the set. Values enclosed by $[\cdot]$ are a single list where each value indicates a layer. Mean hyperparameters are average chosen hyperparameters in an ensemble of 10.

Parameter	N-BEATS
Search range	
Stacks	[Trend, Generic]
Blocks per stack	$\{2, 3, 4\}$
θ dimenions	$\{[4, 8], [8, 16], [16, 32]\}$
Mean hyperparameters	
Blocks per stack	2.6
Mean θ dimenions	13.8

TABLE A.6: LSTM specific hyperparameter ranges used in Section 4.3.2. Values enclosed by $\{\cdot\}$ are choices within the set. Values enclosed by $[\cdot]$ are a single list where each value indicates a layer. Mean hyperparameters are average chosen hyperparameters in an ensemble of 10.

Parameter	LSTM
Search range	
LSTM layers	$\{[8], [16], [32], [16, 8], [32, 16], [32, 16, 8]\}$
Mean hyperparameters	
Mean total LSTM units	32.8
Mean no. LSTM layers	1.7

TABLE A.7: Transformer specific hyperparameter ranges used in Section 4.3.2. Values enclosed by $\{\cdot\}$ are choices within the set. Values enclosed by $[\cdot]$ are a single list where each value indicates a layer. Mean hyperparameters are average chosen hyperparameters in an ensemble of 10.

Parameter	Transformer
Search range	
Key size	$\{4, 8\}$
No. heads	$\{1, 2\}$
No. transformer blocks	1
Dropout rate	$\{0.2, 0.4\}$
Mean hyperparameters	
Key size	$\{4, 8\}$
No. heads	$\{1, 2\}$
No. transformer blocks	1
Dropout rate	$\{0.2, 0.4\}$

A5 Hyperparameters used in Chapter 5

In this section, we provide a list of hyperparameter search ranges and mean hyperparameters used to train the neural networks in Section 5.4.1 and 5.4.2. Hyperparameter search was performed on all combinations of hyperparameters. Both cryptocurrency and U.S. equities datasets share the same hyperparameter ranges but with hyperparameter search performed separately. Table A.9 records mean hyperparameters over 10 network trainings for each network architecture.

TABLE A.8: Hyperparameter ranges used in Section 5.4.1 and 5.4.2. The 'LSTM layers' hyperparameter is a list, with the length of the list indicating how many LSTM layers were used and each element of the list indicating the number of units of each LSTM layer. Similarly, 'Hidden layers' indicate the number of fully connected hidden layers. Each element of the list indicate the dimension of that hidden layer. ADAM is the optimiser proposed by Kingma and Ba (2015).

Parameter	Search range
LSTM layers	$\{[16, 8], [32, 16, 8], [32, 16], [64, 32, 16]\}$
Hidden layers	$\{[8], [16, 8]\}$
Dropout rate	$\{0.2, 0.3, 0.4\}$
Activation	ReLU
Batch size	1,000
Batch normalisation	Yes
Early stopping	Patience 5 / Tolerance 0.0001
Learning rate η	0.01
Optimiser	ADAM

TABLE A.9: Mean hyperparameters are calculated over the ensemble of 10 networks for Ensemble, Evidential and Combined. Mean LSTM and hidden units is the average number of LSTM or hidden units in the network. 'no.' indicate number of layers.

%	Ensemble	Evidential	Combined
Mean dropout rate	33	26	25
Mean total LSTM units	56	41.6	61.6
Mean no. LSTM layers	2.5	2.4	2.5
Mean total hidden units	19.2	11.2	12.8
Mean no. hidden layers	1.7	1.2	1.3

A6 Marginal distribution of a Scale Mixture

From Equation (5.10), we have $\mathrm{N}(y|\gamma, \frac{\sigma^2}{\lambda})\,\mathrm{Gam}(\lambda|\alpha,\beta)$. Marginalising over λ produces the data likelihood,

$$\begin{aligned}
\mathrm{p}(y|\gamma,\sigma^2,\alpha,\beta) &= \int_\lambda^\infty \mathrm{p_N}(y|\gamma,\sigma^2\lambda^{-1})\mathrm{p_G}(\lambda|\alpha,\beta)\,\mathrm{d}\lambda \\
&= \int_\lambda^\infty \left[\sqrt{\frac{\lambda}{2\pi\sigma^2}}\exp\left\{-\frac{\lambda(y-\gamma)^2}{2\sigma^2}\right\}\right]\left[\frac{\beta^\alpha}{\Gamma(\alpha)}\lambda^{\alpha-1}\exp^{-\beta\lambda}\right]\mathrm{d}\lambda \\
&= \frac{\beta^\alpha}{\Gamma(\alpha)\sqrt{2\pi\sigma^2}}\int_{\lambda=0}^\infty \lambda^{\alpha-\frac{1}{2}}\exp\left\{-\frac{\lambda(y-\gamma)^2}{2\sigma^2}-\beta\lambda\right\}\mathrm{d}\lambda \\
&= \frac{\beta^\alpha}{\Gamma(\alpha)\sqrt{2\pi\sigma^2}}\left[\frac{(y-\gamma)^2}{2\sigma^2}+\beta\right]^{-(\alpha+\frac{1}{2})}\int_{\lambda=0}^\infty\left\{\lambda\left[\frac{(y-\gamma)^2}{2\sigma^2}+\beta\right]\right\}^{\alpha-\frac{1}{2}} \\
&\quad \exp\left\{-\lambda\left[\frac{(y-\gamma)^2}{2\sigma^2}+\beta\right]\right\}\mathrm{d}\left\{\lambda\left[\frac{(y-\gamma)^2}{2\sigma^2}+\beta\right]\right\},
\end{aligned}$$

since $\int_0^\infty x^{\alpha-1}\exp(-x)\,\mathrm{d}x = \Gamma(\alpha)$,

$$= \frac{\beta^\alpha}{\sqrt{2\pi\sigma^2}}\frac{\Gamma(\alpha+\frac{1}{2})}{\Gamma(\alpha)}\left[\frac{(y-\gamma)^2}{2\sigma^2}+\beta\right]^{-(\alpha+\frac{1}{2})}$$

and re-arranging $\beta^\alpha = (\frac{1}{\beta})^{-\alpha} = (\frac{1}{\beta})^{-(\alpha+\frac{1}{2})+\frac{1}{2}}$,

$$\begin{aligned}
&= \frac{\Gamma(\alpha+\frac{1}{2})}{\Gamma(\alpha)}\frac{1}{\sqrt{2\pi\sigma^2\beta}}\left[\frac{(y-\gamma)^2}{2\sigma^2\beta}+1\right]^{-(\alpha+\frac{1}{2})} \\
\mathrm{p}(y|\gamma,\sigma^2,\alpha,\beta) &= \mathrm{St}\left(y;\gamma,\frac{\sigma^2\beta}{\alpha},2\alpha\right). \qquad \text{(A.8)}
\end{aligned}$$

To show that the last step of Equation (A.8) is true, we start with the probability density function of the t-distribution parameterised in terms of precision $\mathrm{St}(y|\gamma,b^{-1},a)$ (Bishop, 2006),

$$\mathrm{St}(y|\gamma,b^{-1},a) = \frac{\Gamma(\frac{a+1}{2})}{\Gamma(\frac{a}{2})}\left[\frac{b}{\pi a}\right]^{\frac{1}{2}}\left[1+\frac{b(y-\gamma)^2}{a}\right]^{-(\frac{a+1}{2})},$$

where γ is location, b is inverse of scale and a is shape[15]. Substituting in $b^{-1} = \frac{\sigma^2\beta}{\alpha}$ and $a = 2\alpha$,

$$\begin{aligned}\mathrm{St}\left(y|\gamma, \frac{\sigma^2\beta}{\alpha}, 2\alpha\right) &= \frac{\Gamma(\alpha+\frac{1}{2})}{\Gamma(\alpha)}\left[\frac{(\frac{\alpha}{\sigma^2\beta})}{2\pi\alpha}\right]^{\frac{1}{2}}\left[1+\frac{\alpha(y-\gamma)^2}{2\sigma^2\alpha\beta}\right]^{-(\alpha+\frac{1}{2})} \\ &= \frac{\Gamma(\alpha+\frac{1}{2})}{\Gamma(\alpha)}\frac{1}{\sqrt{2\pi\sigma^2\beta}}\left[\frac{(y-\gamma)^2}{2\sigma^2\beta}+1\right]^{-(\alpha+\frac{1}{2})}.\end{aligned}$$

A7 Negative log-likelihood of marginal distribution of a Scale Mixture

From Equation (A.8), the NLL of the marginal t-distribution is,

$$\mathrm{p}(y|\gamma,\sigma^2,\alpha,\beta) = \frac{\Gamma(\alpha+\frac{1}{2})}{\Gamma(\alpha)}\frac{1}{\sqrt{2\pi\sigma^2\beta}}\left[\frac{(y-\gamma)^2}{2\sigma^2\beta}+1\right]^{-(\alpha+\frac{1}{2})}$$

$$-\log[\mathrm{p}(y|\gamma,\sigma^2,\alpha,\beta)] = \log\left[\frac{\Gamma(\alpha)}{\Gamma(\alpha+\frac{1}{2})}\right] + \tfrac{1}{2}\log[2\pi\sigma^2\beta] + (\alpha+\tfrac{1}{2})\log\left[\frac{(y-\gamma)^2}{2\sigma^2\beta}+1\right].$$

A7.1 Benchmarking on UCI dataset

In this section, we compare our method to Ensemble and Evidential using the UCI benchmark dataset. This is intended to provide readers with a direct comparison to Lakshminarayanan et al. (2017) and Amini et al. (2020) on the same dataset used in both works. The collection consists of nine real world regression problems, each with 10–20 features and hundreds to tens of thousands of observations. We note the *Wine* dataset within UCI contains discrete values (ratings of wine characteristics, such as color and taste) which may render the assumption of a continuous, symmetrical data distribution less appropriate if these values are skewed. More recently, larger uncertainty quantification datasets have been published in Dheur and Ben Taieb (2023), which may be useful in assessing the state-of-the-art in non-time-series uncertainty quantification methods. We follow Lakshminarayanan et al. (2017) and Amini

[15]Note that the definition of scale b and shape a is used exclusively in this section. Not to be confused with network bias b and activation vector $\boldsymbol{a}$ used in the rest of this thesis.

TABLE A.10: Comparing Ensemble (Lakshminarayanan et al., 2017), Evidential (Amini et al., 2020) and Combined (this work) on RMSE and NLL using the UCI benchmark datasets. Average result and standard deviation over 5 trials for each method. The best method for each dataset and metric are highlighted in **bold**.

	RMSE			NLL		
Dataset	**Ensemble**	**Evidential**	**Combined**	**Ensemble**	**Evidential**	**Combined**
Boston	$\mathbf{2.66 \pm 0.20}$	2.95 ± 0.29	2.89 ± 0.31	2.28 ± 0.05	2.30 ± 0.05	$\mathbf{2.23 \pm 0.05}$
Concrete	5.79 ± 0.16	5.98 ± 0.23	$\mathbf{5.40 \pm 0.18}$	3.07 ± 0.02	3.11 ± 0.04	$\mathbf{2.98 \pm 0.03}$
Energy	1.86 ± 0.04	1.84 ± 0.06	$\mathbf{1.71 \pm 0.20}$	1.36 ± 0.02	1.41 ± 0.04	$\mathbf{1.35 \pm 0.05}$
Kin8nm	$\mathbf{0.06 \pm 0.00}$	0.06 ± 0.00	0.06 ± 0.00	$\mathbf{-1.39 \pm 0.02}$	-1.28 ± 0.03	-1.35 ± 0.02
Naval	$\mathbf{0.00} \pm 0.00$	0.00 ± 0.00	0.00 ± 0.00	$\mathbf{-6.10 \pm 0.05}$	-5.99 ± 0.09	-5.89 ± 0.35
Power	3.02 ± 0.09	3.02 ± 0.08	$\mathbf{2.95 \pm 0.08}$	2.57 ± 0.01	2.56 ± 0.03	$\mathbf{2.53 \pm 0.02}$
Protein	3.71 ± 0.10	4.28 ± 0.23	$\mathbf{3.67 \pm 0.13}$	$\mathbf{2.61 \pm 0.03}$	2.73 ± 0.08	2.70 ± 0.05
Wine	0.60 ± 0.03	$\mathbf{0.56 \pm 0.02}$	0.59 ± 0.03	0.94 ± 0.04	$\mathbf{0.92 \pm 0.04}$	1.00 ± 0.03
Yacht	$\mathbf{1.22 \pm 0.22}$	1.48 ± 0.47	3.97 ± 1.06	1.06 ± 0.08	$\mathbf{0.96 \pm 0.19}$	1.17 ± 0.11

et al. (2020) in evaluating our method on root mean squared error (RMSE, which assesses forecast accuracy) and NLL (which assesses overall distributional fit), and compare against Ensemble and Evidential. While we do not explicitly compare inference speed, as our Combined method also uses ensembling, inference speed is expected to be comparable to Ensemble while being slower than Evidential. We use the source code provided by Amini et al. (2020), with the default topology of a single hidden layer with 50 units for both Ensemble and Evidential[16]. For Combined, as individual modelling of distribution parameters (Section 5.3.2) requires a network with two or more hidden layers, we have used a single hidden layer with 24 units, followed by 4 separate stacks of a single hidden layer with 6 units each. Thus, the total number of non-linear units is 48 (compared to 50 for Ensemble and Evidential). Note that even though the total number of units are similar across the three models, learning capacity may differ due to different topologies.

Table A.10 records experiment results on the UCI dataset. On RMSE, we find that both Ensemble and Combined have performed well, having the best RMSE in four datasets each. In two of the sets (*Kin8nm* and *Naval*), all three methods produced highly accurate results that are not separable to two decimal points. Turning to NLL, we observe a trend towards Combined having lower NLL than the other two methods for four sets, followed

[16]Source code for Amini et al. (2020) is available on Github: https://github.com/aamini/evidential-deep-learning

by Ensemble with three sets. Comparing Combined to Evidential, we find that Combined generally has lower RMSE (7 of 9 sets) and NLL (6 of 9 sets). Although our method is designed for uncertainty quantification of complex time-series and all 9 datasets are pooled (non-time-series) datasets, we still observe some improvements in both RMSE and NLL.

Next, we present further ablation studies on the UCI dataset. Table A.11 records results of *Alternative*, which utilizes ensembling and SMD parameterisation but not separate modelling of hyperparameters. Alternative has the same network topology as Ensemble and Evidential (a single hidden layer with 50 units), as opposed to Combined which has two hidden layers with a total of 48 units. We observe that Ensemble has the lowest RMSE in 5 (of 9) datasets, followed by Alternative (3 of 9), while Alternative has the best NLL in 6 (of 9) datasets and Ensemble has 3 (of 9). On both metrics, Evidential has the least favourable performance. Comparing Combined in Table A.10 and Alternative in Table A.11, Combined has lower RMSE and NLL in 5 of 9 datasets. Thus, we conclude that separate modelling of hyperparameters provided an incremental benefit on the UCI datasets.

TABLE A.11: Comparing Ensemble, Evidential and Alternative (without separate modelling of the four parameters of SMD) on RMSE and NLL using the UCI benchmark datasets. Average result and standard deviation over 5 trials for each method. The best method for each dataset and metric is highlighted in **bold**.

	RMSE			NLL		
Dataset	**Ensemble**	**Evidential**	**Alternative**	**Ensemble**	**Evidential**	**Alternative**
Boston	$\mathbf{2.66 \pm 0.20}$	2.95 ± 0.29	2.87 ± 0.18	$\mathbf{2.28 \pm 0.05}$	2.30 ± 0.05	2.29 ± 0.04
Concrete	5.79 ± 0.16	5.98 ± 0.23	$\mathbf{5.72 \pm 0.15}$	3.07 ± 0.02	3.11 ± 0.04	$\mathbf{3.03 \pm 0.02}$
Energy	1.86 ± 0.04	$\mathbf{1.84 \pm 0.06}$	1.88 ± 0.04	1.36 ± 0.02	1.41 ± 0.04	$\mathbf{1.35 \pm 0.03}$
Kin8nm	$\mathbf{0.06 \pm 0.00}$	0.06 ± 0.00	0.06 ± 0.00	$\mathbf{-1.39 \pm 0.02}$	-1.28 ± 0.03	-1.38 ± 0.02
Naval	$\mathbf{0.00 \pm 0.00}$	0.00 ± 0.00	0.00 ± 0.00	-6.10 ± 0.05	-5.99 ± 0.09	$\mathbf{-6.12 \pm 0.06}$
Power	3.02 ± 0.09	3.02 ± 0.08	$\mathbf{2.97 \pm 0.10}$	2.57 ± 0.01	2.56 ± 0.03	$\mathbf{2.54 \pm 0.02}$
Protein	$\mathbf{3.71 \pm 0.10}$	4.28 ± 0.23	3.75 ± 0.11	$\mathbf{2.61 \pm 0.03}$	2.73 ± 0.08	2.72 ± 0.02
Wine	0.60 ± 0.03	0.56 ± 0.02	$\mathbf{0.55 \pm 0.02}$	0.94 ± 0.04	0.92 ± 0.04	$\mathbf{0.92 \pm 0.02}$
Yacht	$\mathbf{1.22 \pm 0.22}$	1.48 ± 0.47	1.45 ± 0.33	1.06 ± 0.08	0.96 ± 0.19	$\mathbf{0.93 \pm 0.09}$

In Table A.12, we further remove model averaging. The network used is identical to Evidential but trained using the SMD parameterisation (i.e., we simply change the loss function in Evidential to Equation (5.12)). We observe that the network trained using the SMD parameterisation has lower RMSE in 6 of 9 and lower NLL in 8 out 9 datasets. We argue that the improved performance of the SMD parameterisation is due to its simplicity.

TABLE A.12: Comparing Normal-Inverse-Gamma and Normal-Gamma on RMSE and NLL using the UCI benchmark datasets. Average result and standard deviation over 5 trials for each method. The best method for each dataset and loss function is highlighted in **bold**.

	RMSE		NLL	
Dataset	**NIG**	**SMD**	**NIG**	**SMD**
Boston	$\mathbf{2.95 \pm 0.29}$	2.97 ± 0.20	$\mathbf{2.30 \pm 0.05}$	2.31 ± 0.05
Concrete	5.98 ± 0.23	$\mathbf{5.78 \pm 0.23}$	3.11 ± 0.04	$\mathbf{3.05 \pm 0.04}$
Energy	$\mathbf{1.84 \pm 0.06}$	1.87 ± 0.16	1.41 ± 0.04	$\mathbf{1.33 \pm 0.05}$
Kin8nm	0.06 ± 0.00	$\mathbf{0.06 \pm 0.00}$	-1.28 ± 0.03	$\mathbf{-1.37 \pm 0.01}$
Naval	0.00 ± 0.00	$\mathbf{0.00 \pm 0.00}$	-5.99 ± 0.09	$\mathbf{-6.27 \pm 0.09}$
Power	3.02 ± 0.08	$\mathbf{2.98 \pm 0.12}$	2.56 ± 0.03	$\mathbf{2.53 \pm 0.02}$
Protein	4.28 ± 0.23	$\mathbf{3.72 \pm 0.16}$	2.73 ± 0.08	$\mathbf{2.39 \pm 0.05}$
Wine	$\mathbf{0.56 \pm 0.02}$	0.56 ± 0.03	0.92 ± 0.04	$\mathbf{0.87 \pm 0.04}$
Yacht	1.48 ± 0.47	$\mathbf{1.44 \pm 0.49}$	0.96 ± 0.19	$\mathbf{0.91 \pm 0.18}$

A8 Further analysis of parameters in a Scale Mixture

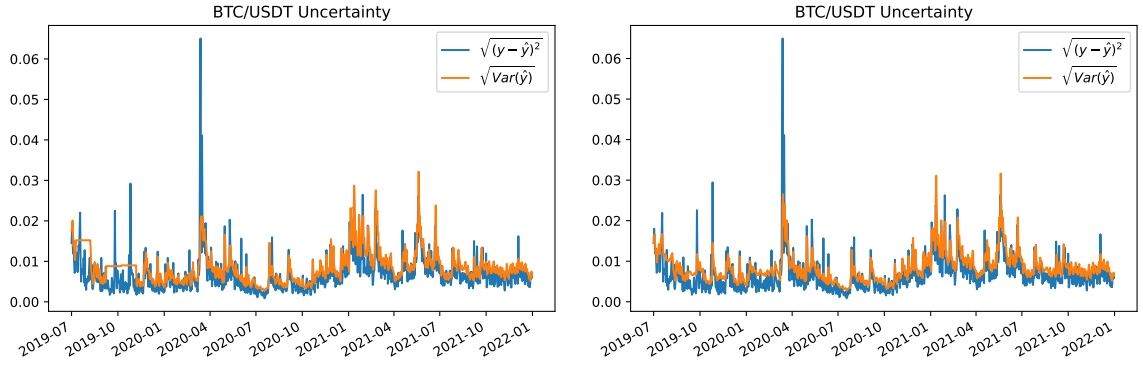
In the network architecture proposed in Section 5.3, output of the network is $\boldsymbol{\zeta} = (\gamma, \sigma^2, \alpha, \beta)$, which parameterises the SMD (Equation (5.10)). However, as noted in Section 5.3, we can set $\alpha = \beta$ and reduce the number of parameters to three (Equation (5.14)). Thus, an alternative specification of the network is to output $\boldsymbol{\zeta} = (\gamma, \sigma^2, \alpha)$ (i.e., three parameters instead of four and are computed through three subnetworks, instead of four in Figure 5.1). We label this network *A=B*. In Table A.13, we compare Combined (4 parameters) with S2B (3 parameters) using the UCI dataset (as introduced in Section A7.1). We observe that A=B is better than Combined on 8 (of 9) datasets on RMSE, while Combined is better than A=B on 1 (of 9). On NLL, A=B is better than Combined on 5 (of 9) datasets, while Combined is better than A=B on 4 (of 9). Even though A=B has a higher number of datasets with lower RMSE and NLL, we note that the differences are very small and are within margin of error (due to randomness in neural network training). Thus, we conclude that the two methods provide near identical results but note that A=B is simpler and more interpretable. However, we choose Combined with four subnetworks to conduct our analysis so that parameters can also be compared with those from Evidential and Extended Evidential in Appendix A9.

TABLE A.13: Comparing A=B (3 parameters) to Combined (4 parameters) on RMSE and NLL using the UCI benchmark datasets. Results are averaged over 5 trials and the best method for each dataset and metric are highlighted in **bold**.

	RMSE		NLL	
Dataset	A=B	Combined	A=B	Combined
Boston	2.91 ± 0.17	$\mathbf{2.89 \pm 0.31}$	2.27 ± 0.04	$\mathbf{2.23 \pm 0.05}$
Concrete	$\mathbf{5.39 \pm 0.19}$	5.40 ± 0.18	2.99 ± 0.03	$\mathbf{2.98 \pm 0.03}$
Energy	$\mathbf{1.56 \pm 0.16}$	1.71 ± 0.20	$\mathbf{1.30 \pm 0.05}$	1.35 ± 0.05
Kin8nm	$\mathbf{0.06 \pm 0.00}$	0.06 ± 0.00	$\mathbf{-1.36 \pm 0.02}$	-1.35 ± 0.02
Naval	$\mathbf{0.00 \pm 0.00}$	0.00 ± 0.00	-5.87 ± 0.12	$\mathbf{-5.89 \pm 0.35}$
Power	$\mathbf{2.93 \pm 0.08}$	2.95 ± 0.08	$\mathbf{2.53 \pm 0.02}$	2.53 ± 0.02
Protein	$\mathbf{3.60 \pm 0.10}$	3.67 ± 0.13	2.83 ± 0.04	$\mathbf{2.70 \pm 0.05}$
Wine	$\mathbf{0.57 \pm 0.02}$	0.59 ± 0.03	$\mathbf{0.96 \pm 0.03}$	1.00 ± 0.03
Yacht	$\mathbf{2.31 \pm 0.43}$	3.97 ± 1.06	$\mathbf{1.11 \pm 0.09}$	1.17 ± 0.11

A9 Further analysis of Evidential on uncertainty quantification in cryptocurencies

The Evidential method utilises NormalInverseGamma output layer (no separate subnetworks for distribution hyperparameters) and a t-distributed NLL derived from the NIG distribution (Equation (5.9)). In Section 5.4.1, we have observed that Evidential fails to provide uncertainty forecasts that track time-varying volatility. However, this was not observed in our proposed Combined method. In here, we test the effects of separate modelling of distribution hyperparameters in the output layer for Evidential. We label this as the *Extended Evidential* method. Square-root of average squared forecast error and square-root of forecast uncertainty for BTC/USDT for Extended Evidential and Combined are shown in Figure A.3. We find that separate modelling of distribution hyperparameters significantly improved accuracy of uncertainty forecasts of Extended Evidential. Forecast uncertainty of both Extended Evidential and Combined are generally similar, with the exception of still some block-like features for Extended Evidential in 2019. Comparing cryptocurrency experimental results of Extended Evidential to Combined, we find that Combined is better than Extended Evidential on both IC and NLL, and is better than Extended Evidential on RMSE at higher decimal places.



(a) Extended Evidential (b) Combined

FIGURE A.3: Actual prediction error and predicted uncertainty of Extended Evidential and Combined for BTC/USDT. Square root of the average forecast error and forecast uncertainty on each day shown.

TABLE A.14: Ablation study comparing Extended Evidential to Combined on average IC, RMSE and NLL for cryptocurrencies time-series forecasts. Average result and standard deviation over 10 trials for each method. Best method for each dataset is highlighted in **bold**.

Metric	Extended Evidential	Combined
IC (%)	6.39 ± 2.56	$\mathbf{9.87 \pm 3.17}$
RMSE (%)	0.867 ± 0.001	$\mathbf{0.867 \pm 0.001}$
NLL	-3.35 ± 0.01	$\mathbf{-4.14 \pm 0.01}$

EXAMINER 1

Comment

This thesis focuses on the innovation and applications of deep learning (DL) techniques to address open problems associated with forecasting tasks in financial markets, e.g., stock return prediction, in the context of quantitative portfolio management, with a particular attention on solving the issues of the time-varying data generation process (DGP), low signal-to-noise ratio, heavy tail, and volatility clustering. There are three chapters (Chapters 3-5) dedicated to relevant contributions, which correspond to two published papers and one manuscript in preparation. Overall, this thesis is well-structured, easy-to-follow, and makes valuable impacts on the applications of machine learning (ML) techniques to financial applications. Some detailed comments and suggestions are given below.

Response

Thank you very much for your positive comments and your time in reviewing my thesis. Please find below my responses to your comments.

Chapter 1

Comment: In **Chapter 1**, Eq. (1.3) (1.4 in updated thesis) and (1.4) (1.6 in updated thesis) should be explained in a clearer way. It is difficult from the current description to understand how to obtain input X and define function F with respect to risk and cost.

Response: In p.10, we have added a description of the cross-sectional regression problem. At time t, we regress stock returns observed at t on features observed at $t-1$ for the cross-section of N stocks. This is known as a cross-sectional regression. Eq. (1.4) is an abstraction of the risk model, which can be machine learning based. We have provided an example of a linear regression-based risk model on p.11 to illustrate how risk is estimated by practitioners. Cross-sectional regressions are repeated for every cross-section in $1, \dots, t-1$, resulting in M time-series of cross-sectional regression coefficients $\{\hat{\xi}_1, \dots, \hat{\xi}_M\}$ (known as *factor returns*), where M is the number of features and each time-series is $t-1$ long. Factor risk is estimated by computing the variance-covariance matrix ($\boldsymbol{V}'_t \in R^{M\times M}$) of the factor returns. $\boldsymbol{V}'_t$ measures risk at the factor level and is converted back to stock-level variance-covariance by: $\widehat{\boldsymbol{V}}_t = \boldsymbol{X}_t \boldsymbol{V}'_t \boldsymbol{X}_t^T$, where $\boldsymbol{X}_t \in R^{N\times M}$ is the matrix of stock-level features used in the cross-sectional regressions. Note that features used in the return forecasting model (Eq. 1.2) can also be used to estimate risk. More details on this are provided in p.11. Similarly, Eq. (1.6) is an abstraction of the transaction cost model, which can be based on machine learning methods. We have added the transaction cost model proposed in Grinold & Khan (1999) (Eq. 1.7, end of p.11 to beginning of p.12) as an example of transaction cost model. In Eq. (1.7), expected transaction cost is assumed to consist of three components: 1) a percentage commission payable to brokers (e.g., banks); 2) bid-ask spread (e.g., if the best bid is \$1.00 and best ask is \$1.01, rather than waiting in the bid queue and pay \$1.00/sh, a buy order can immediately execute if it is willing to pay the asking price of \$1.01); 3) a market impact component (introduced in Section 1.2 and further explained in Appendix A2.3). Market impact relates to buying (selling) pressure that causes the price to move up (down). In Grinold & Khan (1999), this is assumed to be proportional to the square-root of the ratio between value of the trade and daily average traded value in the stock. To illustrate, suppose that the average daily traded value of a stock is \$1million. A trade worth

Comment: The same applies to the explanation of Eq. (1.5) (1.8 in updated thesis), where the meanings of each term should be better explained. For example, it is difficult to understand the meaning of the 3rd term, based on the current description, for readers like me who are not familiar with mean-variance optimisation.

Response: We have added a discussion on mean-variance optimisation (Eq. 1.8, p.12-14), including an example to explain the intuition of mean-variance optimisation, and an explanation of the three terms in Eq. 1.8. Specifically, Eq. 1.8 is a generalisation of various portfolio optimisation objectives. This is controlled by the risk aversion parameter λ, which serves to control the trade-off between maximising return (by setting $\lambda = 0$) and minimising risk (by setting $\lambda \to \infty$). Let's first consider the case of $\lambda = 0$ and $\hat{c}_t = 0$ (i.e., no transaction cost). The risk penalty (2nd term in Eq. 1.8) becomes 0. Thus, the optimiser maximises return by placing 100% of the portfolio into the stock with the highest return. Next, consider $\lambda = 1000$, a large number that dominates the 1st term of Eq. 1.8 such that the optimiser focuses on minimising risk. In this case, the optimiser will place 100% of the portfolio into the stock with the lowest risk. A value in between (e.g., $\lambda = 5$) will simultaneously maximise return and minimise risk, where the risk term acts like a regulariser. In practice, λ is subjectively chosen by the investor based on their risk tolerance. The cost term comes into play when $t > 1$ and the portfolio must transition from existing weights to new optimal weights (e.g., by selling one stock and buying another). Transaction costs discourage the optimiser from deviating from existing weights, as it is costly to do so. Any deviation must be offset by an increase in expected return that more than compensates for transaction cost. This in turn acts as a regulariser that reduces the difference between the existing portfolio and the new portfolio (as it is costly to change).

Comment: In pages 16-17, it would be more helpful to provide a categorised summary of the factors that may influence a stock's price. In Section 1.6.1, there lacks a clear (mathematical) definition of cross-section prediction tasks, which influences the understanding of the subsequent relevant contents.

Response: We have summarised the discussion on drivers of stock returns in Section 1.5 into a list (beginning of p.20). Namely, these include price-derived features (e.g., past returns, known as the momentum effect), financial statement-derived features (e.g., valuation metrics), social media activity, web searches and media reports. In Chapter 3, we have used the same dataset as Gu et al. (2020) which mainly contains price and financial statement features. We have added a comment on this in the updated thesis. As noted in our first response, a formal definition of cross-section prediction has been added in Section 1.3 (p.10).

Comment: In page 19 (p.21 in updated thesis), it is better to mention (if possible) whether the features (factors) used by those prediction models in comparison are same or different.

Response: We've added a comment (p.21) on the respective datasets. Gu et al. (2020) is the largest, containing 94 firm-level features. Messmer (2017) contains 61 which are all contained within Gu et al. (2020). Abe & Nakayama (2018) contains the least, at 25. However, due to different naming conventions, we were not able to ascertain how many of Abe & Nakayama (2018) are contained within Gu et al. (2020).

Comment: In Section 1.6.2, more approaches suitable for noisy environments, instead of basic time-series prediction methods, should be included.

Response: In Section 1.6.2 (p.23), we have shortened the discussion on time-series prediction in financial markets and added a discussion (p.23) on robustness of neural networks in presence of noise. To the best of our knowledge, there is no prior work on testing the robustness of neural networks in financial markets. Drawing on findings in other applications, Rolnick et al. (2018) find that effective batch size decreases as the level of white noise increases (in image recognition applications). Thus, noisy environments require larger batch sizes. We have expanded the discussion on regularisation techniques for neural networks (p.24). In general, ways to combat noise are using more training data, L1 and L2 penalties, early-stopping and dropout. More recently, multi-task learning (MTL) has been shown to improve generalisation of neural networks. MTL involves the addition of an auxiliary learning

task that is related to the primary learning task and is thought to encourage representation sharing between the primary and auxiliary tasks. In Chapter 4, we introduce the Supervised Temporal Autoencoder, where the auxiliary task is to reconstruction the original input sequence and showed that this improved generalisation of the neural network in out-of-sample data. Towards the end of Section 1.6.2 (p.25), we have also discussed ways of combating noise in financial time-series, including treating the problem as a classification task (which mitigates the heavy tails of asset returns) and applying wavelet transforms, both of which are deficient.

Comment: In Section 1.6.3, it would be helpful to provide more details on how the forecasted uncertainty may be utilised to facilitate portfolio management.

Response: At the end of Section 1.6.3 (p.26), we have added a discussion on how forecast uncertainty can be used in portfolio optimisation (made consistent with the potential applications in the conclusion of Chapter 5). Specifically, forecast uncertainty can be used to size bets, or as advanced warning to protect the portfolio from increasing risk. For example, if forecast uncertainty reaches a certain threshold, an investor could purchase portfolio insurance (e.g., put options which allow the investor to sell stocks to the issuer of the options at a pre-agreed price) or liquidate positions to reduce risk.

Comment: In page 26 (p.30 in updated thesis), the term "sequential neural networks" is confusing.

Response: We have also added a footnote (p.30) to clarify the meaning of "sequential neural network". In this context, we are referring to network architectures that are applicable for time-series applications, such as recurrent neural network (RNN), LSTM and CNN.

Chapter 2

Comment: In **Chapter 2**, it is mentioned in pages 45-46 (p.52 in updated thesis) that f (F) and g (G) denote input and kernel, respectively, which, I think, should be opposite. Please have a check.

Response: In Chapter 2, we have cross-checked against the source (Goodfellow, et al., 2016) and confirm that F (input) and G (kernel) are correct designations in p.52. Goodfellow et al. (2016) used x as input and w as kernel, which carries a different meaning in this thesis.

Comment: The core technique used in Chapter 4, i.e., temporal convolutional networks (TCN) should be explained in more detail. The current description in page 49 is overly simple.

Response: We have expanded the explanation of causal convolutions (p. 53-54) and residual block (p. 54-55). On causal convolutions, we have added Fig. 2.8 (p.54) to demonstrate the mechanics causal padding. In normal convolution, the kernel convolves with values to both sides of the centre of the kernel. However, in a time-series, if we take the centre of the kernel as t, then the right of the centre of the kernel is $t + 1$. Thus, the kernel will be looking ahead in time. In causal convolution, $d \times (k - 1)$ number of zeros is padding to the left of the sequence (where d is dilation rate and k is kernel size). This causes the sequence to shift to the right and thus, the kernel no longer looks ahead in time (as depicted in Fig. 2.8). On residual block, we have added Fig. 2.10 to demonstrate the architecture of each block. In essence, each residual block is comprised of a dilated causal convolution layer, followed by batch normalisation, dropout and ReLU. A skip connection is added to each residual block, such that the input into each residual block is the sum of both input and output of the preceding block. This is motivated by ResNet (He et al., 2016), which showed that this construction allows the network to efficiently pass through the identity (i.e., the original sequence) and each residual block learns modifications to the original sequence.

Chapter 3

Comment: In **Chapter 3**, the motivation behind the proposed online early stopping (OES) strategy is reasonable. However, although early stopping may prevent overfitting (and thus help adapt to the time-varying DGP), it may also possibly lead to underfitting (if the #epochs is insufficient). There lacks explanation/discussion in this regard.

Response: In Chapter 3, we have added a discussion on the best- and worst-case scenarios for the OES algorithm (from the last paragraph of p.71-72). Specifically, the best case scenario is if the DGP is stationary and $\tau = 0$. The worst case scenario has already been discussed at the beginning of Section 3.3.1 (first paragraph of p.68) and has been expanded. It is the case where $\theta_0 = 0$ and the sequence of optimal theta alternates between $\{1, -1, 1, -1, \dots\}$. In this case, estimated steps $\tau = 0$ (as it is best to never update network weights) and regret scales linearly with time and average regret converges to a constant. This means that the network is always underfitting the data. However, as discussed in Section 3.2.3 (beginning of p.67), regret for *non-convex* online optimisation problems is measured in terms of sum of gradients, which is why our worst-case regret scales linearly with T in the worst case. For the problem to be tractable, regret for *convex* problems is typically measured against a benchmark. For example, suppose that the user is asked to come up with an algorithm to compute the mean of a streaming sequence of numbers. Suppose that the user proposed the *running average* algorithm:

$$u_t = \frac{\big((t-1)\, u_{t-1} + x_t\big)}{t},$$

where x_t is observation at t and u_t is the estimated mean at t, $u_0 = 0$, and for all $0 < t \leq T$. The "best minimiser in hindsight" in this scenario would be the average after T observations are made. The mean computed by this algorithm does in fact converge to the benchmark as $t \to \infty$. Thus, the average regret converges to 0 (we have added an introduction to average regret in the beginning of p.67). With this example in mind, in our worst case, our algorithm also converges to the best hindsight minimiser. This provides users with a worst-case performance guarantee, that the algorithm will perform as well as if all the data is made available beforehand and a single neural network is trained on the entire dataset. We consider this a strong performance guarantee and is one that is useful to practitioners.

Comment: The expanding window approach, as a key compared method, should better be explained in more detail. Also, I am curious whether there exist other techniques except the only one selected to compare in this work, i.e., Gu et al. 2020 (named as DNN in page 56 (p.61 in updated thesis) – which is a bit confusing naming), which have been proposed to address the same/similar issue.

Response: We have added explanations for the expanding window approach on p.60. In essence, for every year, the network is trained using all training data. The same network is then used for monthly prediction over the next 12 months, upon which an additional 12 months of data is added to the training set and training is repeated. We have replaced the label "DNN" for Gu et al. (2020) with "EWNN" for expanding window neural network, to reflect the fact that it was trained using an expanding window approach (see Table 3.1, p.75 and Table 3.3 in p.79).

Comment: Comparison to a single method to validate the superiority of the proposed method is a bit risky in terms of the reliability of conclusions.

Response: On alternative benchmarks for time-varying problems, we have discussed the DTS-SGD algorithm proposed by Aydore et al. (2019) in Section 3.2.3 (p.66-67). To our knowledge, this is the state-of-the-art online learning algorithm for non-convex problems and was used for neural network training in Aydore et al. (2019). In Section 3.4, we compared OES against EWNN and DTS-SGD on a

synthetic dataset in Table 3.1 (p.75), and demonstrated superior performance. We have not compared against DTS-SGD on the U.S. equities dataset due to exploding gradient. This is discussed at the beginning of Section 3.5.2 (p.78).

Comment: The meanings of decile 10 and decile 1 in Eq. 3.4 lack a clear explanation. It is difficult to understand the sentence "*A pooled regression with window size w effectively assumes data at t +1 is drawn from the average of the past w observations*" in page 59 (p.65 in updated thesis).

Response: We have added further explanations on "decile return spread" on p.64. Decile return spread is computed as following. For every month, we sort stocks based on the return forecast $\hat{\boldsymbol{y}}_t$. We then place stocks into 10 equally sized deciles and compute average realised return of each decile in $t+1$. The decile return spread is the difference in average return of decile 10 (top decile) and decile 1 (bottom decile). Formally, definition of decile return spread is defined in Eq. 3.4 (p.64). We have reworded the sentence "A pooled regression…" to "A regression that is fitted on a sliding window of size w effectively assumes that data at $t+1$ is drawn from the average DGP of the past $t-w, \dots, t$ cross-sections" on p.65.

Chapter 4

Comment: In **Chapter 4**, the proposed technique is a reasonable idea which has been successfully applied in other applications. However, it is unclear why the input sequence length has been chosen to be that long, i.e, K = 250. Any justifications?

Response: In Chapter 4, we assume $K = 250$, as there are approximately 252 business days per year and is motivated by the momentum effect documented in finance literature (one-year change in price predicts next month's return). We have added commentary to reflect this in Section 4.2.1 (last paragraph of p.96). In the original work of Jegadeesh & Titman (1993), the momentum effect was observed in all 3-, 6-, 9- and 12-month look back windows. Thus, we do not expect the choice of K to have a material impact on the efficacy of our time-series predictions. In empirical results (Table 4.1 in p.108), we compare against 12-month momentum (MOM12, the standard definition of the momentum effect in finance literature) and show that a pattern recognition approach using neural networks (STAE) provides significantly better performance.

Comment: Also, it seems that the OES strategy proposed in Chapter 3 is not applied herein, and instead a basic early stopping method is employed. Any reasons?

Response: In this chapter, we focus solely on advancing techniques for training neural networks in noisy environments as we believe that our contribution (regularising a neural network using a supervised autoencoder) has general applicability in other noisy learning environments. We have noted this in the conclusion of Chapter 4 (Section 4.4, p.117) to highlight this and have added a discussion on combining OES, STAE and uncertainty quantification (contribution of Chapter 5) into a unified framework in the Future Research section (Section 6.2, p.154-155) of this thesis. This is further discussed in our response to comments on Chapter 6.

Comment: The results shown in Table 4.1 among some other tables do not demonstrate consistent superiority of the proposed method. Some discussion should be provided around this.

Response: In Table 4.1 (p.108), our primary performance measure is information coefficient (IC, Eq. 3.2, p.64, computed by taking the mean of cross-sectional correlation across time and is a frequently used performance measure in the finance industry (introduced in Section 3.2.1, p.64). On this metric (Mean IC and Ens. IC in Table 4.1), STAE is superior to all other compared methods. On MSE (Mean MSE and Ens. MSE), TCN has the lowest MSE but they are virtually indistinguishable. We have added additional commentary (reproduced below) on correlation and decile returns on p.109 to highlight

the fact that decile returns are based solely on the top and bottom 10% of the cross-section, which ignores the forecast accuracy in the middle 80% of the distribution. By contrast, correlation is based on the entire cross-section and is a more holistic measure of performance. "*For practitioners, IC, decile returns and Sharpe ratio are important performance metrics (Sharpe ratio and decile returns are related as Sharpe ratios are derived from decile returns, and are used by Gu et al., 2020). However, decile returns focus on top and bottom 10% of forecasts without accounting for the middle 80% of the distribution of forecasts. There are investment strategies that rely on less extreme return forecasts. In general, IC provides a more complete pictures of prediction performance by incorporating all forecasts. The higher IC of STAE reflects better ranking of stocks across the whole distribution, despite having similar decile returns.*"

Chapter 5

Comment: In **Chapter 5**, the uncertainty of the prediction outcome is modelled to follow some distribution which is learnt (in terms of its parameters) together with the prediction outcome itself via neural networks. I suggest that the architectural description in Section 5.3.2 and Fig. 5.1 should depict how the overall training loss is constructed.

Response: We have modified Fig 5.1 (p.134) and first paragraph of Section 5.3.2 (p.134) to show that the four hyperparameters outputted by the network are fed into the SMD marginal NLL (Eq. 5.12, p.132) during training.

Comment: Further, I am curious whether the prediction outcome' distribution assumption can be guaranteed, i.e., what if the actual distribution deviates much from that assumed and formulated in this work.

Response: We can measure the difference between the predicted distribution and empirical distribution of y via negative log-likelihood (used in this work) or KL-divergence. However, these are not "guarantees". Normality of returns is a common assumption when modelling returns in finance literature. However, asset returns are known to exhibit heavy tails (Cont, 2001). Our SMD formulation effectively fits a t-distribution on the data which is a more appropriate data distributional assumption for asset returns given the heavy tails. We have shown that the predictive uncertainty of our proposed Combined method can closely track actual forecast error in real world financial datasets (Fig. 5.2, p.141). Thus, demonstrating its potential usefulness in return forecasting. In the case where the actual distribution is very different from Normal or t-distribution (e.g., bimodal), then we expect predictive uncertainties to be high. Furthermore, they will not closely track actual forecast errors, unlike what we have observed with our method in Fig. 5.2. However, there is no test for t-distributed residuals currently with varying degrees of freedom.

Comment: Also, it would be much helpful to explain how to make use of the predicted uncertainty to help the downstream portfolio optimisation.

Response: We have added a discussion in the introduction (Section 5.1, end of p.119 to first paragraph of p.120) on potential applications of predictive uncertainty. Further potential applications are also provided in the conclusion (Section 5.5, end of p.147). We have reproduced the potential applications here for reference: 1) in Kelly criterion (computed as $y_t/Var[y_t]$, where $Var[y_t]$ is predictive uncertainty) to determine optimal bet size; 2) as "early warning", where if forecast uncertainty of the portfolio reaches a certain level, the investor can liquidate the portfolio to reduce risk or purchase portfolio insurance (e.g., put options).

Comment: In **Chapter 6**, the authors may consider a discussion on how to unify the three works proposed in Chapter 4-6. Currently, they are independent, but intrinsically could be combined.

Response: In Chapter 6 (p.154-155), we have added a discussion on how the three methods discussed in this thesis can be combined into a single method for forecasting stock returns. We propose to combine all three methods into a single network (consisting of 8 subnetworks), using autoencoders to process both firm features and return sequence, and output three parameters ($\alpha = \beta, \sigma^2, \gamma$) of the SMD distribution . We envisage three potential challenges with this approach, being size of the network, sufficiency of data in the OES algorithm and weight given to the reconstruction tasks in the loss function. We also suggest some potential ways of solving these problems. First, on training the proposed network. We propose to utilise transfer learning, a popular technique in training *large language models* (LLMs), where a component of the network is pre-trained using simpler (but related) tasks. We propose to pre-train Subnetwork 1 & 2 (processes firm features and stock returns) using autoencoders, where the subnetworks learn to encode and decode firm features and stock returns. Pre-training using autoencoders can be considered as a way of initialising network weights, such that they can be fine-tuned for the prediction task. After pre-training, the encoder subnetwork learns to encode the input (e.g., firm features and return series) into latent representations that summarises the input. We have shown in Chapter 4 (using STAE) that this regularises the network and improves prediction performance. Second, on sufficiency of data in the OES algorithm, one may find that a single cross-section contains insufficient data to train such a large network. To solve this, we may expand the size of look back window. However, as the size of the look back window increases, the ability of the network to track the time varying DGP decreases, until it converges to the expanding window approach. Size of the look back window may be found using hyperparameter search. Third, in our proposed network architecture, there are two reconstruction tasks (compared to one in STAE). Scales of the reconstruction loss of these autoencoders are likely different to the main prediction task and require further hyperparameter search. More details on this are given in p.154-155.

EXAMINER 2

Comment

The Author attribution statement declares that the thesis is based on 3 published papers. However, it seems that there are only two published papers while the last one is a working paper. Thus, a clarification on this is required. If the last paper is published, the published version should be cited in the thesis. Otherwise, the statement has to be revised.

I think that the candidate may have to comment for the computational burden in the optimization of (3.6) as it may add computational burden to the overall algorithm.

Response

Thank you very much for your positive comments and your time in reviewing my thesis.

We have amended the attribution statement to reflect the two published papers. Computational speed is discussed in the conclusion (Section 3.6, p.89) and is one of the advantages of this algorithm. Our method took 44.25mins for a single pass over the data compared to the method used in Gu et al. (2020) of 5.5 hours. This is because the training dataset is much smaller for our algorithm, which only includes data of the last 2 periods (complexity of OES is $O(T)$), compared to Gu et al. (2020) which is an expanding window of all past data (complexity is $O(T^2)$). The machine used is an AMD Ryzen 7 3700X, running Python 3.7.3, Tensorflow 1.12.0 and Keras 2.2.4. Due to the small size of the feedforward network used in Chapter 3, we have found training on the CPU to be faster than the GPU (Nvidia Geforce RTX 3060).

Machine Learning in Portfolio Management

STEVEN Y. K. WONG

BE (Hons), BCom, MFin

Supervisor: A/Prof. Jennifer S. K. Chan
Associate Supervisor: Dr. Lamiae Azizi

A thesis submitted in fulfilment of
the requirements for the degree of
Doctor of Philosophy

School of Mathematics and Statistics
Faculty of Science
The University of Sydney
Australia

19 September 2023

Abstract

Financial markets are difficult learning environments. The data generation process is time-varying, returns exhibit heavy tails and signal-to-noise ratio tends to be low. These contribute to the challenge of applying sophisticated, high capacity learning models in financial markets. Driven by recent advances of deep learning in other fields, we focus on applying deep learning in a portfolio management context. This thesis contains three distinct but related contributions to literature. First, we consider the problem of neural network training in a time-varying context. Conventional batch training methods assume a stationary data generation process, where training and out-of-sample data are assumed to be drawn from the same distribution. This is suboptimal for applications where the data generation process changes over time, such as in financial markets. To address this, we extend the *early stopping* algorithm into the online context, which we term the Online Early Stopping algorithm. We show that a neural network trained using this algorithm can track a function changing with unknown dynamics. We provide a *regret-bound* for the algorithm and show that the worst-case tracking performance of the algorithm is bound by the time-variance of the data generation process. We compare the proposed algorithm to current approaches in predicting monthly U.S. stock returns and show its superiority. Second, we consider the problem of learning in noisy environments. Noisy learning environments such as financial markets are characterised by low noise-to-signal ratio. This differs to information-rich applications such as image recognition. We propose an approach that regularises the temporal convolutional network using a supervised autoencoder, which we term the Supervised Temporal Autoencoder. We show that the addition of the auxiliary reconstruction task is beneficial to the primary supervised learning task in the context of stock return time-series forecasting. The supervised autoencoder denoises the input and encourages the main network to retain features that are beneficial to both prediction and reconstruction tasks. We also show that the supervised temporal autoencoder is able to learn features directly from the transformed price series, alleviating the need for

handcrafted features. The autoencoder also improves interpretability as users can observe the output of the decoder and inspect features retained by the network. Third, we consider the problem of quantifying forecast uncertainty in time-series with complex structures. Time-varying variance, such as volatility clustering as seen in financial time-series, can lead to large mismatch between predicted uncertainty and realised forecast error. We propose a novel framework to deal with uncertainty quantification under the presence of volatility clustering, building and extending the recent methodological advances in uncertainty quantification for non-time-series data. We outline several methodological advancements, including the use of scale mixture distribution and separate modelling of distribution hyperparameters. To illustrate the performance of our proposed approach, we apply it onto cryptocurrency and U.S. equities time-series forecasting for the designed use-case. We demonstrate superior performance to the current state-of-the-art in both data sets. We further provide an evaluation using a non-time-series benchmark data set (Appendix) to show the general applicability of our framework. Finally, potential future research directions in advancing machine learning in portfolio management is discussed.

Acknowledgements

After 5 arduous years, here I am, putting the final touches to my thesis. Looking back, I am glad that I have spent 5 years of my life (albeit part time) to learn something new about machine learning, and to give back my knowledge to science, however trivial my contributions may be. I would like to start by thanking my supervisors, A/Prof. Jennifer Chan and Dr. Lamiae Azizi, for whom I am forever grateful to have been mentored by. Without their patience, support and knowledge, I would not have made it this far. I would like to thank Prof. Richard Xu for initially accepting me into his PhD cohort. Even though we had to part ways, his machine learning classes were immensely helpful to my PhD. A special thanks to Prof. Maurice Pagnucco for supervising my Honours and leading the UNSW RoboCup team — an unforgetable journey through artificial intelligence that I still cherish today. I would also like to thank the many past and present colleagues who have shaped my understanding of quantitative investing. All of these experiences culminated in this thesis.

Finally, I would like to thank my family for their support and patience. They are the reason for my perseverance.

Contents

List of Figures

List of Tables

Acronyms

APT: Arbitrage Pricing Theory

ARCH: Autoregressive Conditional Heteroskedasticity

ARMA: autoregressive-moving-average

BNN: Bayesian neural network

CAPM: Capital Asset Pricing Model

CNN: convolutional neural network

CRSP: Center for Research in Security Prices

DGP: data generation process

DTS-SGD: Dynamic Exponentially Time-Smoothed Stochastic Gradient Descent

ELU: exponential linear unit

EWNN: expanding window neural network

GARCH: Generalised Autoregressive Conditional Heteroskedasticity

IC: information coefficient

IG: Inverse-Normal

KL divergence: Kullback-Leibler divergence

LSTM: long short-term memory

MCMC: Monte Carlo Markov Chain

MLP: multilayer perceptrons

MOM1: reversal effect

MOM12: momentum effect

MPT: Modern Portfolio Theory

MSE: mean squared error

MTL: multi-task learning

N-BEATS: Neural Basis Expansion Analysis for interpretable Time Series

NG: Normal-Gamma

NIG: Normal-Inverse-Gamma

NLL: negative log-likelihood

NLP: natural language processing

OES: Online Early Stopping

OLS: ordinary least squares

PCA: principal component analysis

PTVII: Pearson Type VII

ReLU: rectified linear unit

RMSE: root mean squared error

RNN: recurrent neural network

SAE: supervised autoencoder

SGD: stochastic gradient descent

SIC: Standard Industrial Classification code

SMD: scale mixture distribution

SML: Security Market Line

STAE: Supervised Temporal Autoencoder

SVM: Support Vector Machine

TCN: temporal convolutional network

TRI: total return index

UCI: University of California Irvine Machine Learning Repository

UQ: uncertainty quantification

List of Notations

Symbol	Description
$\boldsymbol{X}$	Uppercase bold font denotes matrix
$\boldsymbol{x}$	Lowercase bold font denotes vector
$\boldsymbol{x}_i$	i-th row of $\boldsymbol{X}$
$x_{i,j}$	Element in the i-th row and j-th column of $\boldsymbol{X}$
x_i	i-th element of $\boldsymbol{x}$
$(\boldsymbol{X}, \boldsymbol{y}) \sim \mathcal{D}$	Input $\boldsymbol{X}$ and output $\boldsymbol{y}$ (or $\boldsymbol{r}$) drawn from dataset $\mathcal{D}$
p	Probability density function
y	Dependent variable of a regression (e.g., forward returns)
r	Contemporaneous returns ($t-1$ to t)
F	Model (e.g., neural network)
f	Activation function or a single network layer
ξ	Factor return
B	Batch size
b	b-th batch
M	Number of features or independent variables
N	Number of stocks
T	Number of periods
L	Number of layers in a neural network
ℓ	ℓ-th layer
$\boldsymbol{W}^{(\ell)}$	Network weights of layer ℓ
$\boldsymbol{b}^{(\ell)}$	Network bias of layer ℓ
$\boldsymbol{\theta}^{(\ell)}$	$(\boldsymbol{W}^{(\ell)}, \boldsymbol{b}^{(\ell)})$ weight set of layer ℓ
$\boldsymbol{\theta}$	$\bigcup_{\ell=1}^{L} \boldsymbol{\theta}^{(\ell)}$ (all weight sets of neural network)

Symbol	Description
$\boldsymbol{a}$	Activation values
τ	Number of optimisation epochs
k	Kernel size (of convolutional layer)
K	Sequence length
$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})$	Loss between true $\boldsymbol{y}$ and predicted $\hat{\boldsymbol{y}}$
$J(\boldsymbol{\theta})$	Abbreviation for $\mathcal{L}(F(\boldsymbol{X}; \boldsymbol{\theta}), \boldsymbol{y})$
$\hat{\nabla} J(\boldsymbol{\theta})$	Stochastic gradient of $J(\boldsymbol{\theta})$
η	Learning rate

Author attribution statement

The proceeding thesis (with publications) is based on the following two published papers and one working paper:

(1) **Steven Y. K. Wong**, Jennifer S. K. Chan, Lamiae Azizi, Richard Y. D. Xu, "Time-varying neural network for stock return prediction," *Intelligent Systems in Accounting, Finance and Management*, 29(1), 3–18, 2022. This work is presented in Chapter 3.

(2) **Steven Y. K. Wong**, Jennifer S. K. Chan, Lamiae Azizi, Richard Y. D. Xu, "Supervised Temporal Autoencoder for Stock Return Time-series Forecasting," *Proceedings of the IEEE 45th Annual Computer Software and Applications Conference*, Madrid, Spain, 2021. This work is presented in Chapter 4.

(3) **Steven Y. K. Wong**, Jennifer S. K. Chan, Lamiae Azizi, "Quantifying neural network uncertainty under volatility clustering," *working paper*, 2022. This work is presented in Chapter 5.

Steven Wong was responsible for developing new methods, implementation, and writing the 3 papers. Steven Wong was responsible for at least 50% of the contribution, and was the first and corresponding author in all aforementioned papers.

Declaration

This is to certify that to the best of my knowledge, the content of this thesis is my own work. This thesis has not been submitted for any degree or other purposes, except where specified for publication.

I certify that the intellectual content of this thesis is the product of my own work, except where acknowledged with others, and that all the assistance received in preparing this thesis and sources have been acknowledged.

Signed: STEVEN Y. K. WONG

1st February 2023

As supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements are correct.

Signed: JENNIFER S. K. CHAN

1st February 2023

CHAPTER 1

Introduction

Forecasting in financial markets is one of the most difficult problems in machine learning. The prediction problem is *time-varying* and plagued by *low signal-to-noise* in the data. It is markedly different to traditional applications of machine learning which have observed tremendous success. To truly appreciate the unique challenges in applying machine learning to financial markets, the reader has to first develop at least a cursory understanding of financial markets. This chapter will first outline the motivations of this thesis, provide a primer on quantitative portfolio management, discuss the challenges of forecasting in financial markets and provide potential applications of machine learning in portfolio management.

1.1 Motivation

Machine learning has made significant advances across a wide range of applications, such as achieving human-like accuracy in image recognition (e.g. Krizhevsky et al., 2012; Simonyan and Zisserman, 2015; Szegedy et al., 2015; Schroff et al., 2015; He et al., 2016), speech recognition (Graves et al., 2013; Maas et al., 2013), natural language processing (NLP) (Collobert and Weston, 2008; Sutskever et al., 2014), win against a human champion in the game of *Go* (Silver et al., 2016), fully autonomous driving (Bojarski et al., 2016), synthesis of high quality speech from text (van den Oord et al., 2016), and medical image reconstruction (Kang et al., 2017). By contrast, machine learning applied to financial applications is still in its infancy. There have been some adoption of machine learning in financial applications, such as reinforcement learning for optimal trade execution (Mounjid and Lehalle, 2021) and loan default prediction (Turiel and Aste, 2020). However, linear regression is still a staple of

the financial forecasting toolkit[1]. This begs the question — can machine learning techniques revolutionise financial forecasting, as they have done in many other fields? This question is an important one. Australia has the fastest growing pension market in the world, which ranks as the 4th largest globally (Pham, 2019). Employing forecasting techniques, practitioners manage this pool of capital with the aim of generating higher returns for their clients. Any improvements to forecasting and portfolio management using machine learning could benefit the lives of many future retirees.

Recently, Gu et al. (2020) compared a suite of machine learning models on stock return forecasting and offered a glimpse of the potential of machine learning in financial applications. However, for machine learning models to be truly successful in financial applications, they must first overcome some characteristics of asset returns that complicate the forecasting problem, such as heavy tails, low signal-to-noise ratio and time-varying data generation process (DGP). Guided by this, in this thesis, we review and extend the machine learning literature to tackle some of the challenges as highlighted in Section 1.5. We focus on *deep learning*[2] techniques due to their successes in Gu et al. (2020) and in many other fields, such as image recognition and speech.

1.2 Mechanics of financial markets

In the simplest terms, a financial market (or capital market) is a place where investors (buyers and sellers) exchange financial assets, such as stocks, bonds and foreign currencies[3] at an agreed price (Drake and Fabozzi, 2010). Such activity is fundamental to a well-functioning financial market, which facilitates the transfer of capital[4] from savers (as providers of capital) to companies (as users of capital) and allows savers to earn a return on excess capital (Drake

[1]For example, see Grinold and Kahn (1999) for a discussion on forecasting models used by practitioners for stock return forecasting. The models are predominately linear.

[2]Deep learning is a subfield of *machine learning*. Details are outlined in Section 2.

[3]Common stocks are equity instruments which entitle the holder to fractional ownership of a company. An instrument where the company has agreed to repay the amount borrowed plus interest to the holder is called debt (Drake and Fabozzi, 2010).

[4]In the context of this thesis, *capital* refers to money that is available for investment. In financial theory, capital has a more philosophical meaning.

and Fabozzi, 2010). On any given trading day, buyers and sellers offer to buy or sell quoted quantities of assets. The stock market employs a *limit order book* system (Fabozzi et al., 2011a). Two queues are maintained, bid and ask queues, as illustrated in Table 1.1. In the illustrated state, there is no transaction. Suppose the last transaction occurred at $1.00/share. A new buyer is willing to pay $1.01/share for 10,000 shares. The buyer would have executed an order of 5,000 shares at $1.01 and exhausted the first row of the ask queue. The remaining 5,000 shares are added to the top of the bid queue at $1.01. Now, suppose this buyer is willing to buy 10,000 stocks at any price (as opposed to $1.01 in the original example). Then the buyer would have pushed the price up to $1.02 (second row of the ask queue). This buying (selling) pressure pushing the price up (down) is called *market impact* Each change in last traded price is called a *tick*. The amount of trading activity in a stock is called *liquidity*. A highly liquid stock allows a high volume of trading with a relatively small change in price (Amihud, 2002). In our hypothetical example in Table 1.1, the buyer may value the stock at more than $1.00/share (based on the information they have access to). Thus, they are willing to pay a higher price than the previous traded price of $1.00. Conversely, if a seller decides to sell 20,000 shares at $0.99, they are expecting the stock to be worth less than $0.99. Over the course of trading, buyers and sellers continuously impound information into the price, pushing the price higher if information is positive and lower if information is negative. A bad product review that dissuades would-be customers from purchasing the company's products may have a minuscule impact on the share price. On the contrary, some exogenous shocks, such as a company's profit for the quarter, can have a large impact on the share price. It is important to note that the last traded price reflects information of only the marginal investor. A marginal

TABLE 1.1: An illustrative order book for a hypothetical stock. The bid queue reflects potential buyers willing to buy the stock at the specified price. Similarly, the ask queue reflects potential sellers at the specified price.

#	Bid ($/share)	Quantity	#	Ask ($/share)	Quantity
1	1.00	10,000	1	1.01	5,000
2	0.99	3,000	2	1.02	11,000
3	0.98	24,000	3	1.03	2,500
4	0.95	23,000	4	1.05	51,000
5	0.90	2,000	5	1.07	4,000

investor is the investor making the trade at any point-in-time and determining the next traded price (Damodaran, 2022). In the example described at the beginning of this section, the marginal investors are the buyer and seller executing a trade at \$1.01. If a potential buyer values the stock at only \$0.40/sh, the buyer will sit deep in the bid queue and the order is unlikely to be executed. Thus, the price reflects only the information of the marginal investor and not the average information of all (potential) buyers and sellers. Now, suppose that the investor purchased the stock at \$1.00. One month later, the stock rose to \$1.10 and paid a \$0.10 dividend, the stock's *total return* for the month is $\frac{1.1+0.1}{1} - 1 = 20\,\%$, *price return* is $\frac{1.1}{1} - 1 = 10\,\%$ and *dividend yield* is $\frac{0.1}{1} - 1 = 10\,\%$. Unless specified otherwise, "return" refers to total return the investor received for holding the asset over the said period. In this thesis, we denote *contemporaneous return* (i.e., at time t, total return from $t-1$ to t) as r_t and future return (i.e., total return over t to $t+1$) as y_t. We will often use the term *cross-sectional*, which refers to computing certain quantities on a *per period* basis.

1.3 Return expectations in financial markets

Before we attempt to predict financial markets using machine learning, we have to first ask the question — *are financial markets predictable?* If so, what does the finance literature say about how are financial markets predictable? In finance literature, the study of return expectations (in other words, the prediction of returns) is known as *asset pricing*. In this section, we provide a brief discussion of asset pricing models and empirical findings. For interested readers, more details on this topic is provided in Appendix A1.

Underpinning mainstream financial theories is the *efficient market hypothesis* (Fama, 1970). In the weak form, the hypothesis postulates that investors cannot outperform the market (i.e., achieve higher return than the market at the same level of risk) using publicly known information. This assumption forms the basis of well-known theories such as the Modern Portfolio Theory (MPT) (Markowitz, 1952) and Capital Asset Pricing Model (CAPM) (Sharpe, 1964). CAPM is a theoretically grounded asset pricing model, which stipulates that sensitivity to market return is the *only factor* that is predictive of asset returns. This sensitivity measure

is known as *beta* or CAPM β. CAPM β can be found by regressing a stock's returns on returns of the market (as stipulated by Equation (A.4); Jensen, 1968). To readers familiar with physics, in my view, the importance of CAPM to finance is akin to the *Standard Model* (Oerter, 2006) to quantum physics — it provided a return forecasting model with strong theoretical underpinning and won the joint discoverers the Nobel Prize in Economics (The Nobel Foundation, 1990).

However, unlike the Standard Model, CAPM did not withstand the empirical test. Jensen (1968) was the first to note that CAPM did not align with empirical observations of asset returns. Since the publication of CAPM, numerous *anomalies* are found to be predictive of stock returns, such as the *size effect* (small capitalisation stocks outperform large capitalisation stocks; Banz, 1981) and *value premium* (cheap stocks outperform expensive stocks; Stattman, 1980; Rosenberg et al., 1985). Hundreds of firm characteristics are said to contain information on future stock returns — a survey by Harvey et al. (2016) contained 313 published asset pricing anomalies. The true DGP is likely to be significantly more complex than originally suggested by CAPM and that there may be a large set of factors that drive stock returns. A sufficiently large predictor set that could overwhelm linear regression models. The functional forms of predictors are also unknown. For instance, Fama and MacBeth (1973) tested CAPM β and β^2 and found that both were statistically significant in predicting returns and thus, opening the door to the potential use of machine learning in predicting returns.

1.4 A primer on quantitative portfolio management

Investors provide practitioners (investment managers) with capital, either through the pension system or through excess savings. In doing so, investors expect a positive return on their capital. Grinold and Kahn (1999) stated the objective of an investment manager is to *achieve higher risk-adjusted returns than the market*. More formally, the objective can be stated as,

$$\max_{r_p} \frac{\mathrm{E}[r_p - r_b]}{\sigma[r_p - r_b]}, \tag{1.1}$$

where r_p is return of the portfolio, r_b is return of the benchmark[5] and $\sigma[r_p - r_b]$ is standard deviation of portfolio return in excess of the benchmark. From Equation (1.1), it is immediately obvious that $r_p > r_b$ is required in order for the ratio to be positive. For the rest of this section, we provide a high-level overview of quantitative portfolio management as described in Alford et al. (2011) and Grinold and Kahn (1999). We have intentionally left out details of forecasting models in this section, deferring discussions to Appendix A2 for interested readers.

Quantitative portfolio management involves the use of empirical, systematic and mathematical methods to achieve the objective of the investment manager. The process of conducting quantitative portfolio management (a *quantitative investment process*) is comprised of four stages, as illustrated in Figure 1.1.

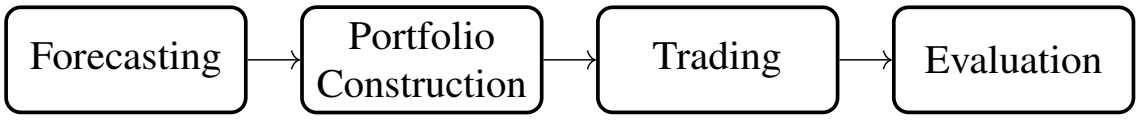


FIGURE 1.1: Illustrative stages of a quantitative investment process. Based on the process described in Alford et al. (2011).

Forecasting is comprised of three components: 1) return forecasts; 2) risk forecasts; and, 3) transaction cost forecasts. Practitioners select stocks based on their return forecasts. Thus, r_p is driven by the predictive power of the practitioner's models. For this reason, return forecasting is described by Alford et al. (2011) as the first and most critical step of an investment process. We start with some features of each stock (also known as *signals* in Grinold and Kahn, 1999; or *factors* and *anomalies* in Section 1.3). Feature examples include market capitalisation, earnings-to-price ratio, and past 12-month return of the stock. Each feature is the result of feature engineering from raw data by practitioners, either by applying domain knowledge or through machine learning (e.g., in Chapter 4, we use time-series neural networks to extract information from stock prices). Let $\tilde{\boldsymbol{X}}_t \sim \mathbb{R}^{N \times M}$ be a matrix of M features of N stocks at time t. Raw feature values are typically converted into scores. Popular

[5] A *stock index* that the portfolio is benchmarked against, e.g., S&P 500 and S&P/ASX 200.

methods include converting raw values into a $[0, 1]$ rank interval (e.g., in Gu et al., 2020) or by standardisation (Grinold and Kahn, 1999),

$$\boldsymbol{x}_{t,m} = \frac{\tilde{\boldsymbol{x}}_{t,m} - \bar{x}_{t,m}}{\sigma(\tilde{\boldsymbol{x}}_{t,m})},$$

where $\tilde{\boldsymbol{x}}_{t,m}$ is the m-th column of feature matrix $\tilde{\boldsymbol{X}}_t$ and $\bar{x}_{t,m}$ is the mean of the m column. Practitioners estimates model F to forecast returns $\hat{\boldsymbol{y}}_t \in \mathbb{R}^N$,

$$\hat{\boldsymbol{y}}_t = F(\boldsymbol{X}_t). \tag{1.2}$$

A popular choice of F amongst practitioners is the *cross-sectional* linear regression (Zhou and Fabozzi, 2011), while neural networks are used in Gu et al. (2020) and Chapter 3. As cross-sectional regression problems are discussed extensively in this thesis, using the example of a linear model as F, we formally introduce the concept of *cross-sectional* prediction in here. Suppose there are N stocks in the market, each with M features, forming input matrix $\boldsymbol{X}_t \in \mathbb{R}^{N\times M}$ at time $t = 1, \ldots, T$. The i-th row in $\boldsymbol{X}_t$ is score vector $\boldsymbol{x}_{t,i} \in \mathbb{R}^M$ of stock i and the m-th column in $\boldsymbol{X}_t$ is score vector $\boldsymbol{x}_t^{(m)} \in \mathbb{R}^N$ of feature m. We define return of stock i as the percentage change in price plus dividends, $r_{t,i} = (p_{t,i} + d_{t,i})/p_{t-1,i} - 1$, where $p_{t,i}$ is price at time t and $d_{t,i}$ is dividend at t if a dividend is paid, and zero otherwise. Regression target at t is return vector $\boldsymbol{y}_t$, where entry i is next period's return $y_{t,i} = r_{t+1,i}$ of stock i. The input-output pair $(\boldsymbol{X}_t, \boldsymbol{y}_t)$ forms a *cross-section* which contains all features at t and realised returns at $t + 1$. The time-series of cross-sections $(1, \ldots, t-1)$ form a *panel dataset* (Wooldridge, 2008): $\mathcal{D}_{t-1} = \bigcup_{\mathtt{t}=1}^{t-1}\{\boldsymbol{X}_\mathtt{t}, \boldsymbol{y}_\mathtt{t}\}$. Given the time-series of cross-sections $\mathcal{D}_{t-1}$, a popular estimation procedure used in finance literature is the *Fama-MacBeth* two-step regression procedure (Fama and MacBeth, 1973). For each $\mathtt{t} \in \{1, ..., t-1\}$, estimate the linear model on the cross-section $\{\boldsymbol{X}_\mathtt{t}, \boldsymbol{y}_\mathtt{t}\}$,

$$\boldsymbol{y}_\mathtt{t} = \hat{\xi}_{\mathtt{t},1}\boldsymbol{x}_{\mathtt{t},1} + \cdots + \hat{\xi}_{\mathtt{t},M}\boldsymbol{x}_{\mathtt{t},M} + \boldsymbol{\epsilon}_\mathtt{t}, \tag{1.3}$$

where $\{\boldsymbol{x}_{\mathtt{t},m} \in \mathbb{R} \,|m = 1, ..., M\}$ are scores of M features at time $\mathtt{t}$, $\hat{\xi}_{\mathtt{t},m}$ are regression coefficients (in finance literature, also known as *factor returns*), and $\boldsymbol{\epsilon}$ are regression residuals. This results in M time-series estimates of factor returns $\{\hat{\xi}_{1,m}, \hat{\xi}_{2,m}, \ldots, \hat{\xi}_{t-1,m}\}, m = 1, \ldots, M$.

Then, for prediction purposes, the expected return of factor m at t is the average of the time-series of observed factor returns over $1, \ldots, t-1$: $\xi'_{t,m} = \frac{1}{t-1} \sum_{j=1}^{t-1} \hat{\xi}_{j,m}$.

The pioneering work by Markowitz (1952) led to the use of variance as a measure of risk and mean-variance optimisation as a method for portfolio construction. Let $\widehat{\boldsymbol{V}}_t \in \mathbb{R}^{N \times N}$ be the estimated variance-covariance matrix,

$$\widehat{\boldsymbol{V}}_t = F^{(\text{risk})}(\boldsymbol{X}_t), \tag{1.4}$$

where $F^{(\text{risk})}$ is the model for forecasting risk. For simplicity, we assume that the risk model uses the same features as the return forecasting model (in practice, they do not have to share the same features). The diagonal of $\widehat{\boldsymbol{V}}_t$ are variance of each stock and off-diagonals are covariance between the row-th and column-th stocks. A linear-regression-based $F^{(\text{risk})}$ is simply an extension of the Fama-MacBeth regression. $t-1$ cross-sectional regressions produces M time-series of factor returns $\{\hat{\boldsymbol{\xi}}_1, \ldots, \hat{\boldsymbol{\xi}}_M\}$. Computing the variance-covariance matrix using the factor returns produces a matrix of factor risks $\boldsymbol{V}'_t \in \mathbb{R}^{M \times M}$. If a factor variance-covariance matrix is used, then the matrix must be expanded back into a stock-level variance-covariance matrix of $N \times N$ dimensions, by multiplying by the feature scores,

$$\widehat{\boldsymbol{V}}_t = \boldsymbol{X}_t \boldsymbol{V}'_t \boldsymbol{X}_t^{\mathsf{T}}. \tag{1.5}$$

As this thesis is mainly focused on return forecasting, we provide further details on risk estimation in Appendix A2.2.

We further define $\hat{\boldsymbol{c}}_t \in \mathbb{R}^N$ to be estimated transaction costs,

$$\hat{\boldsymbol{c}}_t = F^{(\text{cost})}(\boldsymbol{X}_t^{(\text{cost})}), \tag{1.6}$$

where $F^{(\text{cost})}$ and $\boldsymbol{X}_t^{(\text{cost})}$ are transaction cost model and inputs into transaction cost model, respectively. An example of transaction cost model used by practitioners is the square root model (Grinold and Kahn, 1999),

$$\hat{c}_{i,t} = \text{commission} + \frac{\text{bid-ask spread}_{i,t}}{p_{i,t}} + \kappa^{(\text{cost})} \sqrt{\frac{\mathsf{p}_{i,t}^{(\text{trade})}}{\mathsf{p}_{i,t}^{(\text{daily})}}}, \tag{1.7}$$

where commission is payable to facilitators of the trade (e.g., $0.1\,\%$ payable to brokers), bid-ask spread is the difference between the top bid and ask prices in the order book (e.g., bid-ask spread in Table 1.1 is \$0.01), $p_{i,t}$ is price of stock i at t, $\kappa^{(\text{cost})}$ is a scaling factor and is the sole parameter of the model, and $\mathsf{p}_{i,t}^{(\text{trade})}$ and $\mathsf{p}_{i,t}^{(\text{daily})}$ are dollar value of the hypothetical trade and average daily traded value (e.g., 12-month average daily traded value, where daily traded value is the day's share price $\times$ number of shares traded on the day) of stock i at t, respectively. Computation of the hypothetical trade $\mathsf{p}_{i,t}^{(\text{trade})}$ is described later in this section. Equation (1.7) indicates that transaction cost is comprised of three components: 1) a fixed percentage commission; 2) bid-ask spread (represents the cost of buying (selling) at the lowest asking (highest bidding) price rather than waiting in the bid (ask) queue); 3) a market impact component that is proportional to the relative sizes of our trade and liquidity in the stock. Both bid-ask spread and market impact have been introduced in Section 1.2 and are further explained in Appendix A2.3 For example, if the average daily traded value in Stock A is \$1 million, a \$100 trade is unlikely to cause any market impact. However, a \$100,000 buy (sell) is likely to cause the price to move higher (lower). Thus, the investor may pay a higher (or lower) price than the last traded price. In Equation (1.7), this is assumed to be proportional to the square-root of the ratio between value of the trade and daily average traded value in the stock. A further description of the transaction cost forecasting model is also provided in Appendix A2.3.

Combining return, risk and transaction cost forecasts, the portfolio is constructed using mean-variance optimisation (Markowitz, 1952; Grinold and Kahn, 1999),

$$\begin{aligned} \mathbf{w}_t' = \underset{\mathbf{w}}{\operatorname{argmax}}\, & \hat{\boldsymbol{y}}_t^{\mathsf{T}}\,\mathbf{w} - \lambda\,\mathbf{w}^{\mathsf{T}}\,\hat{\boldsymbol{V}}_t\,\mathbf{w} - \hat{\boldsymbol{c}}_t^{\mathsf{T}}(\mathbf{w}-\mathbf{w}_{t-1}) \\ & \text{subject to} \sum_i \mathrm{w}_i = 1, \end{aligned} \tag{1.8}$$

where w_i is the weight of the i-th stock in $\mathbf{w}$, $\mathbf{w}_t'$ are optimal portfolio weights[6] and λ is investor's risk aversion parameter. Note that $\hat{\boldsymbol{y}}_t^{\mathsf{T}}\,\mathbf{w}$ and $\hat{\boldsymbol{c}}_t^{\mathsf{T}}(\mathbf{w}-\mathbf{w}_{t-1})$ are both in units of

[6]Note that the vector of portfolio weights $\mathbf{w}$ (upright) differs from neural network weights $\boldsymbol{W}$ (introduced in Chapter 2) and auxiliary loss weight ω (introduced in Chapter 4).

returns, but $\mathbf{w}^\top \hat{\boldsymbol{V}}_t \mathbf{w}$ is in unit of return variance. Thus, λ also serves as a scaling factor to bring the risk penalty into the same scale as portfolio returns.

We use the following basic example to motivate the discussion on mean-variance optimisation (Equation (1.8)). Suppose that are two stocks with characteristics listed in Table 1.2. The

TABLE 1.2: Hypothetical risks and returns of Stock A and B. *Std Dev* stands for standard deviation of expected return. ρ is correlation between the expected return of Stock A and B. Optimal Portfolio is solved by maximising the Return/Risk of the portfolio by allocating to both Stock A and B.

	Stock A	Stock B	Optimal Portfolio
Expected Return (%)	10.0	8.0	8.7
Std Dev (%)	12.0	9.0	9.2
Return/Risk	0.83	0.89	0.95
$\rho_{A,B}$	0.65	0.65	

goal of the investor is to find the optimal wealth allocation within a set of assets (in this basic example, between Stock A and B). Clearly, if the investor seeks the highest expected return, the investor should allocate $100\,\%$ of their wealth into Stock A. If the investor seeks the lowest risk, then the investor should allocate their wealth into Stock B. Markowitz (1952) showed that the portfolio optimisation problem can be generalised into Equation (1.8), a convex optimisation problem, and that various portfolio objectives (e.g., maximising return, minimising risk, or the simultaneous trade-off of both) can be achieved by varying level of λ. Markowitz (1952) also showed that a better return/risk can be achieved (compared to investing into a single asset) by diversifying across assets with expected returns that are not perfectly correlated (theoretical background on *Modern Portfolio Theory* is given in Appendix A1). In our two-stock basic example, expected return of the portfolio is,

$$\mathrm{E}[r_p] = w_A\,\mathrm{E}[r_A] + w_B\,\mathrm{E}[r_B],$$

where $\mathrm{E}[r_{\{p,A,B\}}]$ and $w_{\{p,A,B\}}$ are expected return and weights of portfolio, Stock A and Stock B, respectively. Expected risk (when measured in variance) of the portfolio is,

$$\sigma_p^2 = w_A^2\sigma_A^2 + w_B^2\sigma_B^2 + 2w_A w_b\sigma_A\sigma_B\rho_{A,B},$$

where $\sigma^2_{\{p,A,B\}}$ is expected risk of portfolio, Stock A and Stock B, respectively. Continuing with our basic example, for illustrative purposes, suppose the investor is seeking the maximum $\frac{\mathrm{E}[r_p]}{\sigma_p}$ (this is known as the *Sharpe ratio* when computed on realised portfolio returns and is introduced in Section 3.2). Then, using a solver, we find that the optimal allocation is $w_A = 0.36$ and $w_B = 0.64$, which results in $\mathrm{E}[r_p] = 0.087, \sigma_p = 0.092, \frac{\mathrm{E}[r_p]}{\sigma_p} = 0.95$ (also shown in the last column of Table 1.2). Thus, by diversifying across two stocks, the portfolio has higher expected return and lower risk than investing in Stock B and Stock A alone, respectively. Solving Equation (1.8) using a solver and assuming transaction cost is zero, we find that the maximum $\frac{\mathrm{E}[r_p]}{\sigma_p}$ objective is equivalent to $\lambda = 5$ in Equation (1.8). Other special cases include maximum return ($\lambda = 0$) and minimum risk ($\lambda \to \infty$). In general, risk aversion parameter λ is subjectively chosen by the investor depending on their risk appetite.

So far, our basic example involves a single period optimisation. Suppose the investor rebalances their portfolio at the end of every month and that the weights immediately prior to rebalancing are $w_A = 0.3$ and $w_B = 0.7$. Changing from $w_A = 0.3$ to $w_A = 0.36$ ($w_B = 0.7$ to $w_A = 0.64$) incurs transaction costs. Thus, the expected *after cost* return of Stock A (B) is likely less than $15\,\%$ ($8\,\%$), and the optimal portfolio in the presence of transaction costs will differ to the theoretical frictionless optimal portfolio. Suppose that the portfolio is \$1 million in value and transaction costs are assumed to be $\hat{\boldsymbol{c}}_t = [0.002, 0.002]^\mathsf{T}$. Then, the hypothetical trade of Stock A is computed as $\mathsf{p}_{i,t}^{(\text{trade})} = 1,000,000 \times (0.36 - 0.3) = 60,000$, which incurs cost of $60,000 \times 0.002 = 120$.

In analysing Equation (1.8), it can be seen that the objective is maximised if the portfolio places $100\,\%$ weight onto the stock with the highest expected return, if risk and costs are ignored. However, with a covariance matrix where the entries $\hat{v}_{t,i,j} < \hat{v}_{t,i,i}, i \neq j$, the risk penalty $\mathbf{w}^\mathsf{T} \widehat{\boldsymbol{V}}_t \mathbf{w}$ encourages diversification and prevents the portfolio from being fully aligned with return forecasts. The portfolio is further constrained by the cost to trade $\hat{\boldsymbol{c}}_t^\mathsf{T}(\mathbf{w} - \mathbf{w}_{t-1})$, which only allows the portfolio to switch between stocks if the increase in expected return is greater than the two-way cost to trade (i.e., the cost of a buy and a sell). A stock position can become "stale" if its return forecast no longer ranks highly but no other stock offers a sufficiently high return forecast to cover transaction costs. In this case, a portfolio can

continue to hold a suboptimal stock even if better (theoretical) options are available. Thus, over time, portfolio weights reflect the weighted averages of past return forecasts, rather than the latest return forecasts. In sum, mean-variance optimisation is a balancing act between maximising return (forecasts), while minimising risk penalty and transaction costs.

Next, the desired portfolio is then implemented by trading the difference between the desired portfolio and the existing portfolio,

$$\mathbf{w}_t = f^{(\text{trade})}(\mathbf{w}'_t - \mathbf{w}_{t-1}),$$

where $f^{(\text{trade})}$ denotes the *trading function* that produces the actual portfolio $\mathbf{w}_t$ (i.e., an abstract function that involves sending orders to the market and observing actual execution of the order) and the trades are given by $\mathbf{w}'_t - \mathbf{w}_{t-1}$. Actual transaction costs incurred by trading is given by,

$$\boldsymbol{c}_t = f^{(\text{tcost})}(\mathbf{w}'_t - \mathbf{w}_{t-1}),$$

where $f^{(\text{tcost})}$ is the actual market impact (as discussed in Section 1.2) and commissions paid for the trades. For example, if inputs into $f^{(\text{tcost})}$ are \$1 million worth of trades, and costs are comprised of $0.15\,\%$ of market impact and $0.05\,\%$ of commission, then actual transaction cost is $c_t = \$1\,\text{million} \times (0.0015 + 0.0005) = \$1000$. To minimise market impact, a practitioner may choose to trade patiently[7]. In doing so, the practitioner incurs opportunity cost (if return forecasts are predictive of returns, waiting for a favourable trade price will lead to foregone returns) and risk around the eventual execution price (Alford et al., 2011). The resultant portfolio may differ from the desired portfolio, due to adverse price movements or prevailing liquidity of the stocks.

Finally, performance of the portfolio is computed as the sum of actual realised stock returns less actual transaction costs,

$$r_{p,t+1} = \mathbf{w}_t^{\mathsf{T}}\, \boldsymbol{r}_{t+1} - \boldsymbol{c}_t^{\mathsf{T}}(\mathbf{w}'_t - \mathbf{w}_{t-1}). \tag{1.9}$$

[7]Using the example in Section 1.2 to illustrate, suppose the maximum price the investor is willing to pay is \$1.00/share. Then, the investor's bid will sit in the bid queue, waiting for a seller who is willing to sell at \$1.00/share. The investor is guaranteed that the price paid is \$1.00 but there is uncertainty as to when the trade will occur (if at all). Conversely, if the investor is willing to pay any price, then the investor can trade immediately but will consume the ask queue and thus "move" the market price with the trade.

This completes the link from return forecasts (the most important step of the investment process) to the outcome of the portfolio (objective of the investment manager).

1.5 Challenges of forecasting in financial markets

So far, we have introduced quantitative portfolio management, basic financial theory and the vast potential feature set. We have also briefly discussed the importance of forecasting. In this section, we start by providing several stylised facts on financial markets before formally describing the challenges of forecasting in financial markets.

Fact 1: Asset returns have heavier tails than stipulated by the Normal distribution (Cont, 2001).

For example, on 26th July 2018, Facebook Inc. reported lower than expected second quarter revenue and daily active user count (Salinas and Castillo, 2018). The stock fell $18.96\,\%$ on the day, as illustrated in Figure 1.2. This event is so rare that assuming daily returns are normally distributed and using observations from *initial public offering*[8] to 25th July 2018, the probability of observing such an event is $6e^{-17}$.

Fact 2: Financial markets can exhibit endogeneity.

Literature and media outlets have documented some evidence of endogenous factors driving stock returns. Over December 2020 to January 2021, social media users coordinated trading activity in GameStop Inc., causing its share price to rise $1998\,\%$ in two months. The dramatic rise in value of GameStop shares was partly driven by a phenomenon known as a *short squeeze* (El-Erian, 2021), where investors betting against a rising stock are forced to unwind their bet, causing further buying pressure on the stock. This caused cascading buying pressure on the stock and an escalating stock price, as shown in Figure 1.3. Endogeneity is also said to have caused some stock market crashes, such as the crash of October 1929 and the "Dot-com" crash of April 2000 (Johansen and Sornette, 2002).

[8] Initial public offering (IPO) is when a private company sell shares to the public for the first time. After IPO, the stock becomes a publicly traded stock on the stock exchange.

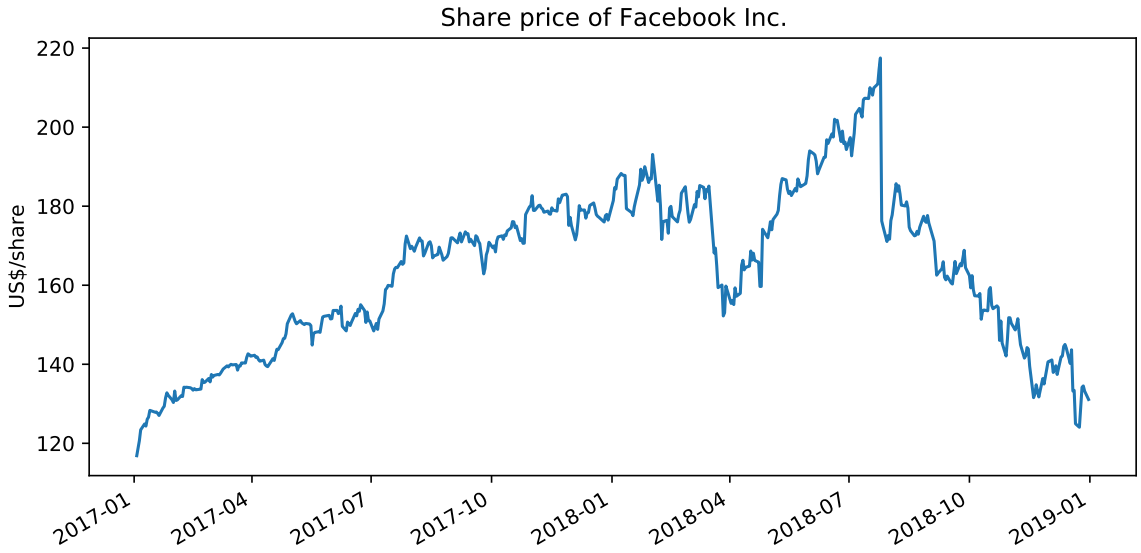


FIGURE 1.2: Share price of Facebook Inc. over 2017–18. The share price of Facebook is observed to be on an upward trajectory prior to July 26, 2018 and a downward trajectory afterwards. Source: Center for Research in Security Prices (CRSP) database.

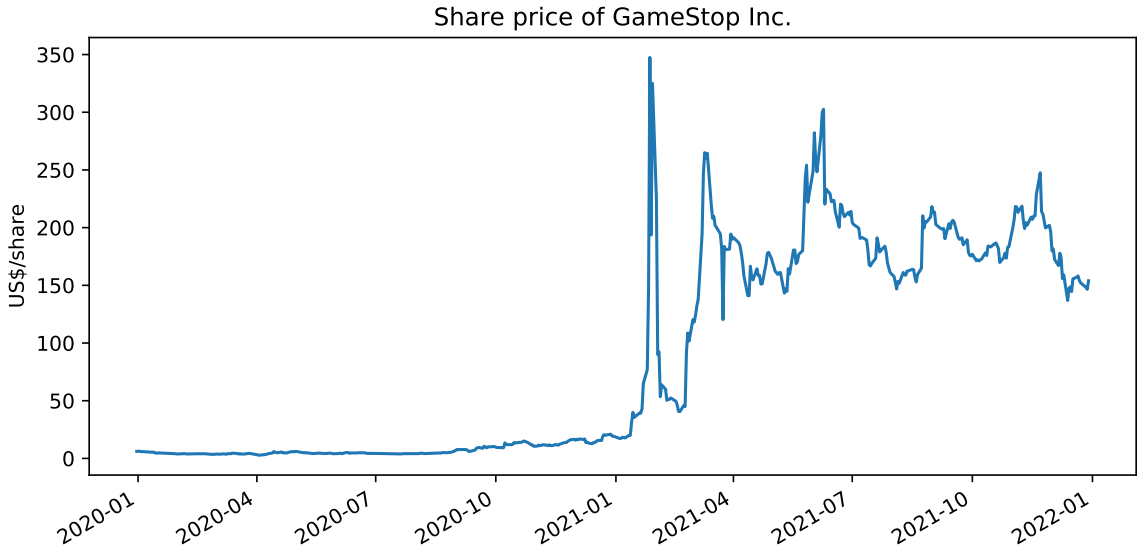


FIGURE 1.3: Share price of GameStop Inc. over 2020–21. The share price of GameStop peaked on January 27, 2021. Source: Yahoo! Finance (2022a).

Fact 3: Asset returns exhibit volatility clustering, where returns display irregular bursts of volatility that are localised in time (Cont, 2001).

Stock market crashes can also be caused by an exogenous shock, such as the 2020 global stock market crash due to an emerging pandemic (Song et al., 2022). For example, the price of Devon Energy, a U.S. oil and gas producer, plummeted during the March 2020 stock market crash, as shown in Figure 1.4. The square of daily returns jumped to 0.14 during the height of the crash, compared to a mean of 0.001 over 2017–2021. Volatility is also seen to *cluster* in time. Following the peak in March 2020, volatility remains elevated over the next 6–12 months.

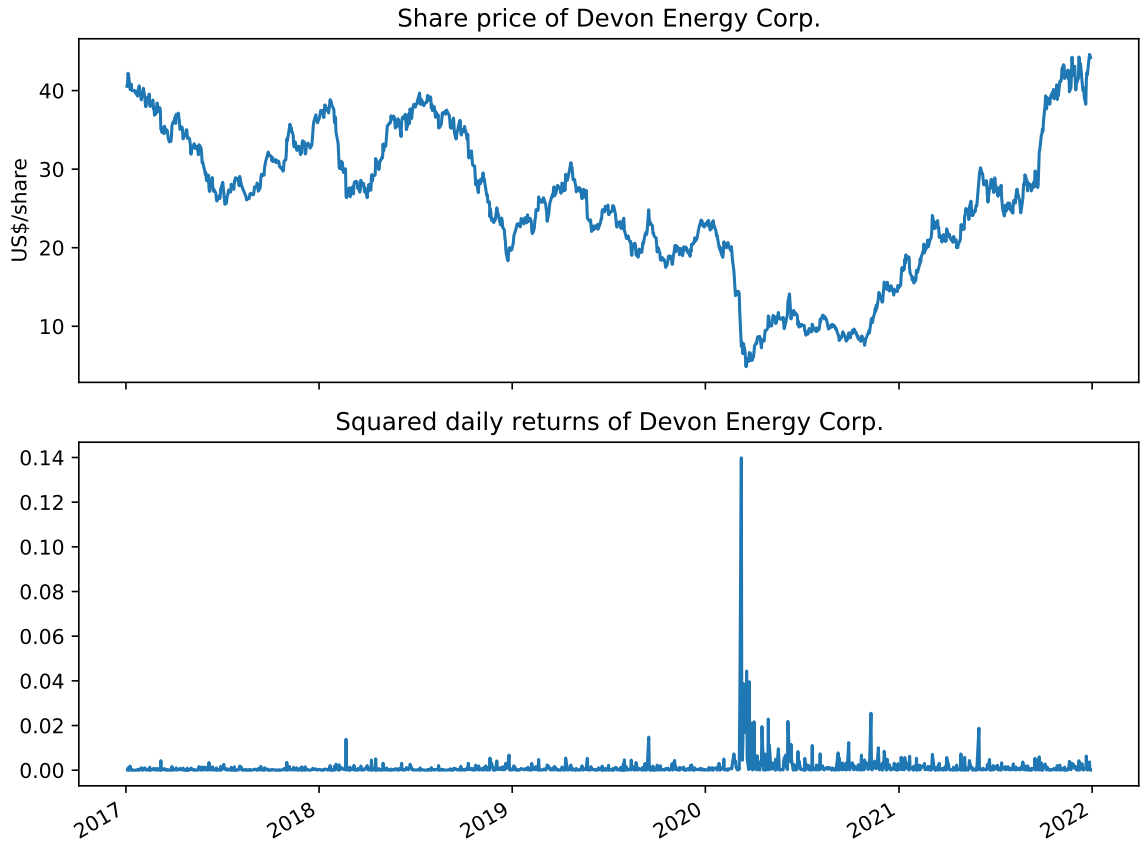


FIGURE 1.4: Share price of Devon Energy Corp. over 2019–21. Daily return volatility spiked to $14\,\%$ during the height of the crash. Source: Yahoo! Finance (2022b).

In addition to **heavy tails** and **volatility clustering**, Cont (2001) has documented other asset return characteristics such as *absence of autocorrelation*, *aggregational Gaussianity*, *skewness* and *conditional heavy tails*.

There is one major distinction between the setup of conventional machine learning applications and financial applications — stationarity of the DGP. In time-series analysis, *non-stationarity* typically refers to properties (e.g., mean and variance) of a time-series changing over time (Nason, 2006). In machine learning literature, non-stationarity refers to the DGP changing over time, such as changes in the conditional distribution of the output given the input (Gama et al., 2014). To avoid ambiguity, we use the terms *time-varying* or *time-variability* of the DGP to refer to non-stationary DGP. However, we continue to use the term *stationary* to describe DGP that do not change over time.

Conventional machine learning models are trained offline, using a historical set of training data and are deployed after batch training (Gama et al., 2014). This training scheme is suitable for stationary problems where the training set is assumed to be drawn from the same DGP as out-of-sample data. As out-of-sample data are drawn from the same distribution as training data, the *generalisation gap*[9] is expected to be relatively small and the model is expected to perform well after deployment. Examples of stationary problems include image recognition (Schroff et al., 2015) and text translation (Sutskever et al., 2014). However, some prediction problems are time-varying, yielding a phenomenon known as *concept drift* (Schlimmer and Granger, 1986; Widmer and Kubat, 1996; Gama et al., 2014). For example, predicting a user's interests when following an online news stream is likely time-varying (Gama et al., 2014). Finance literature has also documented evidence of time-variation of the DGP. Pesaran and Timmermann (1995) estimated linear models with permutations of firm characteristics over time, and performing model selection using both statistical and financial measures on U.S. stocks. Both the selected variables and their coefficients of the best model change over time. Bossaerts and Hillion (1999) reported similar findings in international stocks. There is no consensus on the cause of time-varying predictability in the academic discourse. Some argued that this time-variability is driven by macroeconomic conditions (e.g., Angelidis et al., 2015). While explanations offered by McLean and Pontiff (2016) relate to data-mining bias and effects of arbitrage by investors (which the authors referred to as publication-informed trading). In other words, investors taking advantage of this effect causes its "mispricing" to

[9]Generalisation gap is defined as the difference between out-of-sample loss and training loss (Goodfellow et al., 2016).

disappear (for example, see Dong et al., 2020 for a proposed mechanism with which this occurs). Thus, it is unsatisfactory for a practitioner to learn a static model as out-of-sample performance can vary.

As noted in Section 1.3, there are hundreds or more factors that exhibit predictive power over stock returns. These include (but are not limited to):

- Price-derived features, such as a stock's past performance (Jegadeesh and Titman, 1993).
- Financial statement-derived features, such as valuation metrics (Asness et al., 2013).
- Social media (as illustrated by the GameStop example) and web searches (Huang et al., 2020).
- Media reports (Fang and Peress, 2009).

The dataset in Gu et al. (2020) contains mainly price and financial statement features and is the same dataset used in Chapter 3. In Chapter 4 and 5, only price features are used as these chapters focuses on time-series predictions and uncertainty quantification (of time-series predictions). Such a vast feature set has the potential of overwhelming conventional regression techniques such as ordinary least squares (OLS), due to multi-collinearity (Gu et al., 2020). Thus, any proposed machine learning alternatives to linear models must be able to handle a large feature set.

In sum, stock return prediction poses a unique challenge for machine learning research. Stock returns exhibit difficult to handle statistical characteristics such as heavy tails and volatility clustering, and suffer from low signal-to-noise ratio and time-variability of the DGP. These challenges are distinct from conventional applications of machine learning which have seen significant advances.

1.6 Potential applications of machine learning in portfolio management

As discussed in Section 1.4 and 1.5, machine learning in portfolio management presents some unique challenges and requires new approaches that differ from conventional applications. In this section, we discuss gaps in literature and identify ways in which machine learning can be advanced or applied in financial markets.

1.6.1 Cross-sectional prediction using online deep learning

As noted in Section 1.5, return forecasting is an arduous task. Stock returns are not well-behaved, plagued with heavy tails, low signal-to-noise ratio and time-varying cross-sectional relationships. However, it is also the most important step of an investment process.

Much of the finance industry still relies on linear models. By contrast, machine learning has achieved significant progress in other fields and could similarly offer prediction performance improvements to empirical finance. Weigand (2019) provided a recent survey of machine learning applied to empirical finance and noted that machine learning algorithms show promise in addressing shortcomings of conventional linear models (such as the inability to model non-linearities and handle large number of covariates). Notable works applying neural networks to cross-sectional stock return prediction using a large feature set include Messmer (2017), Abe and Nakayama (2018) and Gu et al. (2020). Both Messmer (2017) and Abe and Nakayama (2018) are straightforward applications of feedforward networks[10] on stock returns, where the input consists of tens of features, predicting U.S. and Japan stock returns, respectively. Gu et al. (2020) compared a set of well-known machine learning models on forecasting U.S. stock returns and found neural networks to provide the best performance. Potential time-variability is assumed to be driven by macroeconomic conditions and is modelled by interacting firm level features with macroeconomic indicators. Arguably, this is an inefficient way of modelling interaction effects, as the 94 firm-level features are

[10] Feedforward neural networks are discussed in Section 2.1.

interacted with 8 macroeconomic variables, resulting in 920 features (together with dummy variables of 74 industries, $94 \times (8 + 1) + 74 = 920$). Of the firm-level features used in the three works, Gu et al. (2020) contains the most firm-level features (94). Messmer (2017) contains 61 and are all contained within Gu et al. (2020). Abe and Nakayama (2018) contains the least, at 25. However, due to different naming conventions, we cannot ascertain how many are contained with Gu et al. (2020). Moreover, they do not consider all possible avenues of time-variability of asset pricing models, such as the effects of investors' own trading, as highlighted by McLean and Pontiff (2016), and exogenous shocks. For instance, Lev and Srivastava (2019) noted that the prominent *value* factor[11] (Rosenberg et al., 1985; Fama and French, 1992) has been unprofitable for almost 30 years — a period that includes multiple business cycles and thus cannot be explained by macroeconomic conditions alone. The authors noted that returns to the value factor have been negative since 2007, suggesting a change in the underlying relationship. There are further empirical evidence of changes in DGP. Employing *genetic algorithms*[12] (Mitchell, 1996) to predict U.S. stock returns, Brogaard and Zareei (2022) also found stock return predictability to have declined over time, which implies that markets have become increasingly more efficient. Other works have sought to incorporate finance theory directly into the network architecture and have used more advanced network architectures. Gu et al. (2021) used an autoencoder to form "latent factors" and factor exposures, in similar spirit as principal component analysis (PCA) (Hastie et al., 2020). The resultant model is analogous to the Arbitrage Pricing Theory (APT) model but with latent factors constructed by the autoencoder from a large feature set. Chen et al. (2021) used a *generative adversarial network* (Goodfellow et al., 2014) to enforce the *no arbitrage condition* in APT and reported strong performance in predicting U.S. stock returns. Changing DGP is modelled with 178 macroeconomic indices. The authors reported declining performance over time, and that using a small number of macroeconomic indices is only marginally better than no macroeconomic data (i.e., a fixed DGP), and that including all 178 indices led to severely

[11] Suppose the share price is \$1.00 and the firm's asset value (net of debt) is \$2.00 per share. Then the *book-to-market* score is $2/1 = 2$. A high score is interpreted as the stock trading cheaply relative to value of its assets. This factor has been profitable since 1920s but its profitability has greatly diminished since its discovery in 1986. Lev and Srivastava (2019) argue that this is due to deficiencies in accounting standards and economic development.

[12] Genetic algorithms randomly search through candidate model specifications through simulated evolution.

worse performance. Financial markets have also observed plenty of exogenous shocks over time, some of which do not have a parallel in history (e.g., the COVID-19 pandemic).

These empirical evidence suggest changes in the DGP may be unpredictable (e.g., due to investors' own arbitrage and exogenous shocks). Thus, there exists a gap in literature for deep learning models that can track changes in the DGP of financial markets driven by unknown dynamics.

1.6.2 Time-series pattern recognition in noisy environments

Stock returns are notoriously noisy. The best performing model in Gu et al. (2020) had R^2 of $0.4\,\%$[13]. In practice, cross-sectional correlation between expected return and actual realised return of $5\,\%$ can be considered as "good" and $10\,\%$ is "great" (Grinold and Kahn, 1999). By contrast, state-of-the-art image recognition models can achieve image classification accuracy of over $90\,\%$ (for example, see Zhai et al., 2021). Thus, from a signal-to-noise perspective, financial markets are vastly different from fields where deep learning has excelled, such as image recognition.

In neural network architectural design, a network with greater *depth* is thought to be more efficient than a shallow but *wide* network (i.e., a network with only a few layers but each layer has many nodes) in approximating an arbitrary function (Lu et al., 2017). Training very deep neural networks (over 100 layers deep) for image classification problems saw a breakthrough in the form of *ResNet* (He et al., 2016), where skip connections that "jump" over one or more layers are added to allow uninterrupted information flow between connecting layers. This alleviates the problem of *vanishing gradient*, where the magnitude of the gradient diminishes as training error is backpropagated through the network (Kolen and Kremer, 2001). However, in very noisy environments such as financial markets, focus of training should be on robustness rather than expressiveness, as highly expressive networks may overfit on noise, leading to poor out-of-sample performance. To this end, there is no conclusive evidence in literature regarding the robustness of neural networks in noisy environments. Drawing on

[13] Gu et al. (2020) used a non-standard definition of R^2. Further details are provided in Section 3.2.1 and Section 3.5.

findings in other applications, Rolnick et al. (2018) finds that neural networks are robust to high levels of artificially injected mis-classified labels in simple image recognition tasks. The authors note that effective batch size (a concept that we will introduce in Section 3.2.2) decreases as the level of white noise increases. Thus, highly noisy environments require larger batch sizes. On the contrary, Moradi et al. (2021) finds that neural networks are not robust to noise in clinical text. The authors inject character-level and word-level perturbations to reflect realistic typographic errors encountered in the real world and find three different language models trained specifically on clinical texts to have experienced material accuracy declines in medical diagnostic tasks.

Ways to combat noisy data include increasing the amount of data used in training (Rolnick et al., 2018)), which may not be readily available, and regularising the model. Popular regularisation techniques for neural networks are L_1 and L_2 penalties (Goodfellow et al., 2016), *early-stopping* (Goodfellow et al., 2016) and *dropouts* (Srivastava et al., 2014). L_1 and L_2 penalties in neural networks are analogous to their counterparts in linear models and shrink network weights toward zero. Early-stopping can be interpreted as L_2 penalties, and dropout can be interpreted as ensembling using subnetworks. Both of these techniques are introduced in Section 2.4. For classification problems, Patrini et al. (2017) propose to estimate noise rates in class labels and introducing a correction term in the loss function which negates the probability that a label is assigned due to noise. Multi-task learning (MTL) has also been shown to improve generalisation performance across a range of classification tasks, such as facial landmark recognition (Zhang et al., 2014) and natural language processing (Collobert et al., 2011). MTL involves the addition of an auxiliary learning task that is related to the primary learning task. The auxiliary learning task is thought to encourage representation sharing and is introduced more formally in Section 4.1. There remains a need for regularisation techniques specifically designed for regression noisy environments that may potentially have broader applications outside of finance.

In Section 1.6.1, we have introduced the cross-sectional prediction problem in finance. An alternative to cross-sectional prediction that is applicable in financial markets is *time-series forecasting*. This can be interpreted as pattern recognition on past stock price or return

patterns to forecast future returns. Convolutional neural networks (CNNs) have proved to be invaluable in image recognition tasks (e.g., Krizhevsky et al., 2012) and could extract more patterns from share prices beyond anecdotal patterns documented in technical analysis. Sezer et al. (2020) provided a recent survey on financial time-series forecasting with deep learning and noted long short-term memory (LSTM) was the most popular method, followed by CNN. Most works are straightforward applications of different neural network architectures on stock returns (e.g., Chen et al., 2015), and are on 1–3 days ahead forecasts. This differs to the 1-month ahead forecast of the momentum effect which, we argue, is more relevant for investment managers due to constraints of transaction costs. There are two approaches to dealing with noise in financial time-series forecasting in literature, both of which we see as being deficient. First is to treat the time-series forecasting problem as a classification problem (i.e., 1 if the stock rose over the next day, 0 otherwise; see Chen et al., 2015; Altilio et al., 2019). This neglects the magnitudes of expected returns which will help practitioners in differentiating relative performance of stocks. Second is to first apply wavelet transform (Meyer, 1993) to denoise the sequence, then fit the denoised trend using a neural network (e.g., Yan and Ouyang, 2017; Li and Tam, 2017). Wavelet transform treats share price oscillation around a trend as "waves" which are then removed. This relies on fitting parametric waves onto the sequence and may inadvertently remove useful features from the sequence. Gap exists in existing literature for an end-to-end neural pattern recognition technique that is robust to noisy patterns in stock prices.

1.6.3 Forecast uncertainty quantification

Consider the following thought experiment. Suppose a practitioner has a model that can perfectly forecast next day's asset returns and that the practitioner's goal is to maximise terminal wealth. Then, on each day, the most rational decision would be to place all of the investor's wealth into the asset with the highest expected return on the next day. Next, suppose that the investor's model is a noisy estimator of future asset returns. Then, the investor may choose to diversify across multiple assets and not place all their wealth on a single bet. Based on this thought experiment, we would expect forecast certainty to have a role in

the portfolio optimisation process. Various bet allocation models have been developed. In wagering, where bets are independent and have well-defined binary outcomes, the optimal bet allocation strategy is the *Kelly criterion* (Kelly, 1956). Forecast certainty is incorporated into the Kelly criterion via expected probabilities of discrete outcomes. The Kelly criterion has been extended to the case of Gaussian distributed outcome, where the optimal bet size is scaled by the inverse of variance (Byrnes and Barnett, 2018). Mean-variance portfolio optimisation (Markowitz, 1952) assumes that asset returns are described by mean and variance of their expected returns. The resultant portfolio is extremely sensitive to expected returns which are difficult to forecast. Black-Litterman portfolio optimisation was proposed to address this shortcoming (Black and Litterman, 1992). In Black-Litterman, practitioners provide both their "views" (expected returns) and "strength" of their views (forecast certainty). These views are then incorporated into portfolio optimisation as priors in a Bayesian manner. Thus, it is useful for a neural network to provide both the conditional mean (forecast) and conditional variance (forecast uncertainty) which can then be used downstream in portfolio optimisation, such as in determining optimal bet size.

Bayesian neural networks offer both forecasts and forecast uncertainties through imposing a full Bayesian treatment over the entire network (Mitros and Namee, 2019). This involves placing priors on network weights and training the network using Monte Carlo Markov Chain (MCMC). However, a full Bayesian treatment incurs a high computational cost (Quiroz et al., 2019). Recent advances focus on generating parameters of a distribution that is assumed to have generated the data, (e.g., Lakshminarayanan et al., 2017; Amini et al., 2020). These works provide an interesting way of quantifying forecast uncertainty, without the cost penalty of a full Bayesian approach. However, both Lakshminarayanan et al. (2017) and Amini et al. (2020) were developed for non-time-series applications and do not consider the possibility of variance changing over time. In the context of financial time-series forecasting, stock returns are known to exhibit *volatility clustering* (as discussed in Section 1.5). Gaps exist in literature for neural network forecast uncertainty quantification techniques that can handle time-varying uncertainty, particularly in the context of financial time-series. From a finance application perspective, forecast uncertainty can be used to size bets, or as advanced warning to protect the portfolio from increasing risk. For example, if forecast uncertainty reaches

a certain threshold, an investor could purchase portfolio insurance (e.g., put options which allow the investor to sell stocks to the issuer of the options at a pre-agreed price) or liquidate positions to reduce risk.

1.6.4 Other possible directions

In addition to the aforementioned applications of machine learning in portfolio management, machine learning can also be used to extract useful information from unstructured data and for efficient trading (in cost minimisation). These two topics are not addressed in this thesis. Nonetheless, we provide a discussion on the two topics as future research directions.

Inputs form the bedrock of any prediction model. Pre-existing feature sets used in literature and in practice typically evolve around price, company financial information and other economic variables. Recently, there is an emerging trend towards incorporating *alternative data* in the investment process. These datasets comprise of unconventional and often unstructured information about a company or industry. For instance, Ranco et al. (2015) used Support Vector Machine (SVM) to classify $1.5\,\text{million}$ tweets on Twitter[14] on 30 stocks in the Dow Jones Industrial Average index (Dow Jones) into positive, neutral and negative sentiment over 15 months. The authors found that polarity of tweet sentiment was associated with $1\,\%$ of cumulative excess return in the three days following the peak. Souma et al. (2019) used a LSTM to classify Reuters[15] news articles into positive and negative sentiment. The authors reported positive prediction accuracy when applied to high frequency intraday tick data on stocks in the Dow Jones index. Other potentially useful unstructured data include job listing websites, product reviews, website traffic, satellite imagery of factories and car parks, and product internet search trends. Techniques such as deep learning can be used to convert these unstructured data into quantifiable data and incorporated in a prediction model.

After a portfolio has been selected, investors would implement the portfolio in the most cost efficient manner. As introduced in Section 1.2 and discussed in Appendix A2.3, market

[14]Twitter is a social media platform where users can post short messages (called *tweets*) of any topic. URL: twitter.com

[15]Reuters is a global news outlet. URL: www.reuters.com

impact could substantially reduce realised returns. Therefore, the goal is to design a trading policy (e.g., when and how to split a trade into parcels) that minimises costs incurred. This is related to the *one-way trading problem* in computer science (El-Yaniv et al., 2001), where a player observes a price sequence and decides whether or not to accept the current price. The game ends when the required amount has been traded. There are two approaches to this problem — online learning and reinforcement learning. Online learning (also called no-regret learning, a concept to be introduced in Section 3.2.3) approaches the optimal trading problem through the lens of game theoretics. The problem is set as a game against an adversary (also called the *nature*) and the goal is to compare favourably to the best expert in hindsight (e.g., the best model parameters trained using all observations up to t). Dworkin et al. (2014) proposed the *Pursuit-Evasion Without Regret* algorithm for optimal trading, which extends online learning to incorporate a *state*[16]. The authors showed that the proposed algorithm outperformed the *constrained follow-the-lazy-leader* algorithm on inventory management. Distinct features of no-regret learning are the assumption of adversarial outcome and the focus on worst-case performance (as nature can arbitrarily choose an outcome that is to the worst detriment of the player). Possible extensions to this line of literature is to allow for *some* predictability in intraday returns, such as the reversal pattern. Another possible approach is *reinforcement learning*, which focuses on learning the optimal policy itself. Nevmyvaka et al. (2006) was the first to apply reinforcement learning on the optimal trading problem and reported substantial reduction in trading costs. Optimal policy learnt was based on time left to trade and quantity remaining. One potential advancement is to use *deep reinforcement learning* (e.g., see François-Lavet et al., 2018) to learn a more complex Q-value function, beyond simply using the time and quantity dimensions. For instance, other variables such as market capitalisation and recent share price performance may contain useful information for optimal trade execution.

[16]The application of Dworkin et al. (2014) is in algorithmic trading, where an algorithm is used to autonomously trade stocks in order to achieve an objective (e.g., minimise cost). In this context, state refers to the inventory position in a given stock. Conventional no-regret learning algorithms are stateless. This stateful augmentation blends reinforcement learning with no-regret learning (Dworkin et al., 2014).

1.7 Contributions and structure of the thesis

At this point, the reader has been introduced to the portfolio management problem and relevant financial theory. As discussed in Section 1.5 and 1.6, there are several gaps in literature on applying deep learning in a portfolio management context.

This thesis provides three significant advances in deep learning techniques applicable to financial markets. Each contribution details a methodological improvement to deep learning that addresses a challenge in financial markets (e.g., time-varying DGP, low signal-to-noise and volatility clustering), combined with a demonstration of the contribution in a financial market application. As deep learning is featured in all three main contributions of this thesis, a comprehensive review of neural networks is provided in Chapter 2, covering common network architectures (Section 2.1, 2.5.1, 2.5.2), network training and optimisation (Section 2.2), weight initialisation (Section 2.3), activation functions (Section 2.1), and neural network regularisation techniques (Section 2.4). This chapter provides readers with an essential understanding of neural networks which is required to fully grasp the advances proposed in the rest of this thesis.

In Chapter 3, we address the time-varying cross-sectional prediction problem (as described in Section 1.6.1) by introducing the Online Early Stopping (OES) algorithm for training neural networks online. The neural network trained using OES is able to adapt to changes in the DGP over time. We provide an optimality guarantee, where the performance of the algorithm is lower bound by a multiple of the variance of the DGP. We compare OES to a stationary network (i.e., a network that is trained offline and does not vary with time) and Dynamic Exponentially Time-Smoothed Stochastic Gradient Descent (DTS-SGD), a state-of-the-art online non-convex optimisation algorithm (introduced in Section 3.2.3). We demonstrate the benefits of OES on a synthetic dataset and in predicting U.S. stock returns as a direct comparison to Gu et al. (2020). This application tackles both the cross-sectional stock return forecasting problem (as discussed in Section 1.6.1) and time-variability problem of financial markets (as discussed in Section 1.5). We show that a neural network trained using OES outperformed the state-of-the-art on the aforementioned problems. We also demonstrate that

the network is able to track changes in the market, such as turning points of markets, with compelling results that are likely to be useful to practitioners.

In Chapter 4, we simultaneously tackle the time-series pattern recognition and noise-robust learning problems (as discussed in Section 1.6.2) by introducing a supervised autoencoder[17] into a temporal convolutional network. We name this network the Supervised Temporal Autoencoder (STAE). We argue that the supervised autoencoder imposes a nonparametric functional form on the model, encouraging the network to retain features that are beneficial to both the primary forecasting task and the auxiliary reconstruction task. The reconstruction task also improves the interpretability of the model, as users can visualise features retained by the network by inspecting the reconstructed sequence. We show that the proposed STAE provides economically meaningful improvements over the *momentum effect*, a known predictor of stock returns in finance literature. We also provide a precedence on applying sequential neural networks[18] to financial time-series at a large scale and is investable through forecasting 1-month ahead returns. We provide a benchmark against popular sequential neural networks, namely temporal convolutional network (TCN), LSTM, Neural Basis Expansion Analysis for interpretable Time Series (N-BEATS) and transformer (both N-BEATS and transformer are state-of-the-art architectures for time-series/sequential applications), and demonstrate class-leading performance. We hypothesise that supervised autoencoder is a potent regularisation technique for neural networks that may find applications in other noisy environments.

In Chapter 5, we combine and extend two state-of-the-art methods, Ensemble and Evidential, into a unified framework for the quantification of uncertainty in financial time-series (as discussed in Section 1.6.3). The framework comprises of four improvements, namely the use of SMD, separate modelling of distribution hyperparameters, ensembling and use of second order return information. We propose a simplified parameterisation of the problem as a scale mixture, which leads to the use of a Gamma prior on a variance scaling variable. Using the UCI benchmark dataset, we demonstrate uncertainty quantification performance that is overwhelmingly in favour of SMD over Normal-Inverse-Gamma (NIG). Comparing to

[17] A network architecture introduced in Section 2.5.3.

[18] In this context, sequential neural network refers to neural networks that are applicable for time-series applications, such as recurrent neural network (RNN), LSTM and CNN.

Ensemble and Evidential on the UCI dataset, cryptocurrency and U.S. equities time-series forecasts, we demonstrate class-leading performance on all three datasets in terms of widely used prediction performance measures. We show that only our proposed framework can provide uncertainty estimates that track realised forecast errors. On the UCI benchmark dataset, we show that our proposed framework benefits uncertainty quantification in non-time-series applications. Thus, we conjecture that some or all of our proposed improvements may benefit applications in other areas of machine learning.

In Chapter 6, we summarise the contributions provided in this thesis and discuss future research directions. In particular, we highlight several potential improvements to our work.

CHAPTER 2

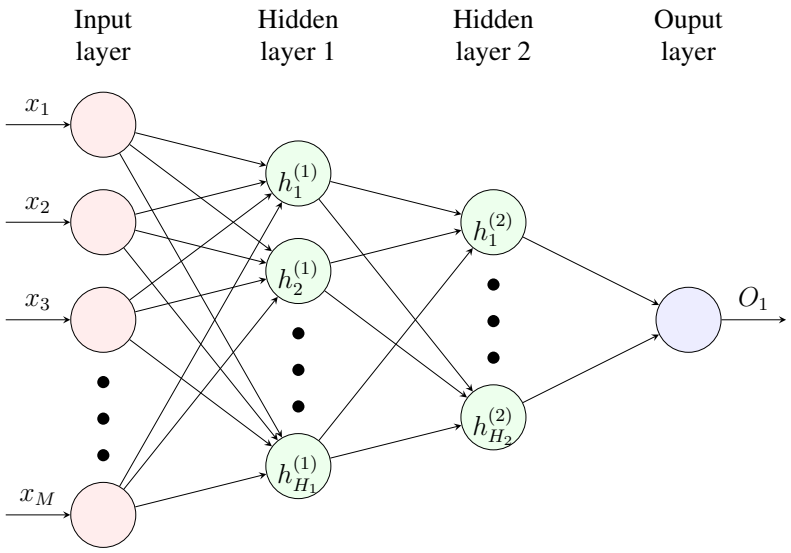
Deep learning

This thesis introduces several advances to deep learning models for applications in financial markets. As deep learning is the main apparatus of this thesis, a literature review is provided in this chapter.

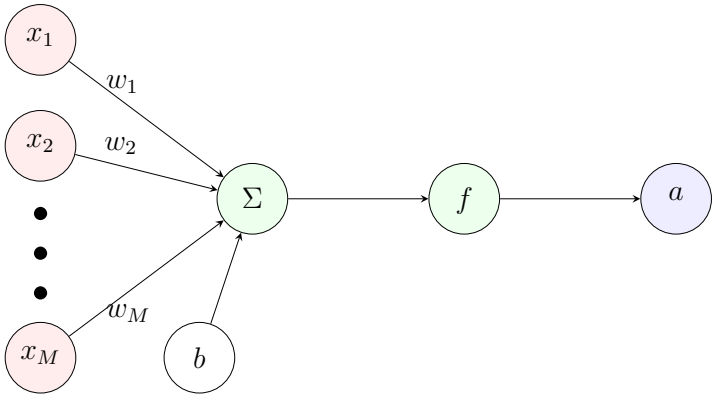
Neural networks are a broad class of high capacity models which were inspired by the biological brain and can theoretically learn any function (a property known as the *Universal Approximation Theorem*; see Hornik et al., 1989; Cybenko, 1989; Goodfellow et al., 2016). In the 19th century, studies of brain activity led to the discovery of neurons that "fire an activation" in response to some activity or stimulus (Bain, 1873; James, 1890). The introduction of the *perceptron* (Rosenblatt, 1958), *multilayer perceptron* (Rosenblatt, 1961) and the *backpropagation* training algorithm (Rumelhart et al., 1986a) heralded the beginning of neural networks. However, it is the increase in computing power and advances in training deeper networks that led to the development of modern neural networks. The term *deep learning* refers to learning with a neural network with many hidden layers, a property that is thought to contribute to the *approximation capacity* of the network (Malach et al., 2021). Further advances in neural network design and training led to the success of deep learning across multiple domains. The rest of this section presents neural network architectures used in this dissertation.

2.1 Feedforward neural networks

The simplest form of neural network, the feedforward network, also known as multilayer perceptrons (MLP), is a subset of neural networks which forms a finite acyclic graph (Goodfellow et al., 2016). There are no loop connections and values are fed forward, from the



(a) Feedforward network



(b) A neuron

FIGURE 2.1: In Figure 2.1(a), an illustration of a fully connected network with two hidden layers. Red, green and blue nodes signify input, hidden and output neurons, respectively. H_ℓ refers to the number of units in ℓ-th layer. Arrows indicate direction of flow for the output value of the respective node. Input to the network has M dimensions. In Figure 2.1(b), a single neuron (depicted as $h_j^{(\ell)}$ in Figure 2.1(a)). Input to the neuron ($\{x_1, x_2, \ldots, x_M\}$) are first linearly transformed and aggregated (depicted as Σ in the centre node). Then, a non-linear activation f is applied, leading to activation value a. More formally, the output of the neuron is $a = f(w_1x_1 + w_2x_2 + \cdots + w_Mx_M + b)$, where $w_M \in \mathbb{R}$ is network weight corresponding to input dimension M and $b \in \mathbb{R}$ is bias.

input layer to hidden layers, and to the output layer. Each layer contains one or more neurons (also called perceptrons). The operation of a neuron involves two steps. First, the input is

linearly combined with layer weights. Then, a non-linear activation function is applied on the result, as depicted in Figure 2.1(b). A feedforward network is also called a fully connected network if every node has every node in the preceding layer connected to it, as illustrated in Figure 2.1(a).

Each layer (denoted $\ell \in \{1, 2, ..., L\}$) consists of H_ℓ units (i.e., dimension of the output of the layer), activation function $f^{(\ell)}$, network weights $\boldsymbol{W}^{(\ell)} \in \mathbb{R}^{n_{\ell-1} \times n_\ell}$ and bias $\boldsymbol{b}^{(\ell)} \in \mathbb{R}^{n_\ell}$. This includes the output layer (denoted $\ell = L$). Input to the ℓ-th layer is the output of the previous layer $\boldsymbol{a}^{(\ell-1)}$. Output of the layer is computed as (in matrix form),

$$\begin{aligned} \boldsymbol{z}^{(\ell)} &= (\boldsymbol{a}^{(\ell-1)})^\mathsf{T} \boldsymbol{W}^{(\ell)} + \boldsymbol{b}^{(\ell)} \\ \boldsymbol{a}^{(\ell)} &= f^{(\ell)} \left(\boldsymbol{a}^{(\ell-1)}; \boldsymbol{W}^{(\ell)}, \boldsymbol{b}^{(\ell)} \right) = f^{(\ell)} \left(\boldsymbol{z}^{(\ell)} \right). \end{aligned} \tag{2.1}$$

where $\boldsymbol{a}^{(\ell)} = (h_1^{(\ell)}, \ldots, h_{H_\ell}^{(\ell)})$, and $\boldsymbol{z}^{(\ell)}$ is termed *weighted input*. In other words, each layer of the feedforward network performs a transformation of the input (output of the previous layer) using the specified activation function. The Universal Approximation Theorem stipulates that activation functions have the following properties (Liew et al., 2016):

(1) The output is non-constant for all ranges of inputs;
(2) Bounded within a range;
(3) Continuous over all values of point c of its domain, where $\lim_{x \to c} f(x) = f(c)$;
(4) Monotonically increasing;
(5) Differentiable everywhere, where $\lim_{c \to 0} \frac{f(x+c) - f(x)}{c}$ exists for all x.

The fifth property is not a requirement of universal approximation but is required for back-propagation learning algorithms. Popular choices of activation functions include (Goodfellow et al., 2016):

- **ReLU**: $f(x) = \max(0, x)$;
- **Sigmoid** (σ): $f(x) = \frac{1}{1+e^{-x}}$;
- **Hyperbolic tangent** ($\tanh$): $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

Sigmoid and tanh are closely related activation functions, as $\tanh(x) = 2\,\sigma(2x) - 1$. tanh resembles the identity function near 0, as the gradient is close to 1 and $\tanh(0) = 0$. Thus, when training a neural network with small activation values (i.e., close to zero), a feedforward network with tanh activation functions behaves like a linear model (Goodfellow et al., 2016). Both sigmoid and tanh are used in recurrent networks, an architecture type that we will discuss in Section 2.5.1. ReLU is a simple piece-wise activation function, where non-linearity is provided by the $\max$ function. Owing to its simplicity, unimpeded gradient flow (when the input value is positive) and ease of computation, ReLU (Jarrett et al., 2009; Nair and Hinton, 2010) quickly became the activation function of choice for many applications and has enabled the breakthrough in training state-of-the-art deep networks (Krizhevsky et al., 2012; Ramachandran et al., 2018). However, ReLU also has several weaknesses (Goodfellow et al., 2016). First, due to the $\max$ function, ReLU is discontinuous at zero. Thus, the derivative is undefined when output is exactly zero. Second, its second derivative is zero almost everywhere and there is no information pass through if the input value is negative. The zeroing out of negative values has led to a phenomenon known as "dead neurons", where a neuron can be "stuck" in a non-active state (either by random initialisation or subsequent gradient updates) as negative bias can cause the affine transformation of the input to be negative. As the neuron is always outputting 0, its weights are not updated by gradient-based learning algorithms (Maas et al., 2013). Note that both sigmoid and tanh suffer from the same problem when input is extremely positive or negative, leading to gradient that is close to zero. Third, a ReLU layer that has non-zero mean activation acts as bias for the next layer (Clevert et al., 2016). A network with many ReLU layers will thus have accumulating biases. This phenomenon is known as *bias shift* which slows down training. Various improvements to ReLU have been proposed, such as exponential linear unit (ELU) (Clevert et al., 2016),

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{Otherwise} \end{cases}, \tag{2.2}$$

and Leaky ReLU (Maas et al., 2013),

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{Otherwise} \end{cases}. \tag{2.3}$$

At the time of writing, a potential candidate for the state-of-the-art in activation function is the *Swish* function (Ramachandran et al., 2018), defined as $f(x) = x \cdot \text{sigmoid}(x)$. The Swish function is a smooth convex function that dips as it approaches 0 from the negative end, turning positive at 0 and asymptotically approaches ReLU at the infinity. The authors have reported improved performance over other variants of ReLU in image recognition tasks. Illustrative activation of each activation functions is shown in Figure 2.2.

Recall that $\boldsymbol{a}^{(0)}$ is the input layer (input data in the form of vector or matrix). For regression problems, the output layer typically has linear activation, where the output is a linear combination of the input and layer weights, and no non-linearity is applied. For classification problems,

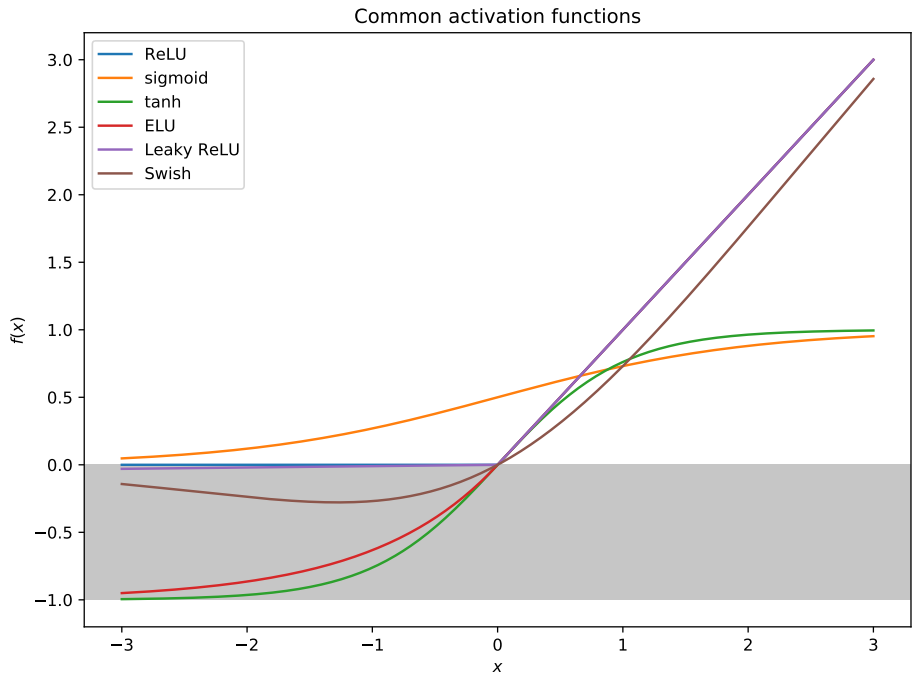


FIGURE 2.2: lllustration of *ReLU*, *sigmoid*, *tanh*, *ELU*, *leaky ReLU* and *Swish*. For ELU, $\alpha = 1$ is used. Note that Leaky ReLU is the same as ReLU where $x > 0$ and marginally negative $x < 0$. ELU and ReLU are also the same when $x > 0$. Region with $f(x) < 0$ is shaded in grey.

popular choices for output layer activation are the *sigmoid* function $\text{sigmoid} : \mathbb{R} \to (0, 1)$ (which can be used to model probability of observing a single class), or *softmax*,

$$\text{softmax}_i(\boldsymbol{x}) = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}},$$

where $\text{softmax}_i(\boldsymbol{x})$ is the predicted probability of observing class $i = 1, \ldots, C$, normalised by the probability of observing every other class.

2.2 Neural network training

In this section, we discuss the learning objective, backpropagation and optimisation methods for neural networks.

Learning, in a theoretical context, involves finding a useful approximation $\hat{G}(x)$ to the function $G(x)$ (Hastie et al., 2020), where G is function of interest and x is input to the function. In this dissertation, we are concerned with finding approximations to $\hat{G}(x)$ using a neural network. For brevity, we drop the layer designation and denote the entire network by F and weight vector set $\boldsymbol{\theta} = \bigcup_{\ell=1}^{L} \{\boldsymbol{W}^{(\ell)}, \boldsymbol{b}^{(\ell)}\}$, where L is the number of layers and $\boldsymbol{\theta}^{(\ell)} = \{\boldsymbol{W}^{(\ell)}, \boldsymbol{b}^{(\ell)}\}$ is the weight set of layer ℓ. We further denote a training instance of a *supervised learning* problem[1] as $(\boldsymbol{x}, y) \sim \mathcal{D}$, where $\mathcal{D}$ is the training set. For regression problems, both inputs $\boldsymbol{x} \in \mathbb{R}^M$ and outputs $y \in \mathbb{R}$ are real values, where M is the dimension of the input (i.e., the number of *features*). A popular choice of loss function is the mean squared error (MSE) (Goodfellow et al., 2016),

$$\mathcal{L}(F(\boldsymbol{X}; \boldsymbol{\theta}), \boldsymbol{y}) = \frac{1}{N} \sum_{j=1}^{N} (y_j - F(\boldsymbol{x}_j; \boldsymbol{\theta}))^2, \tag{2.4}$$

where $N = |\mathcal{D}|$ is number of observations in training set. For classification problems, loss is measured as terms of relative entropy between the data distribution $\text{p}_{\mathcal{D}}$ and distribution

[1] Supervised learning refers to a learning problem where the goal is to predict values of the outputs given some values of inputs, which may be measured or preset (Hastie et al., 2020).

provided by the network $\mathrm{p}_{\text{model}}$,

$$\mathrm{D}_{\mathrm{KL}}(\mathrm{p}_{\mathcal{D}}||\mathrm{p}_{\text{model}}) = \int_y \mathrm{p}_{\mathcal{D}}(y) \log \left(\frac{\mathrm{p}_{\mathcal{D}}(y)}{\mathrm{p}_{\text{model}}(y)} \right) \mathrm{d}y,$$

and is typically implemented as the sum over all training instances,

$$\mathrm{D}_{\mathrm{KL}}(\mathrm{p}_{\mathcal{D}}||F) = \sum_{i=1}^{N} y_i \log \left(\frac{y_i}{F(\boldsymbol{x}_i)} \right),$$

where the neural network $F(\boldsymbol{x}_i)$ is assumed to output an approximation to the empirical distribution $\mathrm{p}_{\mathcal{D}}$. This loss function is called Kullback-Leibler divergence (KL divergence) (Kullback and Leibler, 1951; Goodfellow et al., 2016). A related loss function for classification is *cross-entropy* (Goodfellow et al., 2016),

$$\begin{aligned} \mathrm{H}(\mathrm{p}_{\mathcal{D}}, \mathrm{p}_{model}) &= \mathrm{H}(\mathrm{p}_{\mathcal{D}}) + \mathrm{D}_{\mathrm{KL}}(\mathrm{p}_{\mathcal{D}}||\mathrm{p}_{\text{model}}) \\ \mathrm{H}(\mathrm{p}_{\mathcal{D}}) &= -\int_y \mathrm{p}_{\mathcal{D}}(y) \log \mathrm{p}_{\mathcal{D}}(y)\, \mathrm{d}y, \end{aligned} \tag{2.5}$$

where $\mathrm{H}(x, y)$ is cross-entropy between the distributions of x and y and $\mathrm{H}(x)$ is the *entropy* of the distribution of x. Cross-entropy can be interpreted as the difference between two distributions (the observed data and output of the model). For the purposes of neural network training, it can be seen that cross-entropy and KL divergence are equivalent as the entropy of the data is a constant.

Neural networks are trained using stochastic gradient descent (SGD). SGD is an iterative optimisation algorithm. The algorithm begins with a set of starting weights (weight initialisation is discussed in Section 2.3) and a randomly sampled batch of observations (termed *minibatch*). There are two reasons why a stochastic gradient is used as opposed to the full gradient over the entire training set (in which case it is simply referred to as *gradient descent* or *batch gradient descent*). First, SGD is the main way to train linear models on very large datasets (Goodfellow et al., 2016). Large training sets are typically required to achieve good *generalisation*[2]. Suppose the training set consists of $N = 1\,\text{billion}$ observations. A single

[2]Generalisation refers to prediction or classification performance on unseen data (Goodfellow et al., 2016). For a model to have good generalisation performance, the difference in performance of the model on training and out-of-sample data is minimised.

gradient step that utilises the entire training set will be very computationally expensive. If one minibatch consists of $B = 100$ observations, a single pass through the data will result in $\frac{1\,\text{billion}}{100} = 10\,\text{million}$ gradients steps. SGD relies on updating weights by the *expected gradient*, calculated over a small batch and is thus computationally inexpensive. This greatly reduces the computational cost per weight update (Goodfellow et al., 2016). Secondly, noisy gradients can help the optimiser escape *saddle points*[3] (Ge et al., 2015; Jin et al., 2017; Kleinberg et al., 2018). For highly non-convex functions with many local minima and saddle points, stochastic gradient can be interpreted as working on a smoothed, convolved version of the loss function (Kleinberg et al., 2018). Thus, SGD has greater capacity to traverse the loss surface of highly non-convex loss functions than batch gradient descent.

Next, we formally describe the SGD algorithm. Let $\mathcal{D}$ be the training set and $(\tilde{\boldsymbol{X}}_b, \tilde{\boldsymbol{y}}_b) \sim \mathcal{D}$ be the b-th randomly-drawn minibatch (where $\tilde{\cdot}$ signifies a minibatch; not to be confused with network bias $\boldsymbol{b}$). The basic stochastic gradient update equation is (Goodfellow et al., 2016),

$$\begin{aligned} \boldsymbol{\theta}_{b+1} &= \boldsymbol{\theta}_b - \eta \hat{\nabla} J(\boldsymbol{\theta}_b) \\ J(\boldsymbol{\theta}_b) &= \mathcal{L}(F(\tilde{\boldsymbol{X}}_b; \boldsymbol{\theta}_b), \tilde{\boldsymbol{y}}_b), \end{aligned} \tag{2.6}$$

where the gradient of $J(\boldsymbol{\theta}_b)$ at $\boldsymbol{\theta}_b$ is $\nabla J(\boldsymbol{\theta}_b)$, $\boldsymbol{\theta}_b$ is weight vector set at minibatch b, η is step size (also called learning rate), $\hat{\nabla}$ is stochastic gradient and $\mathcal{L}$ is the loss function. The $J(\boldsymbol{\theta}_b)$ notation is used for brevity. Equation (2.6) describes gradient update at the network level. A *forward pass* of information through the network, from the input layer, through hidden layers and to the output layer is called *forward propagation*. Once training loss of the minibatch is computed, the backpropagation algorithm (Rumelhart et al., 1986a) is used to attribute the loss to weights and bias of each layer by computing the partial derivatives of each layer. Let $c_b = J(\boldsymbol{\theta}_b)$ be the scalar loss for batch b. Then, the partial derivatives with respect to the weights and bias of the output layer are (dropping the batch subscript for clarity),

$$\hat{\nabla}^{(\ell)} = \frac{\partial c}{\partial \boldsymbol{a}^{(\ell)}} \frac{\partial \boldsymbol{a}^{(\ell)}}{\partial \boldsymbol{\theta}^{(\ell)}}. \tag{2.7}$$

[3] A saddle point is a point on the surface of the graph of a function where derivatives in orthogonal directions are all zero, but is not a local extremum of the function (Wainwright and Chiang, 2005).

We denote the *gradient vector set* attributable to the ℓ-th layer as $\hat{\nabla}^{(\ell)}$. The first partial derivative $\frac{\partial c}{\partial \boldsymbol{a}^{(\ell)}}$ is the rate of change of loss with respect to output of the final layer (i.e., $\ell = L$, the derivative of the loss function). For a regression problem where the output is a scalar, $\frac{\partial c}{\partial \boldsymbol{a}^{(\ell)}}$ is also a scalar. If the output of the network is multi-dimensional, then $\frac{\partial c}{\partial \boldsymbol{a}^{(\ell)}}$ is a vector of gradients. The second partial derivative $\frac{\partial \boldsymbol{a}^{(\ell)}}{\partial \boldsymbol{\theta}^{(\ell)}}$ is a gradient vector of the ℓ-th layer's output w.r.t. its weights. For squared loss (Equation (2.4)), the derivative is (a single instance of observation shown),

$$\frac{\partial c}{\partial a_i^{(\ell)}} = \frac{2}{B}(a_i^{(\ell)} - y_i),$$

where B is the size of a minibatch. Similarly, loss attributable to the $\ell - 1$-th layer can be computed using the chain rule,

$$\hat{\nabla}^{(\ell-1)} = \frac{\partial c}{\partial \boldsymbol{a}^{(\ell)}} \frac{\partial \boldsymbol{a}^{(\ell)}}{\partial \boldsymbol{a}^{(\ell-1)}} \frac{\partial \boldsymbol{a}^{(\ell-1)}}{\partial \boldsymbol{\theta}^{(\ell-1)}}.$$

Gradient of the network in Equation (2.6) is the collection of gradients of weights and biases of all layers,

$$\hat{\nabla} J(\boldsymbol{\theta}) = \{\hat{\nabla}^{(1)}, \hat{\nabla}^{(2)}, \ldots, \hat{\nabla}^{(\ell-1)}, \hat{\nabla}^{(\ell)}\}. \tag{2.8}$$

A single gradient update step is the simultaneous update of all layers by the gradient collection scaled by the learning rate, computed using backpropagation.

Equation (2.6) is a recursive algorithm. The network is trained iteratively using each minibatch. Minibatches are drawn from the training set without replacement until the exhaustion of the training set. One cycle through the training set is called an *epoch*. The optimal number of epochs (denoted τ) to train the network is found by monitoring loss on the validation set[4]. Training is stopped when the validation loss decreases by less than a predefined amount, called *tolerance*. This procedure is called *early stopping* (Morgan and Bourlard, 1990; Reed, 1993; Prechelt, 1998; Mahsereci et al., 2017). Algorithm 1 contains the schematics of an early stopping algorithm, adapted from Algorithm 7.1 and Algorithm 7.2 in Goodfellow et al. (2016). Early stopping can be seen as a regularisation technique which limits the optimiser to search in the parameter space near the starting parameters (Sjöberg and Ljung, 1995; Goodfellow et al., 2016), as training is terminated when training no longer decreases

[4] A validation set is a portion of data that is withheld from training and is used for hyperparameter tuning.

validation loss. The early termination of training restricts network weights to be closer to the initial values. In particular, given τ, the product $\eta\tau$ can be interpreted as the effective capacity which bounds reachable parameter space from $\boldsymbol{\theta}_0$ (network weights at the start of training), thus early stopping behaves in a similar way to L_2 regularisation (Goodfellow et al., 2016). In Chapter 3, we propose an online version of early stopping that allows the network to learn "online".

Algorithm 1 Early stopping procedure. Training stops when validation loss does not improve by at least ε for Q iterations.

Require: Maximum iterations $\{\mathcal{T} \in \mathrm{N} \,|\mathcal{T} > 0\}$; tolerance $\{\varepsilon \in \mathbb{R} \,|\varepsilon > 0\}$; patience $\{Q \in \mathrm{N} \,|Q > 0\}$; step size $\{\eta \in \mathbb{R} \,|\eta > 0\}$, training set $\{\boldsymbol{X}_{train}, \boldsymbol{y}_{train}\}$, validation set $\boldsymbol{X}_{test}, \boldsymbol{y}_{test}$

function EARLYSTOPPING($\boldsymbol{\theta}, \boldsymbol{X}_{train}, \boldsymbol{y}_{train}, \boldsymbol{X}_{test}, \boldsymbol{y}_{test}$)
  $\boldsymbol{\theta}_{best} \leftarrow \boldsymbol{\theta}$
  $q \leftarrow 0$
  $J_{best} \leftarrow \infty$
  for $k = 1, ..., \mathcal{T}$ **do**
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\hat{\nabla} J(\boldsymbol{y}_{train}, F(\boldsymbol{X}_{train}; \boldsymbol{\theta}))$
    $J' \leftarrow J(\boldsymbol{y}_{test}, F(\boldsymbol{X}_{test}; \boldsymbol{\theta}))$
    if $J' < J_{best}$ **then**
      $\tau_{best} \leftarrow k$
      $\boldsymbol{\theta}_{best} \leftarrow \boldsymbol{\theta}$
      $J_{best} \leftarrow J'$
    end if
    if J' did not improve by at least ε **then**
      $q \leftarrow q + 1$
      if $q \geq Q$ **then**
        break ▷ Assume convergence
      end if
    else
      $q \leftarrow 0$
    end if
  end for
  return τ_{best}, $\boldsymbol{\theta}_{best}$
end function

Advances in optimisation have allowed deeper and more sophisticated neural networks to be trained. In the rest of this section, we will describe, at a high level, improvements to neural network optimisation.

SGD uses a universal learning rate for all parameters of the network. However, some parts of the network may require less updates than others (e.g., if they learn rarely seen but are highly predictive features). A universal learning rate may lead to over-learning (under-learning) in parts of the network that are frequently (infrequently) updated. Parameter-specific learning rate was introduced in AdaGrad (Duchi et al., 2011), where learning rate for each parameter is scaled by the cumulative sum of the square of past gradients. This has the effect of decreasing learning rate for parameters that are frequently updated.

It was also observed that the magnitude of gradients can be very different across different parts of the network. Hinton and Tieleman (2012) proposed RMSProp, which deals with this problem by dividing the gradients by the square root of the moving average of squared gradients for each weight. This effectively standardises the gradients. Adam (Kingma and Ba, 2015) extends AdaGrad and RMSProp, computing adaptive learning rates based on estimates of first and second moments of the gradients. Rather than updating by the actual gradient $\hat{\nabla} J(\boldsymbol{\theta}_k)$, Adam updates by the exponentially weighted average of past gradients (thereby creating a momentum effect) scaled by the square root of the exponentially weighted average of squared gradients. The exponentially weighted estimates are updated recursively on each iteration, multiplied by a fixed decay rate. Kingma and Ba (2015) demonstrated faster learning (faster decrease in training loss) on benchmark datasets. This was further extended by Dozat (2016), incorporating Nesterov's accelerated gradient (also called Nesterov momentum; Nesterov, 1983) into Adam, named NAdam. The addition of Nesterov momentum has previously been shown to improve regular SGD in hard to optimise problems (Sutskever et al., 2013). At each update step, regular momentum (as used in Adam) is the weighted average between the latest gradient and the weighted average of gradients of the previous step. It was observed that a better quality gradient can be obtained by computing the latest gradient using weights updated by the previous step's weighted average gradients. This was shown to further improve optimisation speed and, at the time of writing, can be considered as the state-of-the-art in neural network optimisation.

2.3 Network weight initialisation

Weight initialisation is an area of active research and is thought to play crucial roles in enabling deeper networks and speeding up network training (Glorot and Bengio, 2010). Early neural networks used sigmoid activation with randomly initialised network weights. This type of network architecture was not conducive to training deep neural networks. Randomly initialised weights are typically drawn from a zero mean distribution (e.g., $\mathrm{U}[\frac{1}{\sqrt{H_\ell}}, \frac{1}{\sqrt{H_\ell}}]$, where H_ℓ is the number of units in the layer and is also called dimension of the layer) (Glorot and Bengio, 2010). Note that dimension of the first layer of a neural network (i.e., the input layer) has the same dimension of the input (i.e., $H_0 = M$). A neural network that uses sigmoid as activation function is prone to an accumulation of bias in a deep network. This is due to mean activation of 0.5 for the sigmoid function. Shift in the distribution of inputs to each hidden layer will cause the sigmoid function to become saturated[5], as illustrated in Figure 2.3. In here, it can be seen that if the expected value of inputs to a sigmoid function is very high or very low, its derivative asymptotically approaches 0. For this reason, Glorot

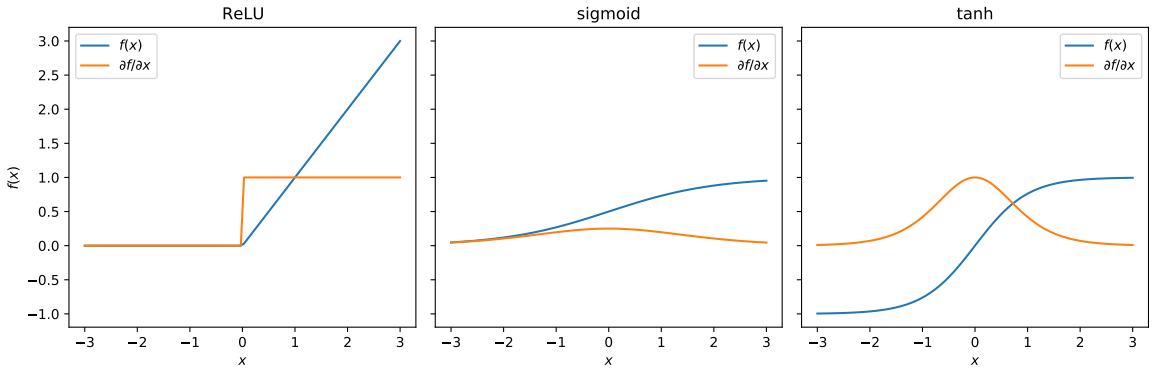


FIGURE 2.3: Illustration of *ReLU*, *sigmoid* and *tanh*, and their respective first derivatives. ReLU has a discontinuity at 0, while both sigmoid and tanh suffer from saturation at extreme values as gradient approaches 0.

and Bengio (2010) recommended against initialising network weights with small random values if the sigmoid function is used — an issue that is partially alleviated with the advent of ReLU, where gradient does not saturate for positive input values. ReLU expedited training

[5] *Saturated sigmoid* occurs when the input to the sigmoid function is close to zero. The sigmoid function outputs values that asymptotically approach zero. Thus, as gradient vanishes, training is impeded.

and allowed deeper networks to be trained[6]. Another contributing factor to slow learning is the decrease in variance of backpropagated gradient as one moves from the output layer backwards (Bradley, 2010; Glorot and Bengio, 2010). This is a deficiency of backpropagation, where loss is progressively attributed to each layer starting from the output. Variance of the gradient diminishes as loss is backpropagated through a deep network. To minimise the decrease in gradient variance attributable to weight initialisation, Glorot and Bengio (2010) proposed *normalised initialisation* (also called *Glorot initialisation* or *Xavier initialisation*), where initial weights are drawn from $\mathrm{U}\left(-\sqrt{\frac{6}{H_{\ell-1}+H_\ell}}, \sqrt{\frac{6}{H_{\ell-1}+H_\ell}}\right)$ or $\mathrm{N}\left(0, \frac{2}{H_{\ell-1}+H_\ell}\right)$ (note that $H_{\ell-1}$ is the input dimension and $M^{(\ell)}$ is the output dimension of the layer). Variance of the distribution with which weights are drawn accounts for the change in dimension between layers. This ensures that variance is constant throughout the network at the initial stage of training[7]. Derivation for the variance parameter was based on linear activation. However, the authors showed that normalised initialisation is still beneficial for networks with sigmoid or tanh activations. He et al. (2015) argued that the assumption of linear activation is invalid for a network with ReLU activation, and proposed an alternative activation (also called *He initialisation*), where weights are drawn from $\mathrm{N}\left(0, \frac{2}{M^{(\ell-1)}}\right)$. The authors demonstrated superior image classification performance using ReLU activation. In sum, weight initialisation strategy can have profound influence on network convergence and the ability to successfully train deep neural networks. Kumar (2017), Glorot and Bengio (2010) and He et al. (2015) have demonstrated the importance of maintaining stable variance of gradient across the layers. To this end, He initialisation is shown to be superior for networks with ReLU activation, while Xavier initialisation can be used for networks with sigmoid and tanh activations.

[6]However, as noted earlier in Section 2.1, this was until network architecture hit the bottleneck of ReLU. ReLU has positive expected activation. This leads to bias shift which slows down training and is addressed by ELU (Clevert et al., 2016) and other variants of ReLU that allow negative activation.

[7]However, it is not until the advent of *residual connections* that "depth barrier" of deep neural network is truly broken.

2.4 Other network architectural considerations

In addition to the aforementioned architectural considerations, modern neural networks typically also use *batch normalisation* (Ioffe and Szegedy, 2015) and *dropout* (Srivastava et al., 2014).

Batch normalisation is the transformation of the output of each hidden layer. For each minibatch with batch size B, input to the batch normalisation layer $\boldsymbol{X} \in \mathbb{R}^{B \times M}$ (i.e., output of the preceding hidden layer $\boldsymbol{a}^{(\ell-1)}$), each instance of the batch is standardised along each dimension,

$$\tilde{x}_{i,m} = \frac{x_{i,m} - \mathrm{E}[\boldsymbol{x}_m]}{\sqrt{\mathrm{Var}[\boldsymbol{x}_m]}},$$

where $x_{i,m}$ is input dimension $m = 1, \ldots, M$ of instance i of the batch and $\boldsymbol{x}_m \in \mathbb{R}^{B \times 1}$ is the column vector of the m-th dimension (batch subscript dropped for legibility). Note that mean and variance are computed per batch, not over the entire training set. The standardised values are then linearly transformed,

$$\boldsymbol{a}_m = \alpha \tilde{\boldsymbol{x}}_m + \beta,$$

where α and β are learnable parameters of the batch normalisation layer that shift and scale the standardised input. Note that a batch normalisation layer is added after a hidden layer and is thus a transformation of the output of the preceding hidden layer. Ioffe and Szegedy (2015) argued that batch normalisation reduced *internal covariate shift*, a phenomenon where the distribution of network activations changes as network parameters are updated during training. The authors argued that by fixing the distribution of activation of each hidden layer, both training speed and accuracy of classifiers improved (in benchmark image classification datasets).

However, subsequent works have disputed the working mechanism of batch normalisation. Various alternative explanations have been proposed. Santurkar et al. (2018) found that batch normalisation induces a smoother loss function surface and gradients. This surface is easier to traverse by the optimiser, allowing for faster training. Approaching the problem from a classical optimisation perspective, Kohler et al. (2019) argued that batch normalisation

works by splitting the optimisation task into optimising length and direction of the parameters separately. This was motivated by a related technique, *weight normalisation*, which normalises neural network weights to separate lengths from their directions (Salimans and Kingma, 2016). Kohler et al. (2019) demonstrated that this reparameterisation provided faster convergence. In sum, both alternative explanations relate batch normalisation to aspects of optimisation.

Dropout (Srivastava et al., 2014) involves randomly dropping out units (along with their connections) throughout the network during training. Implementation is straightforward. Each forward pass through the dropout layer (typically placed after a hidden layer) is multiplied by a 1-and-0 mask, where the zeros are randomly drawn according to the pre-specified dropout rate. Each time random dropout is applied, the masked network is a subnetwork of the full network. At inference time, the full network is used (without masking) where the weights are the result of averaging over many different subnetworks. Thus, dropout can be interpreted as an inexpensive way of ensembling within the network. Dropout was shown to improve neural network performance across a range of vision, speech recognition, document classification and computational biology tasks (Srivastava et al., 2014).

2.5 Specialised network architectures

2.5.1 Recurrent neural networks

RNN (Rumelhart et al., 1986b; Goodfellow et al., 2016) and its variants have been the workhorses of speech and languages (Chiu et al., 2018) and other sequential applications. RNNs differ from feedforward networks in that they are (directed or undirected) graphs along a temporal dimension. RNNs maintain a vector of hidden states that encode the observed sequence and are recursively updated as new observations become available. Thus, recurrent networks can be thought of as sharing parameters for each temporal step. A recurrent network can be configured to have different feedback connections, with the most basic being recurrence of the hidden state, as illustrated in Figure 2.4. In here, the hidden state of each temporal step is recursively updated. Input x_t is first combined with the hidden state of the previous step,

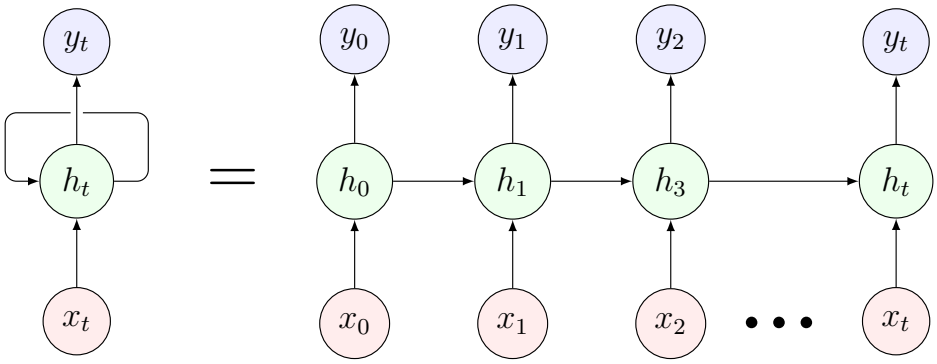


FIGURE 2.4: **Left**: An illustration of a recurrent layer with cyclical connection back to itself for each temporal step. After each activation, the hidden state is carried forward to the next temporal step. **Right**: The unfolded *computational graph* of the recurrent layer for each temporal step. In here, x, h and y signify input, hidden state and output, respectively.

resulting in an updated hidden state,

$$\boldsymbol{h}_t = f(\boldsymbol{x}_t^\mathsf{T}\boldsymbol{W}_x + \boldsymbol{h}_{t-1}^\mathsf{T}\boldsymbol{W}_h + \boldsymbol{b}_h),$$

where $\boldsymbol{W}_x$ are weights for the input, and $\boldsymbol{W}_h$ and $\boldsymbol{b}_h$ are weights and bias for the previous hidden state. The updated hidden state is then used to compute output of the layer,

$$\boldsymbol{y}_t = g(\boldsymbol{h}_t^\mathsf{T}\boldsymbol{W}_o + b_o),$$

where g, $\boldsymbol{W}_o$ and b_o are activation function, weights and bias for converting the hidden state to output.

In the example presented in Figure 2.4, the recurrent connection is hidden-to-hidden. The recurrent connection can also be output-to-hidden. However, such recurrence is less powerful as it requires that the output units capture all past information that the network uses to make predictions (Goodfellow et al., 2016). There are advantages in using specialised network architectures for sequential problems (Goodfellow et al., 2016). First, the same recurrent network can be used to model sequences of varying length, a type of input that is common in NLP problems where sentences and texts are of arbitrary length. A feedforward network requires a fixed structure and can only model sequences of pre-defined length. Second, consider a classification problem where sequences of arbitrary length K map to C different classes. If one were to model such a problem using feedforward networks, the number of

parameters in the model will scale by $\mathrm{O}(C^K)$. However, as parameters are shared across temporal steps in a recurrent network, the number of parameters in a RNN is $\mathrm{O}(1)$ with respect to sequence length (Goodfellow et al., 2016). This is also implicitly assuming that the conditional distribution of y_{t+1} is stationary given y_t.

Due to its flexibility in modelling sequences (both input and output) of arbitrary length, recurrent network architectures can be designed to suit many different applications. The different architecture types are illustrated in Figure 2.5. The one-to-one architecture corresponds to a

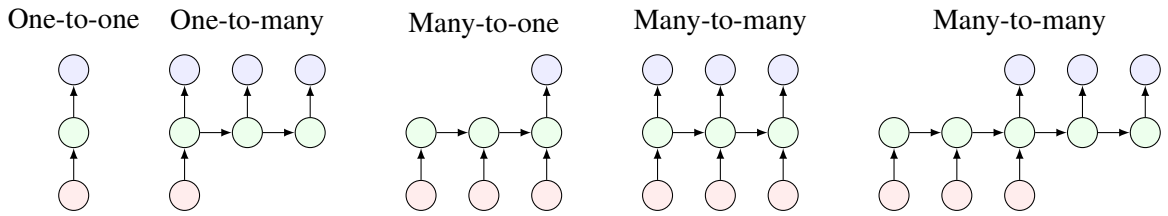


FIGURE 2.5: Types of recurrent architectures. Red, green and blue nodes signify input, hidden (recurrent) and output neurons, respectively. Each type differs by the type of association between input, recurrence and output.

conventional use of recurrent networks, such as weather forecasting (Cebeci, 2019). Similar to a pooled regression, the RNN is presented with one observation of input and makes a forecast in a one-to-one manner on each time step. The one-to-many architecture receives an encoding of a task and generates a sequence. This type of architecture have been used for music synthesis (Jaques et al., 2017). Many conventional time-series forecasting applications can be classified as many-to-one, where the network observes a sequence of inputs and makes a classification or prediction. Financial time series forecasting problems such as Li et al. (2017) and those in Chapter 4 are examples of many-to-one networks. There are two types of many-to-many architectures — one where inputs and outputs are in sync along the time dimension; and where outputs lag inputs by an offset. The former is used in applications such as named entity recognition (Katiyar and Cardie, 2018), while the latter is used in machine translation, such as the *Seq2seq* model (Sutskever et al., 2014). In Seq2seq, the encoder part of the recurrent network converts the sentence to be translated into a latent representation, which is then re-generated by the decoder portion of the network in the target language.

However, despite their success, recurrent networks are notoriously difficult to train (Pascanu et al., 2013; Bai et al., 2018). In an unrolled computational graph, recurrent networks can be considered as neural networks with unlimited depth. Modelling long term dependencies is especially challenging as gradients propagated through many layers tend to explode or vanish (Bengio et al., 1994; Pascanu et al., 2013). Even if we assume that gradients remain stable, long-term dependencies are difficult to maintain as recursively multiplied Jacobians lead to exponentially smaller weights given to long-term interactions (Goodfellow et al., 2016). To address this, LSTM networks (Hochreiter and Schmidhuber, 1997; Goodfellow et al., 2016) were introduced with a *memory cell* containing a self-loop that allows gradients to flow over long durations. The basic schematics of a LSTM is illustrated in Figure 2.6. *Gating* is central

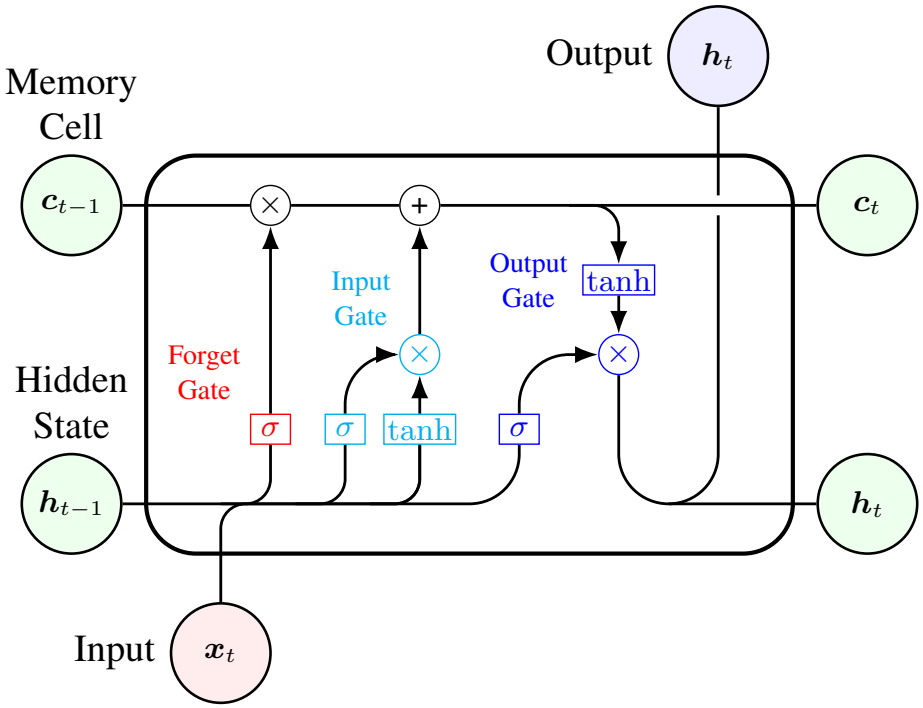


FIGURE 2.6: An illustration of a long short-term memory "cell". The forget gate controls whether information stored in the memory cell is retained. The input gate controls whether the new input is stored into the memory cell. The output gate incorporates information in the memory cell into the output.

to the LSTM, which consists of three gates: the *forget gate*, the input gate (also called update gate) and the output gate. The forget gate controls how much memory from the previous time

step is retained[8],

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}^{(f)}\boldsymbol{x}_t + \boldsymbol{U}^{(f)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(f)}),$$

where $\boldsymbol{f}_t$ is the forget gate unit at time step t, σ denotes the sigmoid function, $\boldsymbol{W}$, $\boldsymbol{U}$ and $\boldsymbol{b}$ are network weights applied the input and hidden state, and bias, respectively. The forget gate combines the input and hidden state into a $\boldsymbol{f}_t \in (0,1)^H$ vector, where H is the dimension of the LSTM unit. The input gate $\boldsymbol{i}_t \in (0,1)^H$ controls how much information is incorporated into the memory cell,

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}^{(i)}\boldsymbol{x}_t + \boldsymbol{U}^{(i)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(i)}).$$

The input and hidden state are transformed into a cell input vector $\tilde{\boldsymbol{c}}_t \in (-1,1)^H$,

$$\tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}^{(c)}\boldsymbol{x}_t + \boldsymbol{U}^{(c)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(c)}).$$

Memory carried forward from $t-1$ is modulated by the forget gate and the cell input vector is modulated by the input gate. The two are then combined to form the updated memory cell state,

$$\boldsymbol{c}_t = \boldsymbol{f}_t \otimes \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \otimes \tilde{\boldsymbol{c}}_t,$$

where $\otimes$ is the Hadmard product (i.e., element-wise multiplication). The output gate $\boldsymbol{o}_t \in (0,1)^H$ controls the amount of information from the memory cell that is incorporated into the hidden state,

$$\begin{aligned}\boldsymbol{o}_t &= \sigma(\boldsymbol{W}^{(o)}\boldsymbol{x}_t + \boldsymbol{U}^{(o)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(o)}) \\ \boldsymbol{h}_t &= \boldsymbol{o}_t \otimes \tanh(\boldsymbol{c}_t).\end{aligned} \tag{2.9}$$

The updated hidden state $\boldsymbol{h}_t$ is the output of the LSTM cell at t. Advances in LSTM (and other recurrent networks) had enabled major breakthroughs in machine translation (Sutskever et al., 2014), speech recognition (Graves et al., 2013) and handwriting recognition (Graves et al., 2009).

[8] In this section, $\boldsymbol{f}$ is used to denote the forget gate, not to be confused with activation function f used throughout this dissertation.

2.5.2 Temporal convolutional networks

CNNs are specialist networks for data that has a known, grid-like topology (Goodfellow et al., 2016). Examples include time-series of fixed length (a 1-D grid) or images (2-D grid). CNN has achieved human-like accuracy in image recognition tasks (Krizhevsky et al., 2012; Szegedy et al., 2015; Schroff et al., 2015). In this section, the mathematical operation *convolution* and a CNN-derived network achitecture known as temporal convolutional network (TCN)[9] are discussed. We refer readers to Goodfellow et al. (2016) for a comprehensive discussion on general CNNs.

Convolution is the modification of one function by another, producing a third function. More formally, the convolution of functions[10] f and g results in function o indexed by i (Goodfellow et al., 2016),

$$o(i) = \int \mathrm{f}(j)\,\mathrm{g}(i-j)\,\mathrm{d}j \tag{2.10}$$
$$o(i) = (\mathrm{f} * \mathrm{g})(i),$$

where $*$ is the convolution operator. To put it more concretely, suppose $\mathrm{g}(\cdot)$ in Equation (2.10) is a weighting function (e.g., a probability density function). Then, $\mathrm{f} * \mathrm{g}$ results in a function that returns the weighted average of f by g. In this case, f is the *input* and g is the *kernel* (also known as *filter*). In image recognition, the input is typically 2-D with discrete index (Goodfellow et al., 2016),

$$O(i,j) = \sum_m \sum_n \mathrm{F}(m,n)\,\mathrm{G}(i-m, j-n).$$

An example of convolution is illustrated in Figure 2.7. This example uses *one* 2-D kernel, with *kernel size* of 2. Size of the kernel relates to how much local information is used each time the kernel is convolved with the input. Each value in output O is the result of a sum-product of four adjacent values (2×2) of input F and kernel G. For time-series input, a 1-D kernel (e.g., with dimensions 2×1) is used which slides along the sequence during convolution. Number

[9] Also known as *dilated convolutional networks*.

[10] Note that in this section, we use (upright) f and g symbols and their capitalised counterparts to illustrate the workings of convolution. Not to be confused with the meaning of f (activation function) in the rest of the thesis.

of kernels[11] is analogous to number of units of a hidden layer in a feedforward network, and dictates the output dimension of the convolution layer. Suppose that number of kernels is 3. Then, G has dimensions $2 \times 2 \times 3$ (three 2×2 kernels stacked together) and convolution is performed three times, resulting in $O \in \mathbb{R}^{3\times3\times3}$. In Figure 2.7, size of the resultant matrix shrank by 1. Various zero-padding strategies exist if the desire is to maintain the dimensions of the output, such as by padding the surroundings of the input with 0.

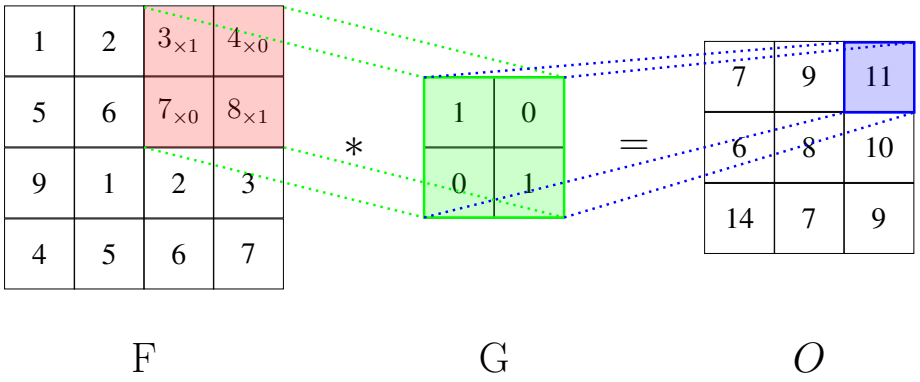


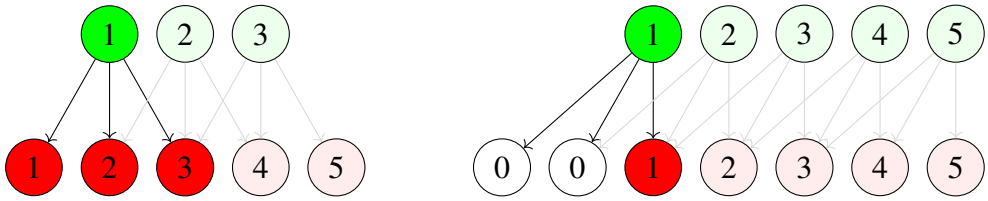
FIGURE 2.7: An illustration of a convolution operation. Input F is convolved with kernel G, producing O. The highlighted cells is a single convolution operation, yielding $3 \times 1 + 4 \times 0 + 7 \times 0 + 8 \times 1 = 11$. The same operation is repeated horizontally and vertically.

Given that regular feedforward networks are underpinned by universal approximation properties, one may wonder why CNNs are required for image recognition problems. Suppose an image has dimensions of 1000×1000 pixels and three colour channels per pixel. The resulting input tensor contains 3 million values. Next, suppose a feedforward network is used for this problem and the first hidden layer contains 500 units. The resultant number of connections at the first hidden layer is $3\,\text{million} \times 500 = 1.5\,\text{billion}$. Such a parameter space is likely infeasibly large to be trained efficiently. CNN solves this problem by utilising three important concepts: *sparse interactions*, *parameter sharing* and *equivariant representations* (Goodfellow et al., 2016). Sparse interactions (also known as *sparse connectivity*) is achieved by using a kernel that is much smaller than the input dimensions. Memory requirement and efficiency can be significantly improved if small, meaningful features can be detected out

[11] Also called number of filters or number of channels. The latter due to it being analogous to colour channels in image applications.

of thousands of pixels. The same small (in dimensions) kernel is used throughout the layer, leading to parameter sharing. This encourages the network to learn a kernel that can extract useful features that are common across the input and greatly reduces the parameter space. This also leads to equivariance, where the same feature appearing in different location in an image (or at different times in a time-series) will lead to the exact same output, just at a different location (different point in time). Thus, only one representation needs to be learned and can detect the same feature anywhere in the input.

As noted in Section 2.5.1, recurrent neural networks are difficult to train and suffer from exploding/vanishing gradient that renders modelling long term dependencies challenging. Sequential processing also makes scaling up computation to take advantage of recent advances in parallel hardware acceleration difficult (Bai et al., 2018). Recently, convolutional networks have been shown to be competitive against recurrent network-based architectures in a range of sequence learning tasks (van den Oord et al., 2016; Kalchbrenner et al., 2016; Bai et al., 2018). van den Oord et al. (2016) proposed a novel CNN architecture, known as *WaveNet* (also known as *dilated convolutional network*), and showed that it generated more naturally sounding speech than LSTM. The main components of WaveNet consist of dilated convolution layers with causal padding (termed *causal convolutions* in van den Oord et al., 2016) and residual connections (He et al., 2016). In standard convolution, the filter convolves with a symmetrical number of values to the left and right of the centre of the filter (as depicted in Figure 2.7). This is problematic in time-series as values on the right represent future observations that would not have been known at the centre. This is mitigated with causal padding which shifts the sequence such that the rightmost value of each convolution corresponds to the centre of the original sequence. Causal padding refers to appending $d \times (k-1)$ zeros to the beginning of the sequence, where d is dilation rate and k is kernel size. This is illustrated in Figure 2.8. In standard convolution (Figure 2.8(a)), the illustration depicts the first convolution of a kernel of size 3 with the first 3 elements of the input sequence. The first output value uses up to the third value from the input sequence and is thus "looking ahead in time." In causal convolution (Figure 2.8(b)) and assuming a dilation rate of 1, $1 \times (3-1) = 2$ zeros are padded to the beginning (left) of the input sequence. The kernel is convolved with the first three elements of the padded sequence (two leading zeros and the first element of the original sequence),



(a) Standard convolution (b) Causal convolution

FIGURE 2.8: In the top row, green nodes denote outputs after convolution with a kernel of size 3, numbered by the n-th convolution operation (striding from left to right). In the bottom row, red nodes denote the input sequence, numbered by the n-th element in the sequence. White nodes denote zero padding. Darker colours denote nodes undergoing convolution. In Figure 2.8(a), the 3rd convolution operation will access the 4th element in the input.

producing the first output value. Thus, the output contains no look ahead information. Note that due to the two padded zeros, causal convolutions produce a longer output sequence than standard convolution with no padding. Standard convolution can also have zeros padded to both beginning and end of the sequence to maintain length of the sequence if required. Dilated convolution (also called *à trous convolution*) expands the *receptive field* of the node by skipping input nodes (i.e., inserting zeros to the kernel). For example, suppose that a network with dilation rates are multiples of 2 and kernel size of $k = 3$, as depicted in Figure 2.9. The first convolution layer in Figure 2.9 has dilation rate of $2^0 = 1$, the second is $2^1 = 2$, third is $2^2 = 4$, and so on. In each dilated convolution layer, $d - 1$ zeros are added in between each value in the kernel (dilation rate of 1 equates to no dilation). For example, the third convolution layer in Figure 2.9 has $2^2 - 1 = 3$ zeros inserted between each value of the kernel, with receptive field that spans the entire sequence of 15 elements. Values that are multiplied by 0 in the kernel (depicted using dashed grey connections) do not contribute to the final result. Thus, the network can cover sequences of arbitrary length by stacking multiple dilated convolution layers and having $d \geq 2$.

Bai et al. (2018) proposed the TCN as a generalisation of WaveNet — without conditioning, context stacking or gated activation. However, it retains the key ingredient of dilated causal convolution layers organised into *residual blocks*, first introduced in ResNet (He et al., 2016) and is Illustrated in Figure 2.10. Each residual block in Figure 2.10 consists of four layers — dilated causal convolution, batch normalisation, spatial dropout (Tompson et al., 2015) and

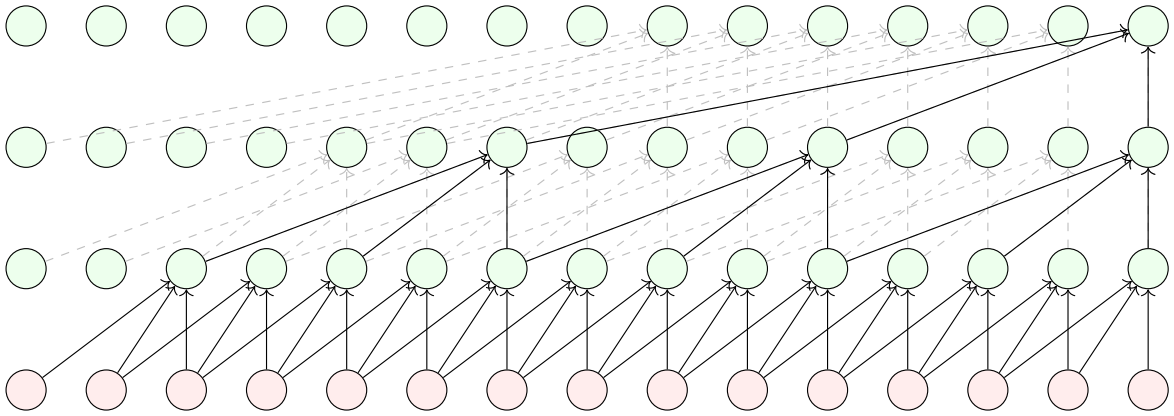


FIGURE 2.9: Causal dilated convolutions with kernel size of 3 and dilation rate of multiples of 2. Red nodes represent the input sequence and the two green layers represent convolution layers. Nodes without a connection or are connected with a dashed grey line are multiplied by zeros in the kernel. The top (output) node reaches the entire input sequence without using every node in the intermediate layer. The receptive field can be varied by increasing or decreasing the number of dilated convolutional layers.

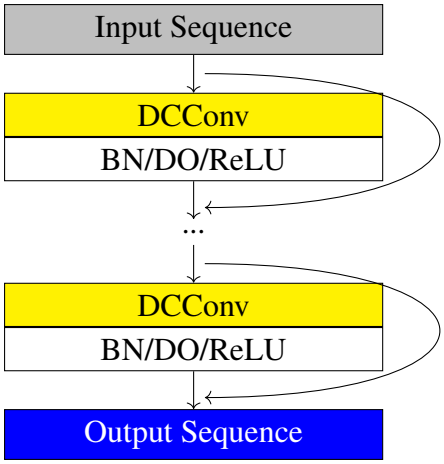


FIGURE 2.10: Illustration of a TCN. Residual blocks correspond to green layers in Figure 2.9 and consist of skip connection, dilated causal convolution (abbrev. *DCConv*), batch normalisation , spatial dropout and rectified linear unit activation layers (abbrev. *BN/DO/ReLU*).

ReLU. Spatial dropout is similar to the regular dropout (introduced in Section 2.4). However, instead of randomly dropping out observations, spatial dropout randomly drops an entire dimension. The output sequence of a residual block is first added to the input sequence (of the said residual block) via skip connection, then feed into the next residual block as

input sequence. Bai et al. (2018) showed that TCN was superior to LSTM across a range of sequence modelling tasks. The authors also noted five advantages of TCN (along with two disadvantages) over recurrent networks, most notably, parallelism and stable gradients. Stable gradients allow convolutional networks to access longer history (one year of daily prices is over 250 observations) than recurrent networks. TCN provides another useful tool for modelling time-series data.

2.5.3 Autoencoders

An autoencoder is an unsupervised neural network that learns a latent representation of the input (Goodfellow et al., 2016). First introduced in LeCun (1987); Bourlard and Kamp (1988); Hinton and Zemel (1993), the autoencoder consists of two parts, an *encoder* that encodes the input into a latent representation $\boldsymbol{h} = f(\boldsymbol{x})$ and a decoder that reconstructs the input $\hat{\boldsymbol{x}} = g(\boldsymbol{h})$. Each of encoder and decoder contains one or more hidden layers. Typically, dimensions of $\boldsymbol{h}$ is chosen to be significantly smaller than the dimension of $\boldsymbol{x}$ such that the autoencoder learns a useful representation that summarises the input, as illustrated in Figure 2.11. Autoencoder learns the latent representation of input via hidden layers, and is equivalent to PCA when it has precisely one linear hidden layer (Baldi and Hornik, 1989; Le et al., 2018). Thus, traditional applications of autoencoders are in dimensionality reduction and generative modelling.

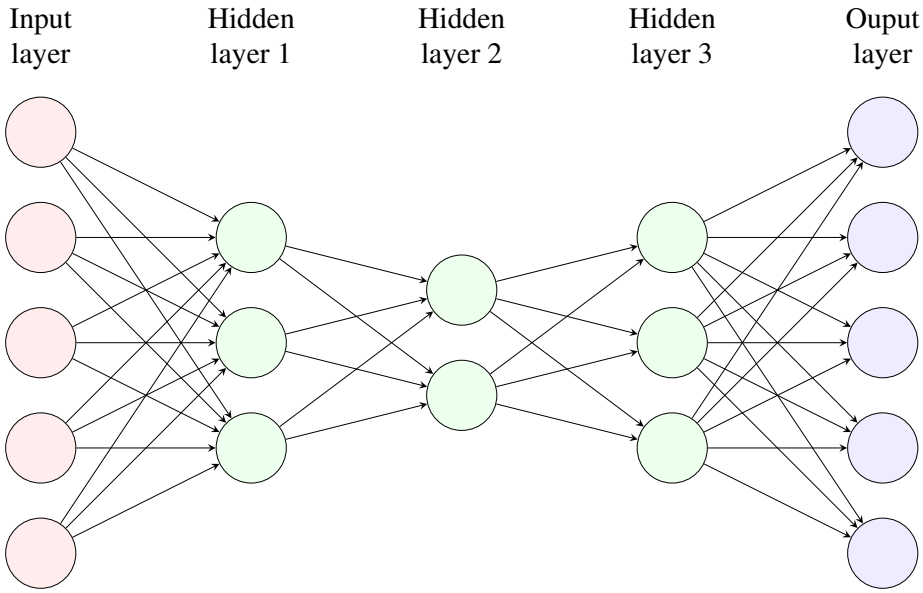


FIGURE 2.11: An illustration of an autoencoder. The input $\boldsymbol{x} \in \mathbb{R}^5$ is converted into a latent representation $\boldsymbol{h} \in \mathbb{R}^2$ using 2 hidden layers. $\boldsymbol{h}$ is then converted back into the (reconstructed) input.

CHAPTER 3

Time-varying neural network for stock return prediction

The motivating application of this chapter is in predicting cross-sectional stock returns in a portfolio context and has been discussed in Section 1.6.1. We propose the OES algorithm and show that a neural network trained using this algorithm can track a function changing with unknown dynamics. We provide a regret-bound for the algorithm and show that the worst-case tracking performance of the algorithm is bound by the time-variations of the DGP. We compare the proposed algorithm to current approaches on predicting monthly U.S. stock returns and show its superiority. Using this algorithm, we also provide evidence that supports recent findings in finance literature that suggest financial markets are time-varying. These contributions have resulted in the following publication:

Steven Y. K. Wong, Jennifer S. K. Chan, Lamiae Azizi, Richard Y. D. Xu, "Time-varying neural network for stock return prediction," *Intelligent Systems in Accounting, Finance and Management*, 29(1), 3–18, 2022.

3.1 Introduction

At every interval, an investor forecasts expected return of assets and performs security selection. This problem is closely related to *asset pricing* (as discussed in Section 1.3 and Appendix A1), cross-sectional predictions and time-varying DGP. To reiterate, literature has found hundreds of factors (Harvey et al., 2016) that can predict stock returns. Literature has also documented evidence of time-variability of the DGP (e.g., Pesaran and Timmermann, 1995; Bossaerts and Hillion, 1999). Both the true functional-form of the DGP and its time-varying dynamics are not known.

To address this, we simultaneously address both challenges by proposing the Online Early Stopping (OES) algorithm, which allows neural networks to adapt to a time-varying function. Our problem is characterised by information release over time and iterative decision making. This shares a remarkably similar setup with *online optimisation*[1]. Optimisation in this context is called *online* as decisions are made with past information but not the future. As noted in Section 2.2, one of the hyperparameters in batch (offline) neural network training is the number of optimisation epochs τ. In OES, we propose to treat τ as a learnable parameter that varies over time (t), as τ_t, and is recursively estimated over time. We provide τ_t with a new meaning — a regularisation parameter that controls the amount of update neural network weights receive as new observations are revealed. Thus, if consecutive cross-sectional observations are very different then we would expect τ_t to be relatively small and the neural network is prevented from overfitting to any one period. Conversely, a slowly changing function will have a high degree of continuity and we would expect the network to fit more tightly to each new observation. Using this training algorithm, a neural network can adapt to changes in the DGP over time. For practitioners, we show that a neural network trained with OES can be a powerful prediction model and a useful tool for understanding the time-varying drivers of returns.

Neural network training is an optimisation problem (Evens et al., 2021). We draw on concepts in online optimisation to provide a performance bound that is related to the variability of each period. We do not assume any time-varying dynamics of the underlying function, a typical approach in online optimisation[2]. The benefit of this approach is that it can track any source of variability in the underlying function, including macroeconomic, arbitrage-induced, market condition-induced, or other unknown sources. For instance, Lev and Srivastava (2019) suggested that the negative return to the value factor was related to diminishing relevance of book equity as an accounting measure. Such drivers would not have been captured by the macroeconomic approach in Gu et al. (2020). Nonetheless, we acknowledge that a limitation of our approach is the difficulty in explaining the source of variability. Attributing the source

[1] We formally describe the context in Section 3.2.1.

[2] Online convex optimisation is typically formulated as a game against an adversary, where solutions are designed to provide worst-case performance guarantees. See (Shalev-Shwartz, 2012) for an overview.

requires measuring interactions between features and sources, of which some may be difficult to quantify. Existing model explanation methods (e.g., Local Interpretable Model-Agnostic Explanations; Ribeiro et al., 2016) attribute model output to input through approximations. Thus, the addition of interaction terms between each feature to each source could lead to a substantial increase in dimensionality.

We provide two evaluations of OES: 1) a simulation study based on a dataset simulated from a non-linear function evolving under a random-walk; 2) an empirical study of U.S. stock returns. The empirical study is based on Gu et al. (2020), who compared several machine learning algorithms for predicting monthly returns of all U.S. stocks. The best performing machine learning model in Gu et al. (2020) as measured by R^2 (this measure is discussed in more detail in Section 3.2.1) was a three-layer fully connected neural network with ReLU activation, dropouts and batch normalisation. The network was trained annually using all available data up to t. The same network is then used for monthly prediction over the next 12 months, upon which an additional 12 months of new data is added to the training set and training is repeated. The majority of the dataset were made available to the public and are used in this chapter. We note that the setup in Gu et al. (2020) is suboptimal for our portfolio selection problem for three reasons. Firstly, (raw) monthly stock returns contain characteristics that complicate the forecasting problem, such as outliers, heavy tails, and volatility clustering (Cont, 2001). These characteristics are likely to impede a predictor's ability to learn. Secondly, the dataset in Gu et al. (2020) contains stocks with very low market capitalisation, are illiquid, and are unlikely to be accessible by institutional investors. Thirdly, at the individual stock level, forecasting stocks' excess returns over risk free rate also encompasses forecasting market excess returns. As practitioners are typically concerned with relative performance between stocks[3], the market return component adds unnecessary noise to the problem of relative performance forecasting. Thus, in addition to comparison with Gu et al. (2020), we also present results based on a more likely use case by practitioners, by excluding stocks with very low capitalisation and forecasting cross-sectionally standardised excess returns. We show that forecasting performance significantly improved based on this

[3]In the simplest form, a long-only investor will hold a portfolio of the top ranked stocks and a long-short investor will buy top ranked stocks and sell short bottom ranked stocks. Thus, relative performance is relevant to practitioners.

re-formulation. We propose to measure performance using the information coefficient (IC), a widely applied performance measure in investment management (Ambachtsheer, 1974; Grinold and Kahn, 1999; Fabozzi et al., 2011b). OES achieves an IC of $4.58\,\%$ on the U.S. equities dataset, compared to $3.82\,\%$ under an expanding window approach in Gu et al. (2020). For context, an IC of $5\,\%$ in predicting cross-sectional returns is considered as "good" by practitioners (Grinold and Kahn, 1999).

A summary of our contributions in this chapter is as follows:

- We propose the OES algorithm which allows a neural network to track a time-varying function. OES can be applied to existing network architectures and requires significantly less time to train than the expanding window approach in Gu et al. (2020). In our tests, OES took 1/7 the time to train and predict as compared to the expanding window approach of expanding window neural network (EWNN)[4]. This finding has a practical implication as practitioners wishing to employ deep learning models have limited time between market close and next day's open to generate features and train new models, which is made worse if an ensemble is required.
- We show that firm features exhibit time-varying importance and that the model changes over time. We find that some prominent features, such as market capitalisation (the size effect) display declining importance over time. This finding is consistent with McLean and Pontiff (2016) and highlights the importance to consider time-varying models.
- We find that firm features, in aggregate, experience a fall in importance in predicting cross-sectional returns during market distress (e.g. Dot-com bubble in 2000–01). The importance of sector dummy variables (e.g., technology and oil stocks) rose over the same period, suggesting the importance of sectors is also time-varying. Our analysis indicates that sectors have an important role in predicting stock returns during market distress. We expect this to be especially true if market stress impacts certain sectors more than others, such as travel and leisure stocks during a pandemic.

[4]Training performed on AMD Ryzen™ 7 3700X, Python 3.7.3, Tensorflow 1.12.0 and Keras 2.2.4.

- Using a subuniverse that is more accessible to institutional investors (by excluding microcap stocks), we show that OES exhibits superior predictive performance. We find that the mean correlation between predictions of OES and EWNN is only $35.9\,\%$ and the monthly correlation is lowest immediately after a shock (e.g., recession). We attribute this to OES adapting to the recovery which manifests as lower drawdown post the Global Financial Crisis.
- We show that an ensemble formed by averaging the standardised predictions of the two models exhibits the highest IC, decile spread and Sharpe ratio. Thus, practitioners may choose to deploy both models in a complementary manner.

In the rest of this chapter, we denote the algorithm of Gu et al. (2020) as expanding window neural network (EWNN) and our proposed Online Early Stopping algorithm as OES. This chapter is organised as follows. Section 3.2 defines our cross-disciplinary problem and provides an overview of online optimisation. Section 3.3 outlines our main contribution of this chapter — the proposed OES algorithm which introduces time-variations to the neural network. Simulation results are presented in Section 3.4, which demonstrates the effectiveness of OES in tracking a time-varying function. An empirical study on U.S. stock returns is outlined in Section 3.5. Finally, Section 3.6 discusses the empirical finance problem and concludes the chapter with some remarks.

3.2 Preliminaries

3.2.1 Problem setup

The setup of the problem in this Chapter largely follows that of the cross-sectional prediction problem defined in Section 1.6.1 and is repeated in here for convenience.

Similar to a classical online learning setup, a player iteratively makes portfolio selection decisions at each period. We call this iterative process *per interval training*. There are N stocks in the market, each with M features, forming input matrix $\boldsymbol{X}_t \in \mathbb{R}^{N\times M}$ at time $t = 1, \ldots, T$. The i-th row in $\boldsymbol{X}_t$ is feature vector $\boldsymbol{x}_{t,i}$ of stock i. To simplify notations, we define

return of stock i as the percentage return over the next period, i.e., $y_{t,i} = (p_{t+1,i} + d_{t+1,i})/p_{t,i} - 1$, where $p_{t,i}$ is price at time t and $d_{t,i}$ is dividend at t if a dividend is paid, and zero otherwise. In other words, the player uses information up to time t (e.g., earnings-to-price ratio at t) to predict a stock's return over t to $t+1$. Player predicts stock returns $\hat{\boldsymbol{y}}_t \in \mathbb{R}^N$ by choosing $\boldsymbol{\theta}_t \in \Theta$, which parameterises prediction function $\{F : \mathbb{R}^{N \times M} \mapsto \mathbb{R}^N; \hat{\boldsymbol{y}}_t = F(\boldsymbol{X}_t; \boldsymbol{\theta}_t)\}$. Market reveals $\boldsymbol{y}_t$ and, for regression purposes, player incurs squared loss,

$$J_t(\boldsymbol{\theta}_t) = \frac{1}{N} \sum_{i=1}^{N} (y_{t,i} - \hat{y}_{t,i})^2.$$

This general iterative portfolio selection setup is shared across Chapter 4, 5 and this chapter. The true function $\Phi_t : \mathbb{R}^{N \times M} \mapsto \mathbb{R}^n$ drifts over time and is approximated by F with time-varying $\boldsymbol{\theta}_t$. Player's objective is to minimise loss incurred by choosing the best $\boldsymbol{\theta}_t$ at time t using observed history up to $t-1$. Both the functional form and time-varying dynamics of Φ_t are not known Hence a neural network is used to model the cross-sectional relationship at each t and the time-variability is formulated as a network weights tracking problem. The loss function J_t verifies the same assumptions adopted in Aydore et al. (2019), which are:

- J_t is bounded: $|J_t| \leq D; D > 0$,
- J_t is L-Lipschitz: $|J_t(\boldsymbol{a}) - J_t(\boldsymbol{b})| \leq L \|\boldsymbol{a} - \boldsymbol{b}\|; L > 0$,
- J_t is β-smooth: $\|\nabla J_t(\boldsymbol{a}) - \nabla J_t(\boldsymbol{b})\| \leq \beta \|\boldsymbol{a} - \boldsymbol{b}\|; \beta > 0$.

We denote the gradient of J_t at $\boldsymbol{\theta}_t$ as $\nabla J_t(\boldsymbol{\theta}_t)$ and stochastic gradient as $\hat{\nabla} J_t(\boldsymbol{\theta}_t) = \mathrm{E}[\nabla J_t(\boldsymbol{\theta}_t)]$, or where the context is obvious, ∇_t and $\hat{\nabla}_t$ respectively.

As performance measure, Gu et al. (2020) used pooled R^2_{oos} without mean adjustment in the denominator,

$$R^2_{oos} = 1 - \frac{\sum_{(t,i) \in \mathcal{D}_{oos}} (y_{t,i} - \hat{y}_{t,i})^2}{\sum_{(t,i) \in \mathcal{D}_{oos}} y^2_{t,i}}, \tag{3.1}$$

where $\mathcal{D}_{oos}$ is the pooled out-of-sample dataset covering January 1987 to December 2016 in the empirical study. There are several shortcomings with this performance measure. The number of stocks in the U.S. equities dataset starts from 1,060 in March 1957, peaks at over 9,100 in 1997 and falls to 5,708 at the end of 2016. A pooled performance metric will place more weight on periods with a higher number of stocks. An investor making iterative portfolio

allocation decisions would be concerned with accuracy *on average over time*. Moreover, asset returns are known to exhibit non-Gaussian characteristics (Cont, 2001). Summary statistics of monthly U.S. stock returns are provided in Table 3.2 (in Section 3.5), which confirms the existence of considerable skewness and time-varying variance. Therefore, we provide three additional metrics. The first metric is the IC, defined as the cross-sectional Pearson's correlation[5] between predictions and actual returns:

$$\text{IC} = \frac{1}{T}\sum_{t=1}^{T}\text{IC}_t, \quad \text{IC}_t = \rho(\boldsymbol{y}_t, \hat{\boldsymbol{y}}_t). \tag{3.2}$$

The time-series of IC is averaged to give the final score, which we refer to as mean IC (Equation (3.2)). Note that even though correlation is a score rather than a percentage, IC is typically expressed as a percentage by practitioners. IC was first proposed by Ambachtsheer (1974) and is widely applied in investment management for measuring predictive power of a forecaster or an investment strategy (Grinold and Kahn, 1999; Fabozzi et al., 2011b). The second metric is the annualised Sharpe ratio, calculated as,

$$\text{SR} = \frac{12 \times \text{E}[P_{t\in\mathcal{D}_{oos}}]}{\sqrt{12 \times \text{Var}[P_{t\in\mathcal{D}_{oos}}]}}, \tag{3.3}$$

where $P_{t\in\mathcal{D}_{oos}}$ is the decile return spread. Decile return spread is computed by first sorting stocks into deciles based on their predicted return at t, where decile 10 (decile 1) contains stocks with the highest (lowest) preducted return. The average return for each decile is then computed. Decile return spread is the difference between monthly returns of decile 10 and decile 1 at t,

$$P_t = \frac{10}{N}\left[\sum_{i=1}^{N} y_{t,i}\delta_{t,i}^{(10)} - \sum_{i=1}^{N} y_{t,i}\delta_{t,i}^{(1)}\right], \tag{3.4}$$

where $\delta_{t,i}^{(d)}$ is the decile indicator and $\hat{y}_t^{(d)}$ is the boundary of the d-th decile of all $\hat{y}_{t,i}$ sorted in ascending order,

$$\delta_{t,i}^{(d)} = \begin{cases} 1 & \text{if } \hat{y}_t^{(d-1)} < \hat{y}_{t,i} \leq \hat{y}_t^{(d)}, \\ 0 & \text{otherwise.} \end{cases}$$

Note that in the experiment results in Section 3.5, we use the label P10-1 for the average decile return spread computed over all t, rather than for a single t in Equation (3.4). The

[5] Rank IC, which uses Spearman's rank correlation instead of Pearson's, is also used in practice.

decile return spread and its associated Sharpe ratio are commonly used to measure economic performance of an investment strategy and are used in Gu et al. (2020). The third metric is the average monthly R^2, where denominator is adjusted by the cross-sectional mean, as a conventional complement to R^2_{oos}.

3.2.2 Neural network training under concept drift

A detailed description of neural networks is provided in Chapter 2, including a description of the standard early stopping algorithm in Section 2.2. In the classical early stopping algorithm, a randomly drawn portion of data is used for validation. Training is stopped when validation loss decreases by lower than a predefined amount. Given optimisation steps τ, the product $\eta\tau$ can be interpreted as the effective capacity which bounds reachable parameter space from starting weights, thus behaving like L_2 regularisation (Goodfellow et al., 2016). A discussion of aspects of neural network training that is relevant to online problems is presented in this section.

For time series problems where chronological ordering is important, popular approaches include expanding window (each new time slice is added to the panel dataset) and rolling window (the oldest time slice is removed as a new time slice is added, Rossi and Inoue, 2012). Instead of randomly splitting training and validation sets, the *out-of-sample* procedure[6] can be used where the end of the series is withheld for evaluation. This is unsatisfactory in the context of stock return prediction for two reasons. First, each period is drawn from a different data distribution $\mathcal{D}$ (hereon denoted by $\mathcal{D}_t$ for dataset drawn at time t, or $\mathcal{D}_{oos}$ for all periods in the out-of-sample dataset). A regression that is fitted on a sliding window of size w effectively assumes that data at $t+1$ is drawn from the average DGP of the past $t-w, \ldots, t$ cross-sections. Secondly, if data is scarce in terms of time periods, estimates for optimal optimisation steps $\hat{\tau}_t$ can have large stochastic error. For instance, monthly data with a window size of 12 months and 3:1 training-validation split. $\hat{\tau}$ is estimated using only 3 months of data. To the best of our knowledge, there is no procedure for adapting early stopping in an online context with time-varying dynamics.

[6] As described in Bergmeir et al. (2018).

Note that even though the problem studied in this chapter contains a time-dimension, the problem itself concerns cross-sectional predictions. Thus, conventional sequential neural networks such as recurrent neural networks (Rumelhart et al., 1986b) and long-short term memory networks (Hochreiter and Schmidhuber, 1997) are not well suited to this problem. However, there have been some recent advances in dealing with concept drifts in time-series problems, such as Liu et al. (2019) who proposed to explicitly model concept drift in a discriminative manner inside an extreme learning machine, and Samanta et al. (2020) who proposed to model time-varying temporality of time-series using a Bayesian approach.

3.2.3 Online optimisation

Optimising network weights to track a function evolving under unknown dynamics is an online optimisation problem. A discussion on relevant concepts in online optimisation is provided in this section. Interested readers are encouraged to read Shalev-Shwartz (2012) for a comprehensive review. In online optimisation literature, iterate is often denoted by x_t and loss function by f_t. We have used $\boldsymbol{\theta}_t$ as iterate to be consistent with our parameter of interest and J_t as loss function to avoid conflict with our use of f as activation function.

Optimality of online optimisation and its variants for this class of problems under various assumptions have been well documented in literature (e.g., Shalev-Shwartz, 2012; Cesa-Bianchi et al., 2012; Dworkin et al., 2014). Thus, online optimisation is also well suited to our iterative portfolio selection problem due to their similarities. Applications of online optimisation in finance first came in the form of the *Universal Portfolios* by Cover (1991). However, most of the early works in online optimisation are focused on the convex case and assume each draw of J_t is from the same distribution (in other words, J_t is stationary). These assumptions are not consistent with our problem. Recently, Hazan et al. (2017) extended online convex optimisation to the non-convex and stationary case. This was further extended by Aydore et al. (2019) to the non-convex and non-stationary[7] case, with the proposed DTS-SGD algorithm. Non-convex optimisation is NP-Hard[8]. Therefore, existing non-convex

[7]Non-stationarity in online optimisation literature refers to time-variability of loss function J_t.

[8]In computer science, NP-Hard refers a class of problems where no known polynomial run-time algorithm exists.

optimisation algorithms focus on finding local minima (Hazan et al., 2017). For this reason, one difference between online *convex* optimisation and online *non-convex* optimisation is that the former focuses on minimising sum of losses relative to a *benchmark* (for instance, the minimiser over all time intervals $\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}\in\Theta} \sum_t J_t(\boldsymbol{\theta})$ is one of the most basic benchmarks), and the latter focuses on minimising sum of gradients (e.g., $\sum_t \nabla J_t(\boldsymbol{\theta}_t)$), without comparison to a benchmark. This sum is called *regret* and the optimisation objective is called *regret minimisation*. In online optimisation, it is desirable to design algorithms that minimises *average regret* over time (e.g., $\frac{\sum_t J_t(\boldsymbol{\theta})}{T}$), as this guarantees that as $T \to \infty$, average regret suffered by the algorithm converges to zero (Shalev-Shwartz, 2012). Readers familiar with time-series analysis might be taken aback by the lack of parameters in a typical online optimisation algorithm. This is due to the game theoretic approach of online optimisation and the focus on worst case performance guarantees, as opposed to the average case performance in statistical learning. Regret bounds are typically functions of properties of the loss function (e.g., convexity and smoothness) and are dependent on environmental assumptions.

At each interval t, DTS-SGD updates network weights using a time-weighted sum of past observed gradients. Time weighting is controlled by a forget factor κ. In analysing DTS-SGD, we note two potential weaknesses. Firstly, neural networks are notoriously difficult to train. Geometry of the loss function is plagued by an abundance of local minima and saddle points (see Chapter 8.2 of Goodfellow et al., 2016). Momentum and learning rate decay strategies (for instance, Sutskever et al., 2013; Kingma and Ba, 2015) have been introduced which require multiple passes over training data, adjusting learning rate each time to better traverse the loss surface. DTS-SGD performs a single weight update at each period which may have difficulties in traversing highly non-convex loss surfaces. Secondly, during our simulation tests, we observed that loss can increase after a weight update. One possibility is that a past gradient is taking the weights further away from the current local minima. This is particularly problematic for our problem as stock returns are very noisy.

3.3 The proposed Online Early Stopping algorithm

3.3.1 Tracking a restricted optimum

We start by providing an informal discussion of the algorithm. Neural networks are universal approximators (Cybenko, 1989; Goodfellow et al., 2016). That is, it can approximate any function up to an arbitrary accuracy. Thus, given a network structure and a time-varying function, network weights trained with data from a single time interval (i.e., a cross-sectional slice of time) neatly summarise the function at that interval. The Euclidean distance between consecutive sets of weights can be interpreted as the amount of variations in the underlying function expressed in weight space. Simply using $\boldsymbol{\theta}_{t-1}$ to predict on t will lead to an overfitted result. To illustrate, suppose $\boldsymbol{\theta}_t \in \mathbb{R}$, $\boldsymbol{\theta}_0 = 0$ and $\boldsymbol{\theta}_{\{t>0\}}$ alternates in a sequence of $\{1, -1, 1, -1, ...\}$[9]. Then, it is clear that using $\boldsymbol{\theta}_1$ to predict on $t = 2$ will lead to a worse outcome than using $\boldsymbol{\theta}_0$. In this scenario, the optimal strategy is to never update weights (or scale updates by zero). Generally, the optimal policy is to regularise updates such that the network is not overfitted to any single period.

In the rest of this section, we present our main theoretical results. Formally, our goal is to track the unobserved minimiser of J_t, a proxy for the true asset pricing model, as closely as possible. In regret analysis, it is desirable to have regret that scales sub-linearly to number of periods T, which leads to asymptotic convergence to the optimal solution[10]. Hazan et al. (2017) demonstrated that in the non-convex case, a sequence of adversarially chosen loss functions can force any algorithm to suffer regret that scales with T as $\Omega\left(\frac{T}{w^2}\right)$[11]. Locally smoothed gradients (over a rolling window of w loss functions) were used to improve *smoothed regret*, with a larger w advocated by Hazan et al. (2017). Aydore et al. (2019) extended this to use rolling weighted average of past gradients which give recent gradients a higher weight to track a dynamic function. Inevitably, smoothing will track a time-varying minimiser with a tracking error that is proportionate to w and the forget factor κ.

[9] This is the worst case scenario for OES which we will revisit at the end of this section.
[10] The sum of regret increases sub-linearly to T. Thus, as $T \to \infty$, average regret $\to 0$.
[11] In computer science, Ω notation refers to the lower bound complexity.

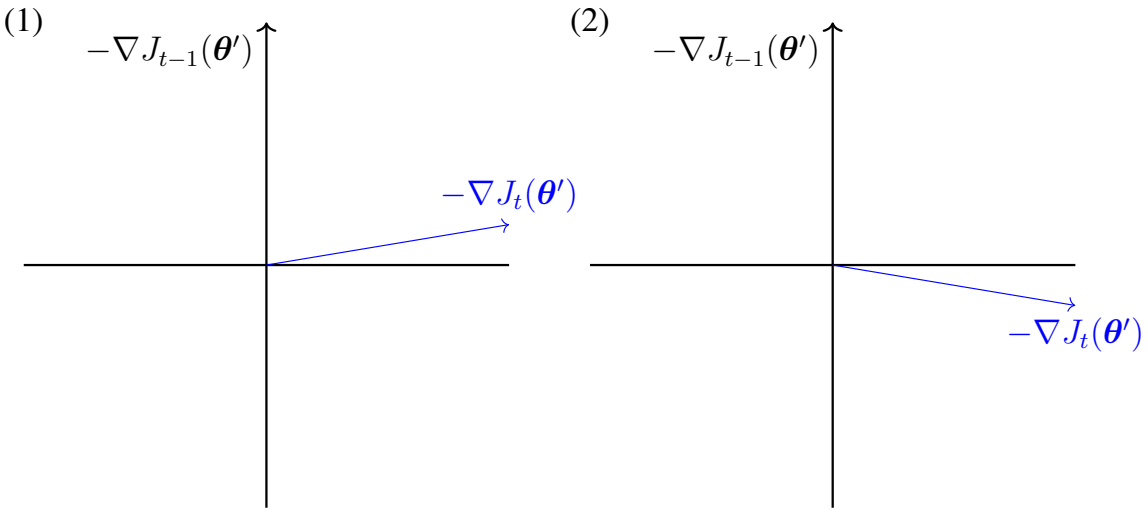


FIGURE 3.1: At each optimisation iteration, weights can be visualised as moving along the direction of $-\nabla J_{t-1}(\boldsymbol{\theta}')$. On the left, optimisation should continue until $-\nabla J_t(\boldsymbol{\theta}')$ is perpendicular to $-\nabla J_{t-1}(\boldsymbol{\theta}')$. On the right, optimisation should terminate.

To address this, we propose a *restricted optimum* (denoted by $\boldsymbol{\theta}_t^*$ at time t) as the tracking target of our algorithm. At time t, the online player selects $\boldsymbol{\theta}_t$ based on observed $\{\nabla_1, \ldots, \nabla_{t-1}\}$. As the network is trained using gradient descent, we propose to restrict the admissible weight set to the path formed from $\boldsymbol{\theta}_{t-1}^*$ and extending along the gradient vector $-\nabla_{t-1}$ (in other words, the path traversed by gradient descent). The point $\boldsymbol{\theta}'$ along this path with the minimum $\|\nabla J_t(\boldsymbol{\theta}')\|$ is the restricted optimum. We argue that the trade-off between restricting the admissible weight space and solving the simplified problem is justified as other points in the weight space are not attainable via gradient descent and is thus unnecessary to consider all possible weight sets in Θ. Without assuming any time-varying dynamics, updating weights using an average of past gradients (similar to Hazan et al., 2017) will induce a tracking error to the time-varying function. To illustrate the restricted optimum concept, let $\boldsymbol{\theta}' = \boldsymbol{\theta}_{t-1}^*$ be our starting point of optimisation, $\boldsymbol{g} = -\nabla J_{t-1}(\boldsymbol{\theta}')$ and $\boldsymbol{g}' = -\nabla J_t(\boldsymbol{\theta}')$. The possible scenarios during training are (also illustrated in Figure 3.1):

(1) If $\left|\cos^{-1} \frac{[\langle \boldsymbol{g}, \boldsymbol{g}' \rangle]}{\|\boldsymbol{g}\| \|\boldsymbol{g}'\|}\right| < \pi/2$, then moving along $\boldsymbol{g}$ will also improve $J_t(\boldsymbol{\theta}')$ until $\boldsymbol{g}$ is perpendicular to $\boldsymbol{g}'$ or $\boldsymbol{\theta}'$ has reached a local minima of J_{t-1}.

(2) If $\left|\cos^{-1}\frac{[\langle \boldsymbol{g},\boldsymbol{g}'\rangle]}{\|\boldsymbol{g}\|\|\boldsymbol{g}'\|}\right| \geq \pi/2$, then following $\boldsymbol{g}$ will not improve $J_t(\boldsymbol{\theta}')$ and training should terminate.

This observation motivates our OES algorithm. In this section, we will use $\boldsymbol{\theta}_t^*$ to denote restricted optimal weights at t and $\boldsymbol{\theta}_t$ to denote the online player's choice of weights. Suppose $\boldsymbol{\theta}_t^*$ evolves under the dynamics of,

$$\boldsymbol{\theta}_t^* = \boldsymbol{\theta}_{t-1}^* - v_{t-1}\nabla J_{t-1}(\boldsymbol{\theta}_{t-1}^*), \tag{3.5}$$

where v_{t-1} is sampled from an unknown distribution. v_{t-1} can be interpreted as a *regulariser* which provides the optimal prediction weights on J_t if we are restricted to travelling along the direction of $-\nabla J_{t-1}(\boldsymbol{\theta}_{t-1}^*)$. In this context, $\|\nabla J_t(\boldsymbol{\theta}_t^*)\|$ is the minimum gradient suffered by the player. Solution to the iterative portfolio selection problem described in Section 3.2.1 contains two loops (one nested within the other). The outer loop recursively updates $\boldsymbol{\theta}_t^*$ for each portfolio selection interval $t = 1, \ldots, T$ (each $\boldsymbol{\theta}_t^*$ in Equation (3.5)). The inner loop relates to the transition between each t, where SGD iteratively updates $\boldsymbol{\theta}_{t-1}^*$ to arrive at $\boldsymbol{\theta}_t^*$ by approximating v_{t-1} in (3.5). In here, let τ_t^* be the optimal number of optimisation steps at time t, τ_t be the estimated number of optimisation steps and k be the k-th SGD optimisation step. At iteration t, we solve optimal optimisation steps τ_{t-2}^*,

$$\tau_{t-2}^* = \underset{\tau' \geq 0}{\operatorname{argmin}}\, J_{t-1}\left[\boldsymbol{\theta}_{t-2}^* - \eta\sum_{k=1}^{\tau'}\nabla J_{t-2}(\boldsymbol{\theta}_{t-2,k}^*)\right]. \tag{3.6}$$

We start from $t-2$ as solving τ_{t-1}^* requires J_t which we are yet to observe. This leads to optimal weights (the restricted optimum) trained on J_{t-2} for prediction on J_{t-1},

$$\boldsymbol{\theta}_{t-1}^* = \boldsymbol{\theta}_{t-2}^* - \eta\sum_{k=1}^{\tau_{t-2}^*}\nabla J_{t-2}(\boldsymbol{\theta}_{t-2,k}^*), \tag{3.7}$$

and can be approximated by,

$$\boldsymbol{\theta}_{t-2}^* - \eta\sum_{k=1}^{\tau_{t-2}^*}\nabla J_{t-2}(\boldsymbol{\theta}_{t-2,k}^*) \approx \boldsymbol{\theta}_{t-2}^* - \eta\tau_{t-2}^*\nabla J_{t-2}(\boldsymbol{\theta}_{t-2}^*),$$

which implies $v_{t-2} \approx \eta\tau_{t-2}^*$. To predict $\hat{\boldsymbol{r}}_t$, we choose $\tau_{t-1} = \frac{1}{t-2}\sum_{q=2}^{t-1}\tau_{t-q}^*$ and train *prediction weights* on J_{t-1} by substituting in $\lfloor \tau_{t-1} + 0.5 \rfloor$ (the rounded up estimate of

optimisation steps),

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}^*_{t-1} - \eta \sum_{k=1}^{\lfloor \tau_{t-1}+0.5 \rfloor} \nabla J_{t-1}(\boldsymbol{\theta}^*_{t-1,k}) \approx \boldsymbol{\theta}^*_{t-1} - \eta\tau_{t-1}\nabla J_{t-1}(\boldsymbol{\theta}^*_{t-1}). \tag{3.8}$$

As η is a constant chosen by hyperparameter search, τ_{t-1} can be interpreted as a proxy to the regulariser v_{t-1}. Using our β-smooth assumption (in Section 3.2.1) and substituting in definitions of $\boldsymbol{\theta}_t$ and $\boldsymbol{\theta}^*_t$ (in Equation 3.8), we obtain total regret,

$$\begin{aligned} \|\nabla J_t(\boldsymbol{\theta}_t) - \nabla J_t(\boldsymbol{\theta}^*_t)\| &\leq \beta \|\boldsymbol{\theta}_t - \boldsymbol{\theta}^*_t\|, \\ \sum_{t=2}^{T} \|\nabla J_t(\boldsymbol{\theta}_t) - \nabla J_t(\boldsymbol{\theta}^*_t)\| &\leq \sum_{t=2}^{T} \beta \|\boldsymbol{\theta}_t - \boldsymbol{\theta}^*_t\|, \\ &\leq \sum_{t=2}^{T} \beta \left\|\eta\tau^*_{t-1}\nabla J_{t-1}(\boldsymbol{\theta}^*_{t-1}) - \eta\tau_{t-1}\nabla J_{t-1}(\boldsymbol{\theta}^*_{t-1})\right\|, \end{aligned} \tag{3.9}$$

where we start from $t = 2$ as our algorithm requires at least 2 cross-sectional observations. The elegance of Equation 3.9 is that it conforms with the conventional notion of regret, with cumulative gradient deficit against an optimal outcome in place of cumulative loss. As τ_{t-1} is the unbiased estimator of τ^*_{t-1}, Equation 3.9 indicates that the cumulative deficit is asymptotically bounded by the variance of τ^*_{t-1}. This concept is illustrated in Figure 3.2. If τ^*_{t-1} is constant, then τ_{t-1} will converge to τ^*_{t-1} and the optimal weights are achieved. Conversely, if τ^*_{t-1} has high variance, then the player will suffer a larger cumulative gradient deficit.

Finally, we discuss the best and worst case scenarios of OES. The best case scenario is if $\boldsymbol{\theta}_t$ is stationary, such that $\tau^*_t = 0$. In this case, from Equation (3.2), regret is 0. The worst case scenario is the example discussed at the beginning of this section. Suppose that $\boldsymbol{\theta}_t \in \mathbb{R}$, $\boldsymbol{\theta}_0 = 0$ and $\boldsymbol{\theta}_{\{t>0\}}$ alternates in a sequence of $\{1, -1, 1, -1, ...\}$. In this case, estimated steps $\tau_t = 0$ and $\boldsymbol{\theta}_t$ never updates. Thus, the upper bound on regret is (from Equation (3.2)),

$$\sum_{t=2}^{T} \|\nabla J_t(\boldsymbol{\theta}_t) - \nabla J_t(\boldsymbol{\theta}^*_t)\| \leq \sum_{t=2}^{T} \beta \left\|\eta\tau^*_{t-1}\nabla J_{t-1}(\boldsymbol{\theta}^*_{t-1}) - 0\right\|,$$

and total regret scales linearly with time, average regret (total regret divided by T) converges a constant and the network always underfit the data. However, as discussed in Section 3.2.3,

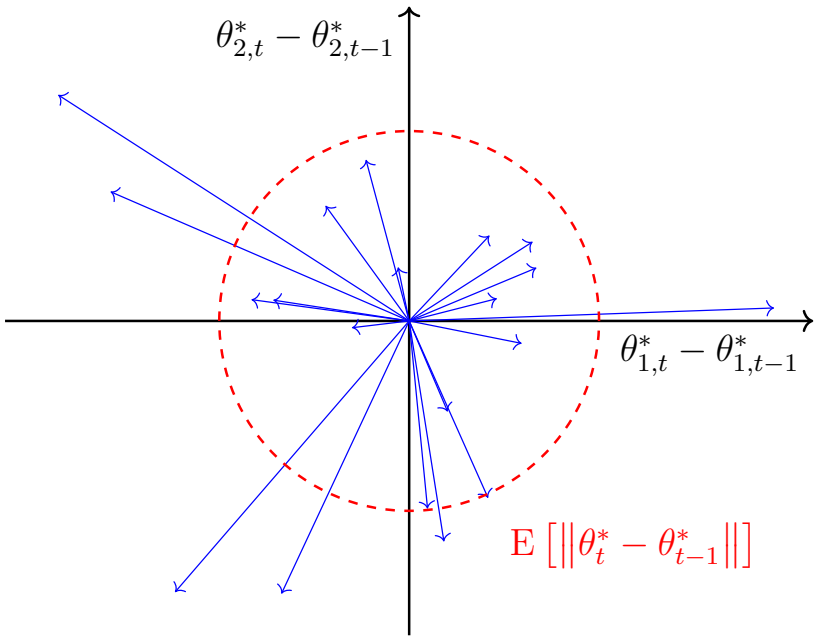


FIGURE 3.2: Illustration of estimating $\mathrm{E}\left[\left\|\boldsymbol{\theta}^*_t - \boldsymbol{\theta}^*_{t-1}\right\|\right]$. Suppose $\boldsymbol{\theta}^*_t = [\theta^*_{1,t} \quad \theta^*_{2,t}]$ is a row vector with two elements. Twenty one random $\boldsymbol{\theta}^*_t$ vectors were drawn with each $\boldsymbol{\theta}^*_t - \boldsymbol{\theta}^*_{t-1}$ pair represented as an arrow. The circle has radius $\frac{1}{20}\sum_{t=2}^{21}\left\|\boldsymbol{\theta}^*_t - \boldsymbol{\theta}^*_{t-1}\right\|$. $\boldsymbol{\theta}_t$ is regularised by limiting how far it can travel from $\boldsymbol{\theta}^*_{t-1}$ which is $\mathrm{E}\left[\left\|\boldsymbol{\theta}^*_t - \boldsymbol{\theta}^*_{t-1}\right\|\right]$.

regret in convex problems is typically compared to a benchmark (e.g., loss suffered by the best hindsight minimiser). In our worst case scenario, the best hindsight minimiser is also $\theta = 0$. Thus, regret suffered by OES converges to the best hindsight minimiser in our worst case. In other words, in the worst case, loss suffered by OES converges to a neural network that is trained on the entire pooled dataset. In Section 3.5, we demonstrate the real world performance of OES on U.S. equities dataset.

3.3.2 Proposed algorithm

Our strategy is to modify the early stopping algorithm to recursively estimate τ_t. An outline is provided below as an introduction to the pseudocode in Algorithm 2:

(1) At t, solve τ^*_{t-2} (Equation 3.6) and $\boldsymbol{\theta}^*_{t-1}$ (Equation 3.7) by training on J_{t-2} and validating against J_{t-1} (step 3 of Algorithm 2).

(2) Recursively estimate τ_{t-1} as the mean of observed $\{\tau^*_1, ..., \tau^*_{t-2}\}$ (line 4).

(3) Start from $\boldsymbol{\theta}^*_{t-1}$ and perform gradient descent for $\lfloor \tau_{t-1}+0.5 \rfloor$ iterations (Equation 3.8). The new weights are $\boldsymbol{\theta}_t$ (line 5–9).

(4) Predict using $\boldsymbol{\theta}_t$ (line 11).

$EarlyStopping$ on line 3 is the classical early stopping procedure as outlined in Algorithm 1 (in Section 2.2). In our implementation of the algorithm, we have used stochastic gradient $\hat{\nabla}_{t-1}$ instead of the full gradient ∇_{t-1}. Validation is performed before the first training step to allow for the case where $\tau_{best} = 0$ (i.e., we start from the optimal weights).

Algorithm 2 General framework for online early stopping. The outer loop recursively estimates τ_{t-1}. See Algorithm 1 for the EarlyStopping function.

Require: data $\boldsymbol{X}_t, \boldsymbol{y}_t \sim p_t$ at interval t; $\boldsymbol{\theta}^*_0$ initialized randomly

1: $\tau' \leftarrow 0$
2: **for** $t = 2, ..., T$ **do**
3: $\quad \tau', \boldsymbol{\theta}^*_{t-1} \leftarrow \text{EARLYSTOPPING}(\boldsymbol{\theta}^*_{t-2}, \boldsymbol{X}_{t-2}, \boldsymbol{y}_{t-2}, \boldsymbol{X}_{t-1}, \boldsymbol{y}_{t-1})$
4: $\quad \tau \leftarrow \frac{\tau(t-2)+\tau'}{t-1}$
5: $\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^*_{t-1}$
6: $\quad$ **for** $i = 1, ..., \lfloor \tau + 0.5 \rfloor$ **do**
7: $\quad\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \hat{\nabla}_{t-1}(\boldsymbol{\theta})$
8: $\quad$ **end for**
9: $\quad \boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}$
10: $\quad$ Receive input $\boldsymbol{X}_t$
11: $\quad$ Predict $\hat{\boldsymbol{r}}_t \leftarrow F(\boldsymbol{X}_t; \boldsymbol{\theta}_t)$
12: $\quad$ Receive output $\boldsymbol{y}_t$
13: **end for**

In the next two sections, we conduct two empirical studies. First is based on simulation data which highlights the use of OES, and the second on predicting U.S. stock returns based on the dataset in Gu et al. (2020) and is presented in Section 3.5.

3.4 Simulation study

3.4.1 Simulation data

For the simulation study, we create the following synthetic dataset:

- $T = 180$ months, each month consists of $N = 200$ stocks.

- Each stock has $M = 100$ features, forming input matrix of $\boldsymbol{X} \in \mathbb{R}^{180\times200\times100}$ and output vector $\boldsymbol{r} \in \mathbb{R}^{180\times200}$.
- Let $x_{t,i,j}$ be the value of feature j of stock i at time t. Each feature value is randomly set to $x_{t,i,j} \sim N(0,1)$.
- Each feature is associated with a latent factor $\psi_{t,j} = 0.95\psi_{t-1,j} + 0.05\delta_{t,j}$, where $\delta_{t,j} \sim \mathrm{N}(0,1)$ and $\psi_{0,j} \sim \mathrm{N}(0,1)$. $\psi_{t,j}$ follows a Wiener process and drifts over time.
- Each output value is $y_{t,i} = \sum_{j=1}^{M} \tanh(x_{t,i,j} \times \psi_{t,j}) + \epsilon_{t,i}$, where $\epsilon_{t,i} \sim \mathrm{N}(0,1)$. Thus, $\boldsymbol{y}_t$ is non-linear with respect to $\boldsymbol{X}_t$ and the relationship changes over time.

We have used the same network setup and hyperparameter ranges as the empirical study on U.S. equities (outlined in Table A.1) but with a batch size of 50. EWNN has the same setup but is re-fitted at every 10-th time intervals. The dataset is split into three 60 interval blocks. Hyperparameters for OES are chosen using a grid search, a procedure called *hyperparameter tuning*. For each hyperparameter combination, the network is trained on the first 60 intervals and validated on the next 60 intervals. Hyperparameters with the minimum MSE in the validation set is used in the remaining 60 intervals as out-of-sample data. Performance metrics are calculated using the out-of-sample set. DGP of the synthetic dataset is designed to be non-linear and time-varying. We expect a slower decay rate to benefit EWNN and a faster decay rate to benefit OES. Size of train, validate and test sets are chosen arbitrarily and is not expected to change the results. DTS-SGD follows the same training scheme as OES, with additional hyperparameters: window period $w \in \{5, 10, 20\}$ and forget factor $\kappa \in \{0.9, 0.8, 0.7\}$. These hyperparameters relate to speed of change of the DGP. A faster changing DGP will lead to smaller window period and forget factor.

3.4.2 Simulation results

Our synthetic data requires the network to adapt to time-varying dynamics. Table 3.1 records results of the simulation. EWNN struggles to learn the time-varying relationships, with mean R^2 of $-8.26\,\%$ and mean rank correlation of $-4.07\,\%$. This is expected as the expanding window approach used in EWNN assumes the relationships at t are best approximated by

TABLE 3.1: Simulation results and selected hyperparameters by hyperparameter search averaged over time and ensemble networks. Values are in percentages unless specified (w refers to number of periods).

%	EWNN	OES	DTS-SGD
Metrics			
Pooled R^2_{oos}	-7.12	50.22	0.13
Mean R^2	-7.77	49.64	-0.33
IC	-4.21	71.24	6.29
Hyperparameters			
Mean L_1 penalty	0.01	0.09	0.04
Mean η	0.55	1.00	0.10
Mean w (periods)			14
Mean κ			83.00

the average relationships in the observed past. OES significantly outperforms the other two methods in this simple simulation, achieving mean R^2 of $49.64\,\%$ and mean rank correlation of $69.63\,\%$. These results demonstrate OES's ability to track a non-linear, time-varying function reasonably closely. There is a preference for higher L_1 regularisation and learning rate. In Aydore et al. (2019), the authors reported issues of exploding gradient with the *static time-smoothed stochastic gradient descent* in Hazan et al. (2017) and that DTS-SGD provided greater stability. In our simulation test, we observe gradient instability with DTS-SGD as well. During training, loss can increase after a weight update. We hypothesise that a past gradient is taking network weights away from the direction of the current local minima and could be an issue with this general class of optimisers. Lastly, we find that mean R^2 tends to be slightly lower than R^2_{oos} (which is reasonable with a smaller denominator of a negative term, see Equation (3.1)).

3.5 Predicting U.S. stock returns

3.5.1 U.S. equities data and model

The U.S. equities dataset in Gu et al. (2020) consists of all stocks listed on NYSE, AMEX, and NASDAQ from March 1957 to December 2016. The average number of stocks exceeds

5,200. Excess returns over risk-free rate are calculated as forward one-month stock returns over Treasury-bill rates. As noted in Section 3.2.1, stock returns exhibit non-Gaussian characteristics. Table 3.2 presents descriptive statistics of excess returns. Monthly excess returns are positively skewed and contain possible outliers that may influence the regression. We follow Gu et al. (2020) in using MSE but note that MSE is not robust against outliers. As noted in Section 3.1, we also provide an alternative setup that excludes microcap stocks. The alternative setup and empirical results are presented in Section 3.5.4.

The feature set includes 94 firm level features, 74 industry dummy variables (based on the first two digits of Standard Industrial Classification code (SIC)), and interaction terms with 8 macroeconomic indicators. The firm features and macroeconomic indicators used in Gu et al. (2020) are based on Green et al. (2017) and Welch and Goyal (2008), respectively. Firm-level features include price-based measures, valuation metrics and accounting ratios. These features are also highlighted in Section 1.5. The purpose of interacting firm-level features with macroeconomic indicators is to capture any time-varying dynamics that are related to (common across all stocks) macroeconomic indicators. For instance, suppose valuation metrics have a stronger relationship with stock returns during periods of high inflation. Then,

TABLE 3.2: Descriptive statistics of monthly excess returns of U.S. equities from April 1957 to December 2016, grouped into 10-Year periods. The numbers in the left column indicate percentiles. Monthly excess returns appear to contain some extreme values, particularly on the positive end. Variance of monthly excess returns varied over time.

%	1957-1966	1967-1976	1977-1986	1987-1996	1997-2006	2007-2016
Mean	0.95	0.25	0.95	0.64	0.90	0.50
Std Dev	9.98	14.89	15.84	18.44	19.93	16.26
Skew	212.44	184.21	365.98	1059.88	502.41	783.70
Min	-76.38	-91.88	-90.14	-99.13	-98.30	-99.90
1	-20.27	-31.41	-33.82	-40.39	-44.61	-38.96
10	-9.26	-14.99	-14.38	-15.61	-17.08	-14.25
25	-4.42	-7.78	-6.54	-6.64	-6.91	-5.76
50	-0.10	-0.65	-0.52	-0.41	0.00	0.24
75	5.14	6.21	6.67	6.18	6.67	5.84
90	11.62	16.23	16.43	16.11	17.57	14.06
99	33.04	49.60	51.99	56.92	65.43	48.08
Max	255.29	432.89	1019.47	2399.66	1266.36	1598.45

this information will be encoded in the interaction term. The aggregated dataset therefore contains $94 \times (8 + 1) + 74 = 920$ features. Each feature has been appropriately lagged to avoid look-forward bias and is cross-sectionally ranked and scaled to $[-1, 1]$. Table A.6 in the Internet Appendix of Gu et al. (2020) contains the full list of firm features.

A subset of the data is available on Dacheng Xiu's website[12] which contains 94 firm-level characteristics and 74 industry classification. Our main result uses $94 + 74 = 168$ firm-level features but results with the full 920 features are also provided as a comparison. At this point, it is useful to remind readers that our goal is to track a time-varying function when the time-varying dynamics are unknown. In other words, we assume that time-varying dynamics between stock returns and features are not well understood or are unobservable. As such, the subset of data without interaction terms is sufficient for our problem. If macroeconomic indicators do encode time-varying dynamics, our network will track changing macroeconomic conditions automatically.

Data is divided into 18 years of training (from 1957 to 1974), 12 years of validation (1975–1986), and 30 years of out-of-sample tests (1987–2016). We use monthly total returns of individual stocks from CRSP. Where stock price is unavailable at the end of month, we use the last available price during the month. Table A.1 (Appendix A3) records test configurations as outlined in Gu et al. (2020) and in our replication. A total of six hyperparameter combinations (L_1 penalty and η in Table A.1) are tested. We use the same training scheme as Gu et al. (2020) to train EWNN. Once hyperparameters are tuned, the same network is used to make predictions in the out-of-sample set for 12 months. Training and validation sets are rolled forward by 12 months at the end of every December and the model is re-fitted. An ensemble of 10 networks is used, where each prediction $\hat{y}_{t,i}$ is the average prediction of 10 networks.

To train OES, we keep the first 18 years (to 1974) as training data, and next 12 years (to 1986) as validation data. For each permutation of hyperparameter set, we have trained an online learner up to 1986. Hyperparameter tuning is only performed once on this period, as opposed to every year in Gu et al. (2020). As the algorithm does not depend on a separate set of data for validation, we simply take the hyperparameter set with the lowest monthly average MSE

[12] Dacheng Xiu's website https://dachxiu.chicagobooth.edu/

over 1975–1986 as the best configuration to use for rest of the dataset. Batch size of 1,000 for OES was chosen arbitrarily.

3.5.2 Predicting U.S. stock returns

In this section, we present our U.S. stock return prediction results. DTS-SGD did not complete training with a reasonable range of hyperparameters due to exploding gradient and is omitted from this section. As an overarching comment, R^2 for both EWNN and OES on U.S. stock returns are very low and are consistent with the findings of Gu et al. (2020). First, results with and without interaction terms are presented in Table 3.3, keeping in mind that our method should be compared against EWNN without interaction terms. Without interaction terms, OES and EWNN achieve IC of $4.53\,\%$ and $3.82\,\%$, respectively. The relatively high correlation of OES (compared to EWNN) indicates that it is better at differentiating relative performance between stocks. This is particularly important in our use case as practitioners build portfolios based on expected relative performance of stocks. For instance, a long-short investor will buy top-ranked stocks and short sell bottom-ranked stocks and earn the difference in relative return between the two baskets of stocks. Mean R^2 are $-12.14\,\%$ and $-9.68\,\%$ for OES and EWNN, respectively. Note that the denominator of mean R^2 is adjusted by the cross-sectional mean of excess returns. Therefore, negative means R^2 of both OES and EWNN indicate that neither method can accurately predict the magnitude of cross-sectional returns. Finally, OES scores $-2.48\,\%$ on R^2_{oos} and EWNN scores $0.22\,\%$. The low values of both methods underscore the difficulty in return forecasting. EWNN achieves higher Sharpe ratio (Equation (3.3)) than OES, at 1.63 and 0.83, respectively. As we will point out in Section 3.5.4, the high Sharpe ratio of EWNN is driven by microcap stocks. Despite the very low R^2, both methods can generate economically meaningful returns. This underscores our argument that R^2 is not the best measure of performance and verifies practitioners' choice of correlation as the preferred measure. We observe similar performance with interaction terms, suggesting that the 8 macroeconomic time series have little interaction effect with the 94 features. In the subsequent results in this section, we only report statistics without interaction terms.

TABLE 3.3: Predictive performance on U.S. equities. Pooled R^2_{oos} is calculated across the entire out-of-sample period as a whole. Mean R^2 and IC are calculated cross-sectionally for each month then averaged across time. P10-1 is the average monthly spread between top and bottom deciles. Sharpe ratio is based on P10-1 return spread and annualised. Mean hyperparameters are calculated over the ensemble of 10 networks and across all periods. *As reported* are results in Gu et al. (2020).

	With Interactions			W/O Interactions	
%	As reported	EWNN	OES	EWNN	OES
Metrics					
Pooled R^2_{oos}	0.4	0.13	-1.93	0.22	-2.48
Mean R^2		-9.89	-11.93	-9.68	-12.17
IC		3.51	4.22	3.82	4.53
P10-1	3.27	1.83	2.10	2.39	2.41
Sharpe ratio	2.36	0.94	0.72	1.63	0.83
Hyperparameters					
Mean L_1 penalty		0.0012	0.0154	0.0024	0.0028
Mean η		0.77	0.10	0.67	0.10

So why do IC and R^2_{oos} diverge? The answer lies in Table 3.4 and Figure 3.3. Here, we form decile portfolios based on predicted returns over the next month and track their respective realised returns. OES predicted values span a wider range than EWNN. This has contributed to a lower R^2, even though OES can better differentiate relative performance between stocks. EWNN used a pooled dataset which will average out time-varying effects. As a result, the average gradient will likely be smaller in magnitude. This is evident from the lower mean L_1 penalty and higher learning rate η chosen by validation. By contrast, OES trains on each time period individually and the norm of the gradient presented to the network at each period is likely to be larger. This led to a lower learning rate chosen by validation. Hence, variance of OES predicted values is higher and potentially requires higher or different forms of regularisation.

In Table 3.4 and Figure 3.3, we observe that the prediction performance of EWNN is concentrated on the extremities, namely P1 and P10, with realised mean returns of $-0.47\,\%$ and $1.92\,\%$ respectively. Stocks between P3 and P7 are not well differentiated. By contrast, OES is better at ranking stocks across the entire spectrum. Realised mean returns of OES are more

evenly spread across the deciles, resulting in higher correlation than EWNN. P10-1 realised portfolio returns are similar across EWNN and OES at 2.39 % and 2.41 %, respectively. However, the difference in mean return spread increases when calculated on a quintile basis (mean return of top 20 % of stocks minus bottom 20 %), to 1.75 % and 1.90 % for EWNN and OES, respectively. This reflects better predictiveness in the middle of the spectrum of OES. An investor holding a well diversified portfolio is more likely to utilise predictions closer to the center of the distribution and experience relative returns that are reminiscent of the quintile spreads (and even tertile spreads) rather than decile spreads. Lastly, forecast dispersion of OES is relatively high compared to EWNN and realised decile returns. We hypothesise that this is due to the small training dataset used by OES on each iteration (consisting of only the cross-section) and suggests additional regularisation may be required.

3.5.3 Time-varying feature importance

So far, our forecasts are predicated on time-varying relationships between features and stock returns. How do features' importance change over time? To examine this, we train the OES

TABLE 3.4: Predicted and realised mean returns by decile where each row represents a decile. *P1* is the mean excess returns of the first decile (0-10% of bottom ranked stocks) and P10-1 is *P10* less *P1* showing the return spread between the best decile relative to the worst decile. *As reported* are original results from Table A.9 in Gu et al. (2020).

	As reported		EWNN		OES	
%	Predicted	realised	Predicted	realised	Predicted	realised
P1	-0.31	-0.92	-0.59	-0.47	-3.53	-0.50
P2	0.22	0.16	0.09	0.15	-1.96	0.03
P3	0.45	0.44	0.37	0.54	-1.07	0.27
P4	0.60	0.66	0.55	0.64	-0.34	0.48
P5	0.73	0.77	0.70	0.73	0.30	0.67
P6	0.85	0.81	0.84	0.78	0.88	0.85
P7	0.97	0.86	0.99	0.85	1.46	1.04
P8	1.12	0.93	1.17	0.96	2.10	1.18
P9	1.38	1.18	1.43	1.26	2.89	1.42
P10	2.28	2.35	2.33	1.92	4.25	1.91
P10-1	2.58	3.27	2.92	2.39	7.78	2.41

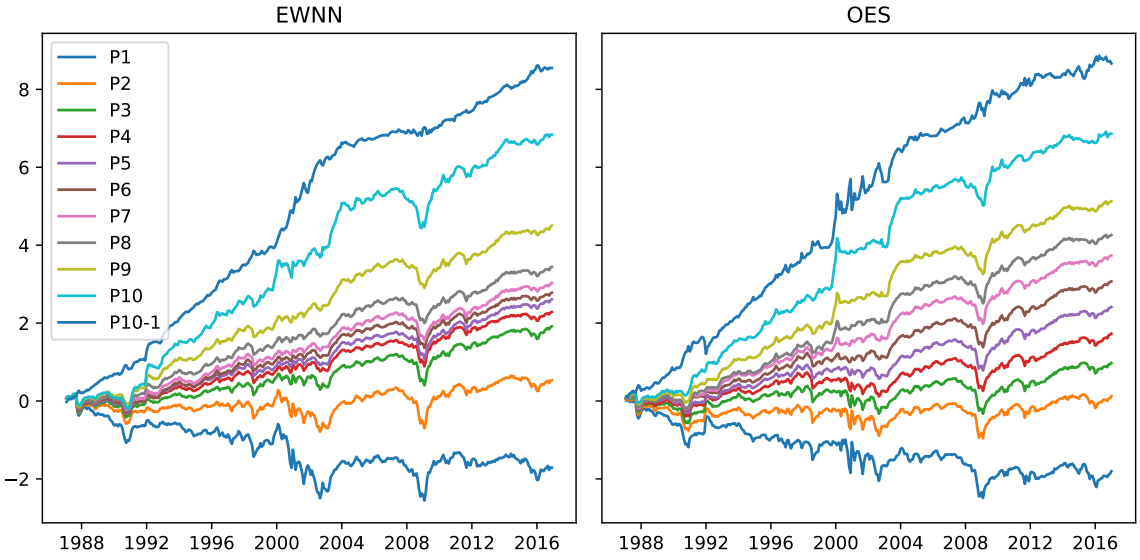


FIGURE 3.3: Cumulative mean excess returns by decile sorted based on predictions by EWNN and OES. Each portfolio follows the same construction as described in Table 3.4. However, cumulative mean excess returns of each portfolio is presented in the chart.

model at every period and make a baseline prediction. For each feature $j = 1, ..., M$, all values of j are set to zero and a new prediction is made. A new R^2 is calculated between the new prediction and the baseline prediction, denoted as $R^2_{t,j}$. The importance of feature j at time t is calculated as $\text{FI}_{t,j} = 1 - R^2_{t,j}$. Our measure tracks features that the network is using. This is different from the procedure in Gu et al. (2020) where R^2 is calculated against actual stock returns, rather than a baseline prediction.

To illustrate the inadequacy of a non-time-varying model, we first track feature importance over January 1987 to December 1991. The top 10 features with the highest feature importance are (in order of decreasing importance): *idiovol* (CAPM residual volatility), *mvel1* (log market capitalisation), *dolvol* (monthly traded value), *retvol* (return volatility), *beta* (CAPM beta), *mom12m* (12-month minus 1-month price momentum), *betasq* (CAPM beta squared), *mom6m* (6-month minus 1-month month price momentum), *ill* (illiquidity), and *maxret* (30-day max daily return). Rolling 12-month averages were calculated to provide a more discernible trend, with the top 5 shown in Figure 3.4. Feature importance exhibits strong time-variability.

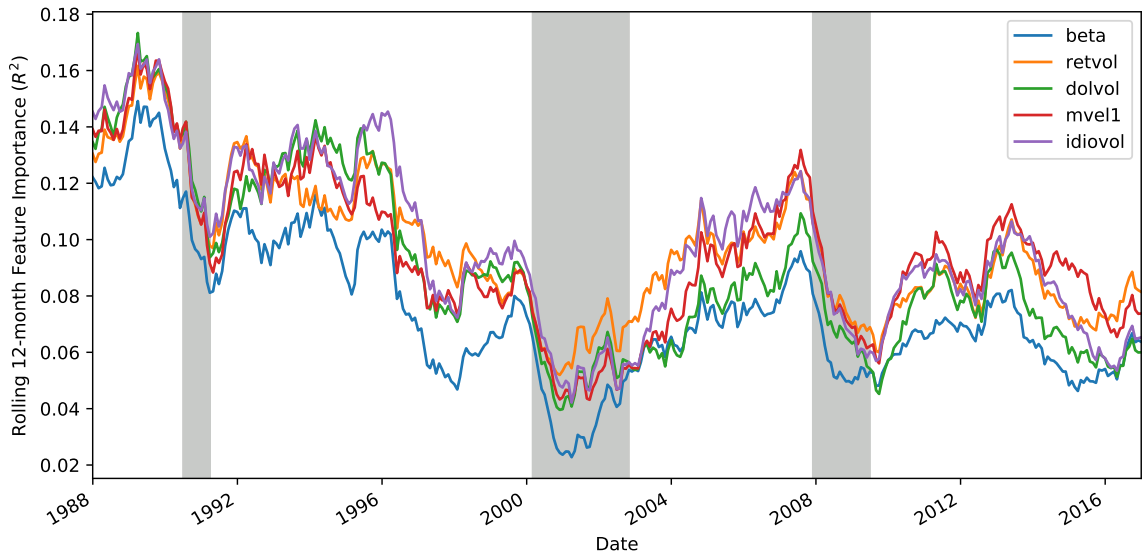


FIGURE 3.4: Top 5 features based on rolling 12-month average feature importance over 1987-1991. Three rapid falls can be seen which coincide with the 1990–91 U.S. recession, Dot-com bubble (2000–03) and the Global Financial Crisis (2007–09). These periods are shaded for reference.

Rolling 12-month average feature importance fell from $14\,\%$ to $16\,\%$ at the start of the out-of-sample period to a trough of $2\,\%$ to $6\,\%$ before rebounding. This indicates that the network would have changed considerably over time. Rapid falls in feature importance can be seen in Figure 3.4, over 1990–91, 2000–01 and 2008–09. These periods correspond to the U.S. recession in early 1990s, the Dot-com bubble and the Global Financial Crisis, respectively. Thus, market distress may explain rapid changes in feature importance.

Next, we examine changes in importance for all features on a yearly basis. Figure 3.5 displays considerable year-to-year variations in feature importance. As there are just a few clusters of features with relatively higher feature importance, the network's predictions can be attributed to a small set of features. This is likely due to the use of L_1 regularisation which encourages sparsity. There is an overall trend towards lower importance over time, consistent with the publication-informed trading hypothesis of McLean and Pontiff (2016). For instance, the importance of market capitalisation (*mvel1*) has decreased over time, as documented in Horowitz et al. (2000). There are periods of visibly lower importance for all features, over 2000–02 and 2008–09, and to a lesser extent 1990 and 1997 (Asian financial

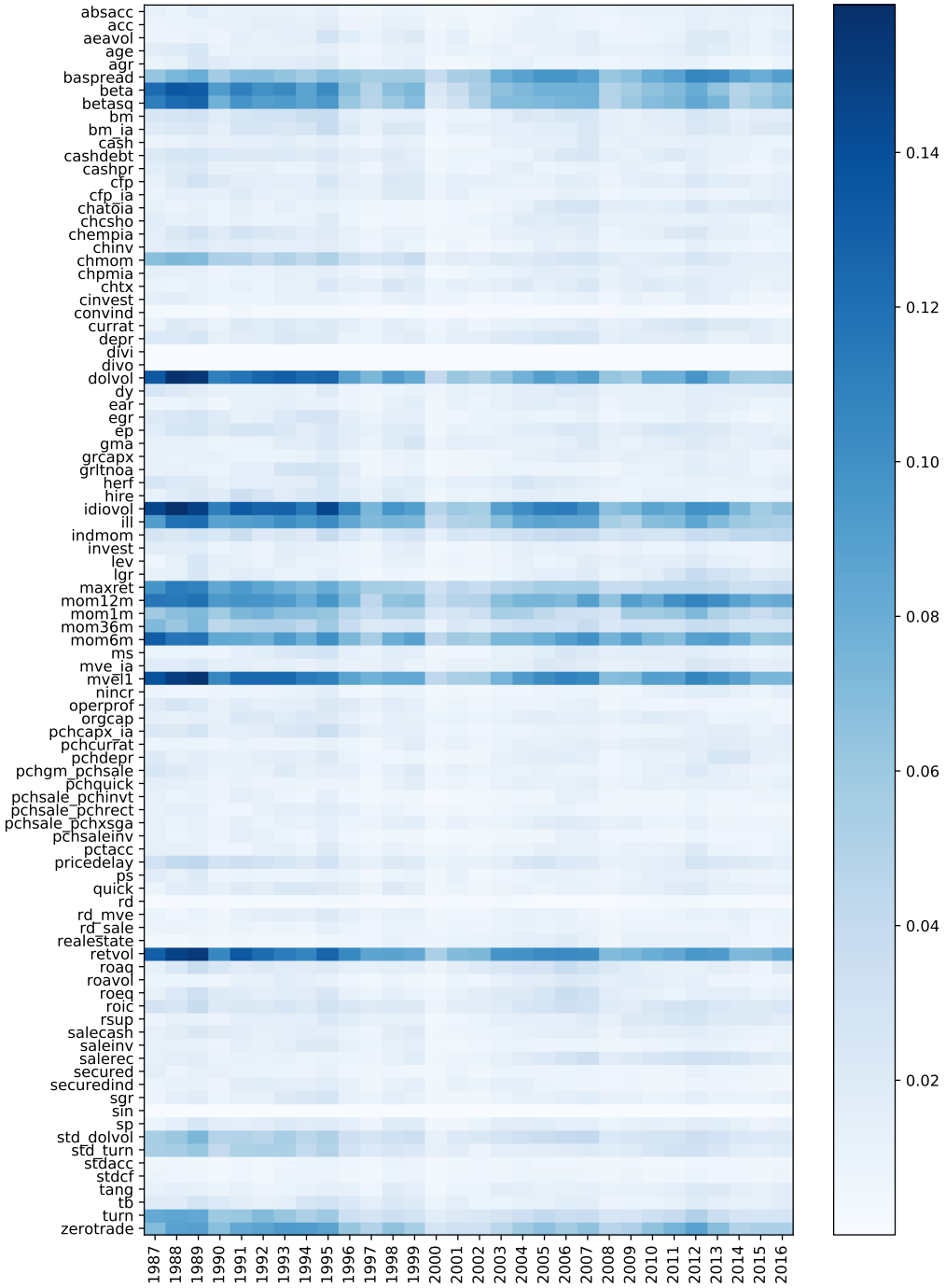


FIGURE 3.5: Yearly average R^2 to baseline predictions (in decimal). The OES network appeared to use only a handful of features. Shades of feature importance are distinctly lighter over 2000–02, 2008–09, and to a lesser extent in 1990 and 1997. Importance of some features have eroded over time (e.g., *dolvol*, *maxret* and *turn*).

crisis). If all features have lower importance during market distress, then what explains stock returns during these periods? To answer this question, we turn to importance of sectors, using SIC 13 (Oil and Gas), 60 (Depository Institutions) and 73 (Business Services) as proxies for oil companies, banks and technology companies, respectively. Figure 3.6 records the rolling 12-month average R^2 to baseline prediction of banks, oil and technology companies. The peak of importance of SIC 73 overlaps with the Dot-com bubble and peak of SIC 60 occurs just after the Global Financial Crisis (which started as a sub-prime mortgage crisis). Importance of SIC 13 peaked in 2016, coinciding with the 2014–16 oil glut which saw oil prices fell from over US$100 per barrel to below US$30 per barrel. This is an example of how an exogenous event that is confined to a specific industry impacts on predictability of stock returns. Thus, a plausible explanation for the observed results is that firm features explain less of cross-sectional returns during market shocks, which becomes increasingly explained by industry groups. This is particularly true if the market shock is industry related. For instance, technology companies during the Dot-com bubble, oil companies during an oil crisis and lodging companies during a pandemic. This underscores the importance to have a dynamic model that adapts to changes in the true model.

In this chapter, we argue that τ_t^* can be interpreted as a measure of variations between consecutive months. Recall that to solve τ_t^*, we train on $\mathcal{D}_t$ and validate on $\mathcal{D}_{t+1}$. Thus, τ_t^* is low if training on $\mathcal{D}_t$ is not beneficial for prediction on $\mathcal{D}_{t+1}$ and τ_t^* is high if $\mathcal{D}_t$ and $\mathcal{D}_{t+1}$ are relatively similar *and* there is a lot of room to update the network before early stopping terminates training. An example of such scenario is when the market is at a turning point, transiting from a risk-averse (risk-seeking) environment to a risk-seeking (risk-averse) environment. Optimal number of iterations τ_t^* is specific to month t and using it to train a network to predict for the next month will lead to overfitting. However, it is still useful to analyse τ_t^* as it provides information on the time-variability of the market as a whole. Figure 3.7 records both τ_t^* and τ_t of OES, including the hyperparameter tuning period (from 1957 to 1986). τ_t converges quickly to approximately 4 iterations and stays relatively stable throughout the approximately 60-year history. Most of the time, τ_t^* fluctuates between 1 and 7, and occasionally jumps to over 10. Periods of U.S. recession and the 2014–16 oil glut have been shaded in grey. Additionally, we have also shaded two market events in green, the

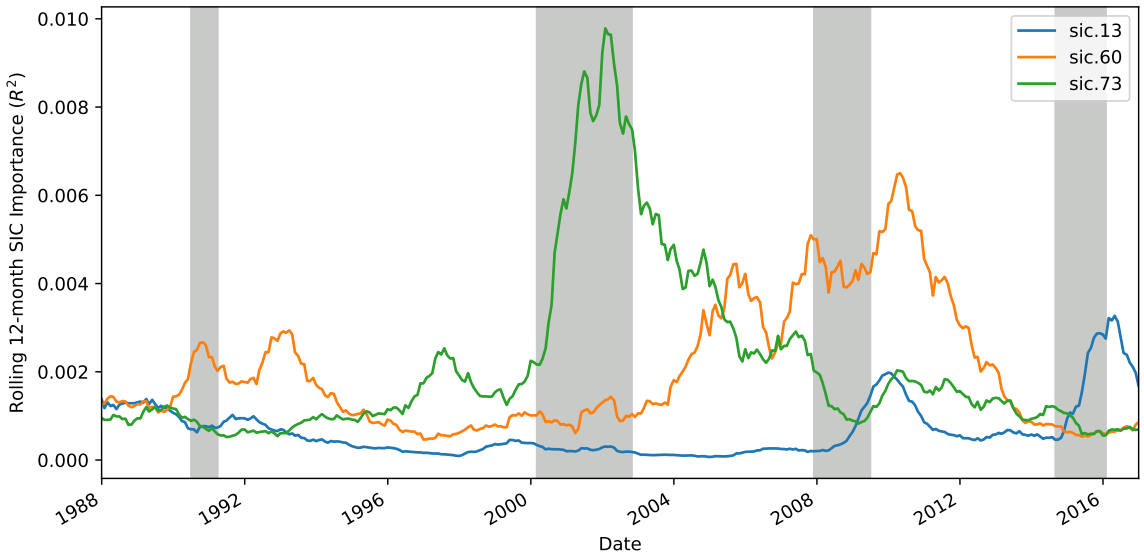


FIGURE 3.6: Rolling 12-month average R^2 to baseline prediction of SIC code 13, 60 and 73, as proxies for oil & gas companies, banks and technology companies, respectively. R^2 of technology companies peaks over 2001–02, banks over 2008–10, and oil companies over 2015–16. Duration of 1990–91 U.S. recession, Dot-com bubble, Global Financial Crisis and the 2014–16 oil glut have been shaded in grey.

Black Monday stock market crash in October 1987, and the collapse of Long-Term Capital Management in August 1998. These events have caused τ_t^* to spike, indicating a sudden change in the underlying DGP. The month with the highest τ_t^* is March 2009, which coincides with the start of a broad market rebound during the depth of the Global Financial Crisis. During these periods ($\tau_t < \tau_t^*$), OES stops training early and prevents overfitting to the large change in DGP.

3.5.4 Investable simulation

As noted in Section 3.1, the dataset in Gu et al. (2020) contains many stocks that are small and illiquid. The U.S. Securities and Exchange Commission (2013) defines "microcap" stocks as companies with market capitalisation below US$250–300 million and "nanocap" stocks as companies with market capitalisation below US$50 million. At the end of 2016, there are over 1,300 stocks with market capitalisation below US$50 million and over 1,800 stocks with

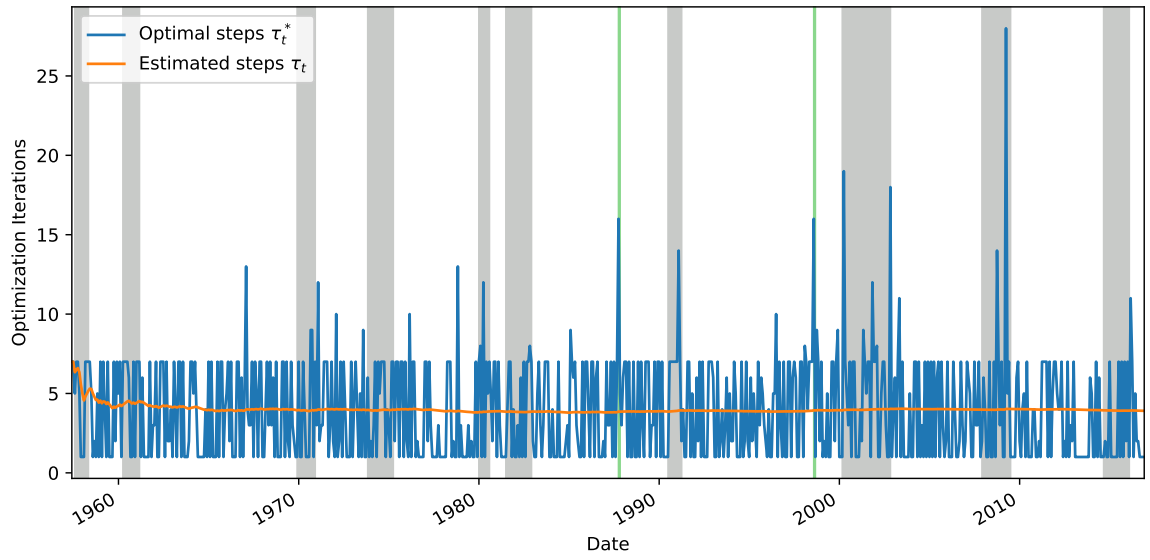


FIGURE 3.7: Optimal and estimated number of optimisation iterations. U.S. recessions and the oil glut (2014–16) have been shaded in grey. Two market shocks — the Black Monday stock market crash in October 1987 and the collapse of Long-Term Capital Management in August 1998, have been shaded in green.

market capitalisation between US$50 million and US$300 million. Together, microcap and nanocap stocks constitute close to half of the dataset as of 2016. Thus, we also provide results excluding these stocks. At the end of every June, we calculate breakpoint based on the 5-th percentile of NYSE listed stocks and exclude stocks with market capitalisation below this value. Once rebalanced, the same set of stocks are carried forward until the next rebalance (unless the stock ceases to exist). This cutoff is chosen to approximately include the larger half of U.S.-listed stocks, with the average number of stocks exceeding 2,600. We label this dataset as the *investable set*. To mitigate the impact of outliers, we also winsorise excess returns at $1\,\%$ and $99\,\%$ for each month (separately). Winsorised returns are then standardised by subtracting the cross-sectional mean and dividing by cross-sectional standard deviation. Standardisation is a common procedure in machine learning and can assist in network training (LeCun et al., 2012). Predicting a dependent variable with zero mean also removes the need to predict market returns which are embedded in stocks' excess returns (over risk-free rate).

This transformation allows the neural network to more easily learn the relationships between relative returns and firm characteristics.

Results based on this investable set are presented in Table 3.5. Both R^2_{oos} and IC improved once microcaps are excluded, with OES scoring $6.05\,\%$ on IC and EWNN on $5.74\,\%$. However, EWNN experienced a significant drop in mean decile spread (to $1.69\,\%$ per month) and Sharpe ratio (0.69), suggesting that microcaps are significant contributors to the results using the full dataset. By contrast, mean decile spread and Sharpe ratio remain stable for OES, at $2.41\,\%$ and 0.82, respectively. This indicates that the predictive performance of OES was not driven by microcap stocks. We believe this is a meaningful result for practitioners as this subset represents a relatively accessible segment of the market for institutional investors. An ensemble based on the average of cross-sectionally standardised predictions of the two models achieved the best IC, decile spread and Sharpe ratio relative to OES and EWNN. Mean monthly correlation between OES and EWNN is only $35.9\,\%$. Thus, an ensemble based on the two methods can effectively reduce variance of the predictions. Monthly correlations between the two models are presented in Figure 3.8. We observe that correlation tends to be

TABLE 3.5: Predictive performance on the investable set. Ensemble is the average of standardised predictions of the two methods. Pooled R^2_{oos} is calculated across the entire out-of-sample period as a whole. Mean R^2 and IC are calculated cross-sectionally for each month then averaged across time. P_t is the average monthly spread between top and bottom deciles. Sharpe ratio is based on P_t and annualised by multiplying $\sqrt{12}$. Mean hyperparameters are calculated over the ensemble of 10 networks and across all periods.

%	EWNN	OES	Ensemble
Metrics			
Pooled R^2_{oos}	0.35	-1.37	
Mean R^2	0.35	-1.37	
IC	5.74	6.05	6.29
P10-1	1.69	2.41	2.60
Sharpe ratio	0.69	0.82	0.96
Hyperparameters			
Mean L_1 penalty	0.0211	0.0046	
Mean η	0.87	0.10	

lowest immediately after a recession or crisis. We hypothesise that OES is quicker to react to economic recovery.

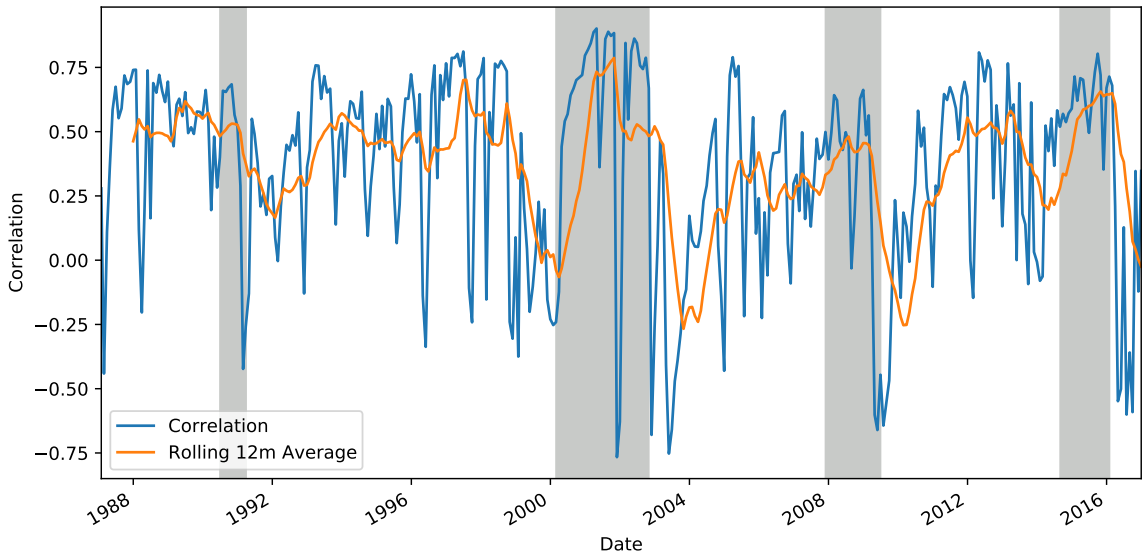


FIGURE 3.8: Monthly and rolling 12-month correlation between predictions of OES and EWNN. Duration of 1990–91 U.S. recession, Dot-com bubble, Global Financial Crisis and the 2014–16 oil glut have been shaded in grey.

Turning to cumulative decile returns presented in Figure 3.9, we observe significant drawdowns for EWNN during recovery phases of the Dot-com bubble and Global Financial Crisis. P1 of EWNN bounced back sharply during these episodes, causing sharp drops in decile spreads and are consistent with *momentum crashes* (Daniel and Moskowitz, 2016). By contrast, decile spreads of OES appear to react to the recovery more quickly. Consistent with prior findings, the spreads between decile 3 to 7 are also better under OES than EWNN in the investable set. Given these favourable characteristics, practitioners are likely to find OES a useful tool to add to the armoury of prediction models.

3.6 Conclusions

Stock return prediction is an arduous task. The true model is noisy, complex and time-varying. Mainstream deep learning research has focused on problems that do not vary over time and,

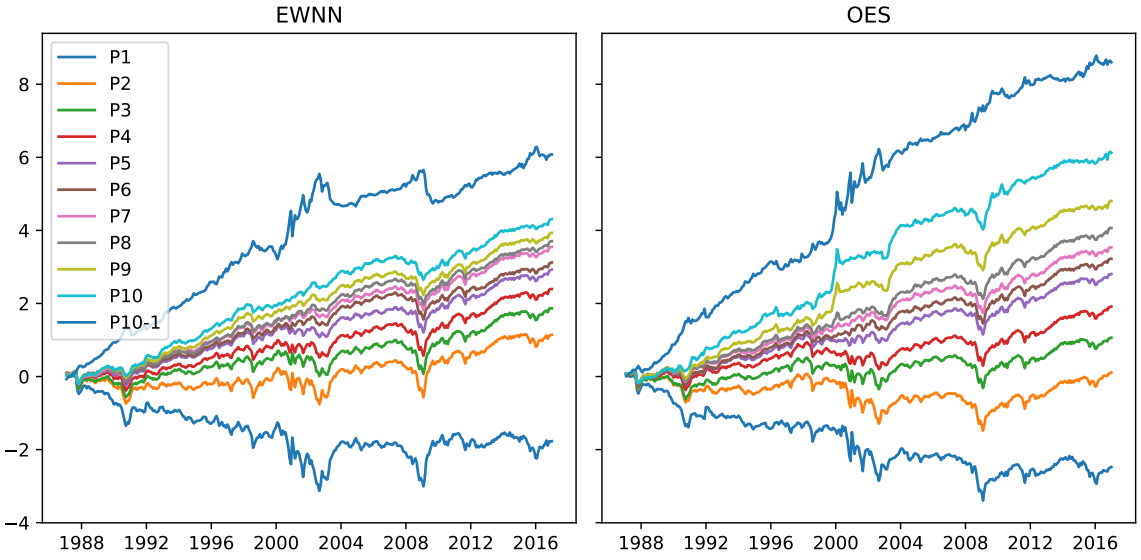


FIGURE 3.9: Cumulative mean excess returns by decile sorted based on predictions by EWNN and OES in the investable set.

arguably, time-varying applications have seen less advancements. In this chapter, we propose an Online Early Stopping algorithm that is easy to implement and can be applied to an existing network setup. We show that a network trained with OES can track a time-varying function and achieve superior performance to DTS-SGD, a recently proposed online non-convex optimisation technique. Our method is also significantly faster, as only two periods of training data are required at each iteration, compared to the pooled method used in Gu et al. (2020) which re-trains the network on the entire dataset annually. In our tests, the pooled method took $5.5\,\text{hours}$ to iterate through the entire dataset (an ensemble of ten networks therefore takes $55\,\text{hours}$)[13]. By contrast, our method took $44.25\,\text{mins}$ for a single pass over the entire dataset (an ensemble of ten networks took $7.4\,\text{hours}$).

Gu et al. (2020) suggested that a small dataset and low signal-to-noise ratio were reasons for the lack of improvement with a deeper network. To this end, we show that only a handful of features contribute to predictive performance. This may be due to correlation between features and the use of L_1 regularisation which encourages sparsity. We also find evidence of

[13]Tests performed on AMD Ryzen™ 7 3700X, Python 3.7.3, Tensorflow 1.12.0 and Keras 2.2.4. Hyperparameter grid search was performed concurrently.

time-varying feature importance. In particular, features such as log market capitalisation (the size effect) and 12-month minus 1-month momentum have seen a gradual decrease to their importance towards the end of our test period, consistent with the publication-informed trading hypothesis of McLean and Pontiff (2016). We find that sectors can also exhibit time-varying importance (for instance, technology stocks during the Dot-com bubble). These results have strong implications for practitioners forecasting stock returns using well known asset pricing anomalies. Excluding microcaps, we find that OES offers superior predictive performance in a subuniverse that is accessible to institutional investors. We find that correlation between OES and EWNN is at its lowest after a recession or crisis. We argue that this is driven by faster reactions of OES in tracking the recovery. An ensemble based on the average prediction of the two models achieves the best IC and Sharpe ratio, suggesting that the two methods may be complementary.

From an academic perspective, recent advances in deep learning such as dropout and *residual connections* (He et al., 2016) may allow deeper networks to be trained, enabling more expressive asset pricing models. Given the higher variance of predictions produced by OES, future work should explore alternative methods of regularisation including dropouts, L_2 penalty or a mixture of regularisation techniques.

In Section 3.3.1, we have discussed the worst case regret of OES which, under adversarial assumptions, can lead to regret that scales linearly with time. We note that our worst case regret converges to the best hindsight minimiser (i.e., the best choice of $\boldsymbol{\theta}$ if the investor is only allowed to pick one $\boldsymbol{\theta}$ for all time periods in hindsight). Thus, this provides users with a guarantee on worst case performance that is no worse than the best hindsight minimiser. The interpretation of this performance guarantee is as follows. In the worst case, our proposed OES algorithm will converge to the performance of a single neural network that is fitted on the entire dataset, as if the problem is stationary. However, this opens up an avenue for future research in advancing online non-convex optimisation algorithms that achieve regret that scales sublinearly with time, such that average regret converges to zero as $T \to \infty$.

In this chapter, we have applied neural networks in a cross-sectional prediction context — inputs into the network are point-in-time attributes of a stock and the problem is treated as a

conventional panel regression problem (with a neural network in place of a linear regression model). The network itself does not learn time-series features of the raw time-series. In Chapter 4, we explore neural networks that can learn from time-series directly.

CHAPTER 4

Supervised temporal autoencoder for stock return time-series forecasting

---

Financial markets are noisy learning environments. We propose an approach that regularises the TCN using a supervised autoencoder, which we term the STAE. We show that the addition of the auxiliary reconstruction task is beneficial to the primary supervised learning task in the context of stock return time-series forecasting. The supervised autoencoder denoises the input and encourages the main network to retain features that are beneficial to both prediction and reconstruction tasks. We show that the supervised temporal autoencoder is able to learn features directly from noisy stock price series, alleviating the need for handcrafted features. These contributions have resulted in the following publication:

Steven Y. K. Wong, Jennifer S. K. Chan, Lamiae Azizi, Richard Y. D. Xu, "Supervised Temporal Autoencoder for Stock Return Time-series Forecasting," *Proceedings of the IEEE 45th Annual Computer Software and Applications Conference*, Madrid, Spain, 2021.

4.1 Introduction

The motivating application of this chapter is in time-series stock return predictions — a problem that can be cast as a pattern recognition task. Using financial time-series forecasting as the motivating application, we focus solely on advancing the existing state-of-the-art deep learning techniques to deal with noise in a regression setting, as discussed in Section 1.6.2. In particular, the noisiness of financial markets exacerbates the difficulty in recognising patterns in stock prices/returns.

Deep learning has become the state-of-the-art in many time-series applications, such as machine translation (Sutskever et al., 2014) and audio generation (van den Oord et al., 2016). Whilst deep learning has achieved tremendous success in sequential applications such as speech and languages, advances in their applications in economics and finance have been relatively modest. Historically, sophisticated machine learning methods have not fared well in forecasting competitions such as the M-competitions[1] (Makridakis and Hibon, 2000; Makridakis et al., 2018; Hyndman, 2020). Empirical results suggest that simpler statistical methods are at least as accurate as sophisticated methods, and models that best fitted training data do not necessarily lead to higher forecasting performance out-of-sample (Makridakis et al., 2018). We argue that time-series in linguistics and speech are *information rich* — where preceding words or sound waves have high information content for the learning task and are naturally well-suited to neural networks with their rich parameterisation. In contrast, time-series in finance and economics are *information deprived* — where signal-to-noise tends to be low (Gu et al., 2020). Thus, good generalisation performance in high-noise environments requires regularisation and certain "scaffolds" to guide the model through the noisy data.

Classical statistical models such as autoregressive-moving-average (ARMA) (Box et al., 1994) models have strong scaffolds. The user picks the orders of autoregressive and moving average terms (through hyperparameter tuning). The two components additively contribute to the forecast in a linear manner. The scaffold "guides" the model to look for autoregressive relationships in the input sequence and residuals, and can be interpreted as a constraint on the functional form of the sequential DGP. However, statistical models such as ARMA are less able to capture more complex or non-linear patterns embedded in the time-series. Recently, Oreshkin et al. (2020) proposed Neural Basis Expansion Analysis for interpretable Time Series (N-BEATS), a hybrid model where basis functions are parameterised by fully-connected blocks. The authors reported state-of-the-art performance on the M4 competition dataset. Users are able to apply domain knowledge when choosing which basis function to use. For instance, if the user knows, *a priori*, that the data exhibits seasonality, then a cyclical basis function can be used. Basis functions provide a "strong form" of scaffold for

[1] M-competitions are forecasting competitions with datasets consisting of mainly business, economics, finance and demographics time-series.

the model. As the overall structure of the model is imposed by the user, the model only needs to fill in the intricacies during learning. This hybrid model approach aids interpretability and generalisation (by preventing an unconstrained neural network from overfitting on noise), but ultimately sacrifices the prized expressiveness of neural networks and requires assumptions on the data generation process. This restrictiveness may be undesirable to some users who want to take a more data-driven approach. At the other end of the spectrum, sequential neural network architectures have taken full advantage of the expressiveness on offer for information-rich applications. LSTM has been a popular choice of neural network architecture for sequential applications (as discussed in Section 2.5.1). More recently, *transformer* models (Vaswani et al., 2017) have achieved breakthrough advances in natural language processing (e.g., Devlin et al., 2019; Brown et al., 2020). The use of "direction-agnostic" attention mechanism allows transformers to perform paired associations in any part of the sequence, and thus solving a limitation of LSTM for language applications (Vaswani et al., 2017; Zeng et al., 2022). However, it remains to be seen if this flexibility afforded by transformers can benefit highly noisy financial time-series. In this chapter, we propose the Supervised Temporal Autoencoder (STAE) as an extension of the TCN (Bai et al., 2018). STAE augments TCN with an auxiliary learning task which acts as a regulariser. We show that STAE improved generalisation of TCN in time-series prediction of monthly returns of a broad set of U.S. stocks. In our proposed dual-objective network, interpretability is achieved through examining the replicated sequence produced by the autoencoder. We show in Section 4.3.2 that the autoencoder has a denoising effect on stock prices. We note that there have been previous attempts in combining an autoencoder with other network architectures (e.g., with LSTM in Heaton et al., 2016 and with CNN in Korczak and Hemes, 2017). However, in these works, the autoencoder is used for dimensionality reduction rather than as a regulariser as proposed in this chapter.

Methods of forecasting stock returns can be classified into two types, by cross-sectional prediction using firm-level features (as described in Section 1.6.1 and Chapter 3); and time-series forecasting[2] (as described in Section 1.6.2 and this chapter). Time-series forecasting implicitly assumes a stock's history contains information about its own future. This assumption stands

[2]Typically, features include a stock's own closing price history and related time-series data such as open, high, low and traded volume.

in contrast to the popular hypothesis within the finance discipline that stock prices evolve under a random walk (Fama, 1965). However, empirical evidence suggests at least a low level of predictability in stock returns. Finance literature has documented one aspect of stock return predictability, termed the MOM12 (Jegadeesh and Titman, 1993), defined as the change in price (adjusted for splits/dividends) from 12 months ago to 1 month ago. MOM12 was found to predict stock returns 1 month ahead. The momentum effect is pervasive and has been found to occur across many asset classes (Asness et al., 2013). Momentum has an intuitive interpretation — stocks that have increased (decreased) in value over (approx.) one year will continue to increase (decrease) in value over the next month. In other words, momentum describes the medium-term trending behaviour of stocks. In this chapter, we show that neural networks can learn this pattern directly from price data. As we will show in this chapter, this simple pattern is in fact very difficult to learn using conventional deep learning models. We argue that this is due to characteristics of stock returns that impedes learning, namely outliers, low signal-to-noise ratio, heavy tails and volatility clustering, as discussed in Section 1.5. Our contributions in this chapter are as follows:

- We propose STAE, a supervised autoencoder augmentation to TCN, a powerful convolutional network for sequential learning. The autoencoder regularises the network and provides a scaffold that can be interpreted as a nonparametric functional-form. This encourages the latent representation to retain information about the input sequence.
- We show that STAE materially improves forecasting performance of TCN and provides an economically meaningful improvement over MOM12, a well-known predictor of returns in financial literature.
- We show that the addition of autoencoder aids interpretability, by allowing the user to inspect the reconstructed input and visualise the features of the original sequence that are retained by the network.
- We establish a benchmark of neural network-based time-series forecasting performance in a large and investable set of U.S. stocks, comparing STAE to TCN, N-BEATS, LSTM and transformer. We show that STAE commands class-leading performance

in this challenging application, and that the reconstruction task is beneficial to forecasting performance even if added at a small weight.

- We provide a precedence on a set of transformations for augmenting raw price series into inputs for neural networks. We show that the network can learn useful features from this transformed series directly, eliminating the need for handcrafted features.

The rest of this chapter is organised as follows. In Section 4.2.1, we outline the problem setup. Section 4.2.2, 4.2.3 and 4.2.4 introduce existing literatures on convolutional neural networks, supervised autoencoders and financial time-series forecasting using deep learning. In Section 4.3, we describe our proposed STAE. Data and experimental setup of our empirical test is outlined in Section 4.3.1 and results in Section 4.3.2. Finally, we provide concluding remarks in Section 4.4

4.2 Preliminaries

4.2.1 Problem setup

We start with the familiar iterative asset return forecasting process of an investor. At every period $t \in \{1, \dots, T\}$, there are N stocks. We define total return index (TRI)[3] $u_{t,i} > 0$ of each stock i at t as the accumulation index, computed as the compounded change in price adjusted by dividends $u_{t,i} = u_{t-1,i}(p_{t,i} + d_{t,i})/p_{t-1,i}$, where $u_{0,i} = 1$, $p_{t,i}$ is price[4] at time t and $d_{t,i}$ is dividend at t if a dividend is paid, and zero otherwise. The input sequence is the log-transformed total return index $\boldsymbol{x}_{t,i} = \{\log u_{t-K+1,i}, \log u_{t-K+2,i}, \dots, \log u_{t,i}\}$, where $K = 250$ is chosen to be the approximate number of trading days per year and is motivated by the momentum effect (i.e., the one-year change in share price exhibiting predictive power on stock returns over the subsequent month). Further pre-processing of the sequence is outlined in Section 4.3.1

[3] As noted in Section 1.2, total return includes both change in price and dividends. TRI is the compounded accumulation index of total returns. On the day a stock pays a dividend, its share price typically falls by roughly the dividend amount. TRI adds back dividends onto the price series such that stocks that pay dividends are not unfairly penalised.

[4] For simplicity, we assume that price is already adjusted for stock splits.

The dependent variable is forward 1-month return (proxied by 20 trading days), computed as the log-difference in TRI $y_{t,i} = \log u_{t+20,i} - \log u_{t,i}$. Note that this differs from the use of percentage returns in Chapter 3 (chosen to be comparable to Gu et al., 2020). Percentage returns are not normally distributed as the left tail is limited to $-100\,\%$, while the right tail is unlimited. Log-difference of the TRI (also known as *continuously compounded return* or *logarithmic return*) is also not normally distributed due to heavy tails (Peiró, 1994) but is often assumed to be Normal for modelling purposes (Isichenko, 2021). These choices allow our time-series model to be directly compared to the momentum and reversal effects, as documented in Jegadeesh and Titman (1993). Finally, the investor's objective is to find the model $F(\boldsymbol{x}; \boldsymbol{\theta})$ that best forecasts forward 1-month returns, by minimising the expected loss,

$$\min_{F, \boldsymbol{\theta} \in \Omega} \mathrm{E}_{\boldsymbol{x}, y \in \mathcal{D}} \left[\mathcal{L}(F(\boldsymbol{x}; \boldsymbol{\theta}), y) \right].$$

In Section 4.3, we further define the model F and parameterisation $\boldsymbol{\theta}$.

4.2.2 Neural networks for time-series applications

In this section, we provide a discussion on neural network architectures that can be applied to time-series applications.

There are three broad categories of sequential neural network architectures: RNN (and its variants, as discussed in Section 2.5.1), TCN (as discussed in Section 2.5.2) and, more recently, *transformer* models (Vaswani et al., 2017). Bai et al. (2018) argued that RNN suffers from several shortcomings, namely exploding and vanishing gradients, lack of parallelism and difficulty in retaining long term memory. TCN, utilising dilated convolutions, is able to model sequence of arbitrary length by increasing the kernel size and stacking multiple dilated convolution layers. Dilated convolutions provide the network with direct gradient flow to any part of the sequence while still preserving the temporal ordering of the sequence, thereby alleviating the problem of unstable gradients in recurrent networks. Using a benchmark dataset, Bai et al. (2018) demonstrated TCN's superior performance against other popular recurrent networks, including the LSTM. TCN is further validated in other sequential applications, such as speech synthesis (van den Oord et al., 2016), weather forecasting (Yan et al., 2020) and

traffic prediction (Dai et al., 2020). Moreoever, CNNs have achieved tremendous success in conventional image recognition tasks (Krizhevsky et al., 2012; Szegedy et al., 2015; Schroff et al., 2015). Thus, TCN makes for an ideal candidate for pattern recognition in time-series applications.

Sequential neural networks are often used for natural language processing applications. Sentence structure plays an important role in languages. For example, "the cat is brown" and "a brown cat" both have the same semantic meaning and differs in *ordering* of words. However, "the dog barked at the car because it was scared" and "the dog barked at the car because it was fast" contain a simple change of words and the subject association is completely different (dog with scared and car with fast). This context dependency proved challenging for conventional recurrent networks where temporal ordering is preserved and information flow is directional. More advanced LSTM-based language models, such as the model underpinning Google Translate[5] (Wu et al., 2016), employ *bidirectional recurrence* (Schuster and Paliwal, 1997) and *attention* (Luong et al., 2015). Bidrectional recurrence utilises separate LSTMs in both directions, while attention allows a word in the sentence to be associated with any other word in the sentence, regardless of adjacency. Both of these features increase flexibility and reduce scaffolding (i.e., information flow is no longer unidirectional). Transformers take this paradigm one step further by dropping recurrence and relying solely on *self-attention* (Vaswani et al., 2017). For each element of the sequence, self-attention computes association scores with every other element in the sequence. Thus, allowing gradients to flow between any pair of elements within the sequence and is *permutation-invariant* (Zeng et al., 2022). This flexibility proved vital in recent breakthroughs in machine translation applications (Vaswani et al., 2017; Devlin et al., 2019; Brown et al., 2020). However, Zeng et al. (2022) argued that time-series modelling involves extracting information from an ordered set of data points, which runs contrary to the permutation-invariant flexibility that is emblematic of transformers. Thus, transformers are unsuitable for long-term time-series forecasting. Whilst acknowledging that transformers are designed to solve different applications than time-series forecasting, driven by their compelling performance in NLP problems, we include transformers in our benchmark of neural network models for financial time-series forecasting.

[5] https://translate.google.com

Lastly, as discussed in Section 4.1, some recent advances focus on combining statistical constructs with neural networks, such as N-BEATS, which have also shown promising results in time-series applications. Rather than taking a data-driven approach (e.g., LSTM, transformers), N-BEATS allows the user to pre-specify basis functions which form the backbone of the model. The basis functions are parameterised by the outputs of fully connected layers. The choice of basis functions can be interpreted as placing a prior on the functional-form of the time-series. In this chapter, we compare time-series forecasting performance of our proposed STAE architecture, to TCN, LSTM, transformers, N-BEATS and MOM12. The selected models represent two distinct approaches to time-series forecasting — one which emphasises on flexibility and "letting the data speak", and one which imposes functional-form restrictions and thus regularises the model.

4.2.3 Supervised autoencoders

Owing to its vast learning capacity, neural networks can also easily overfit. This is particularly problematic for noisy environments such as financial markets. Advances in improving generalisation of neural networks include dropouts (Srivastava et al., 2014), early stopping (Morgan and Bourlard, 1990) and norm regularisation. Suddarth and Kergosien (1990) first proposed using an auxiliary learning task to assist with network training. More generally, MTL has been shown to improve generalisation performance across a range of tasks, such as facial landmark recognition (Zhang et al., 2014) and natural language processing (Collobert et al., 2011). Simultaneously learning multiple tasks can reduce overfitting through shared representations and by leveraging auxiliary information in secondary tasks. Supervised autoencoder (SAE), first proposed by Le et al. (2018), are a special case of MTL where the auxiliary task is to reconstruct the input used for the supervised learning task via an autoencoder (as discussed in Section 2.5.3; LeCun, 1987; Bourlard and Kamp, 1988; Hinton and Zemel, 1993). Le et al. (2018) and Epstein and Meir (2019) provided the theoretical generalisation bounds of autoencoders and showed that the addition of reconstruction error can improve generalisation of a classifier. The reconstruction task exhibit similar stability to l_2 regularisation but without the negative bias from shrinkage. The SAE learns two contradictory

tasks. The supervised learner only wants to retain features that are relevant for the supervised task, while autoencoder wants to retain all features that are relevant for reconstruction of the original input (Epstein and Meir, 2019). Thus, the autoencoder prevents the supervised learner from discarding too many features of the original sequence. We argue that this is beneficial to learning in a noisy environment (such as financial markets) as the supervised learner may be overfitting on spurious correlations. To date, SAEs have been applied to specific tasks such as classifying biological signals (Thiam et al., 2020; Barlaud and Guyard, 2020) and dialect detection (Parida et al., 2020). In this work, we show that SAE can improve generalisation in financial time-series prediction.

4.2.4 Deep learning in financial time-series prediction

Time-series forecasting using deep learning methods have been an active area of research and has been discussed in Section 1.6.2. Sezer et al. (2020) provided a recent survey of financial time-series forecasting using deep learning. In summary, majority of existing works focus on very short horizon forecasting, such as daily return or next day's closing price, using short sequences (e.g., Li et al., 2017 used previous day's close, high, low and open prices to predict next day's closing price). Very short term strategies are typically difficult to implement in practice due to high turnover, transaction costs (commissions, bid-ask spread and market impact) and overnight slippage[6]. Many existing works are also based on a small set of stocks (e.g., Hiransha et al., 2018 is based on 5 stocks) and/or over a short history (e.g., Chandra and Chand, 2016 used 3 stocks over 2006-10). Some previous methods use neural networks to tune parameters of handcrafted features (e.g., Lim et al., 2019). Our work differs from existing works in three ways. Firstly, we forecast forward 1-month return (proxied by 20 trading days, as opposed to daily returns in many existing works) and compare forecasting performance of our proposed network to a known predictor (the momentum effect) in finance literature. We only consider "pattern recognition on stock prices" a success if the neural network can learn additional patterns from stock prices that is above and beyond the momentum effect (which is

[6]Generally, predicted daily returns (based on the expected change in closing prices of today and tomorrow) are not achievable as the positions can only be initiated at market open the follow day at the earliest. If a stock's price is expected to increase tomorrow, the opening price is also likely to be higher than today's closing price. This *close-to-open* slippage is the overnight slippage.

based on just two data points — start and end stock prices). Secondly, in keeping with the spirit of deep learning, our approach performs feature selection automatically and without the need of any feature engineering (apart from log-transform and standardising daily prices). A common transformation of time-series is to take the first difference. If this is beneficial to the forecasting task, we expect the network to learn this directly from the data. Thirdly, we provide empirical results on the largest 3,000 stocks listed in the U.S. over 1984-2020, a dataset in which existing works have not being tested in.

4.3 Proposed STAE and application to stock return forecasting

We propose to augment TCN by adding a decoder which regularises the predictor subnetwork (convolutional encoder and fully connected predictor, as depicted in Figure 4.1). We term this network the Supervised Temporal Autoencoder (STAE). As per TCN, the convolutional

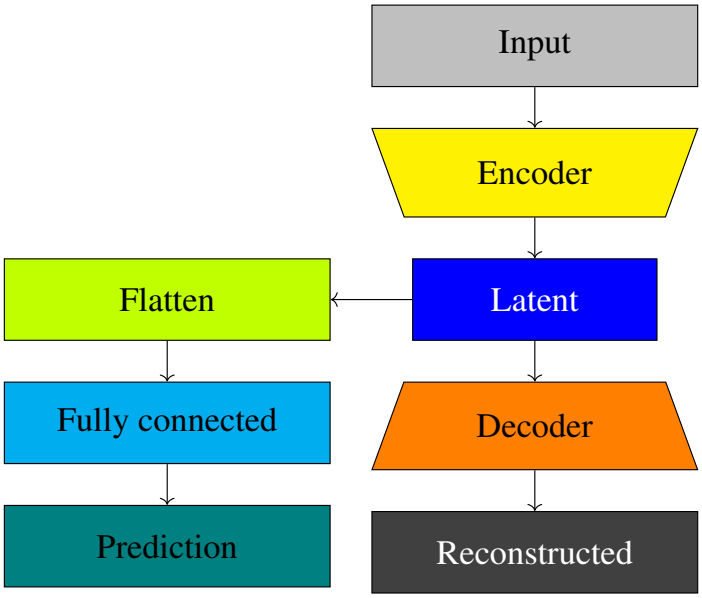
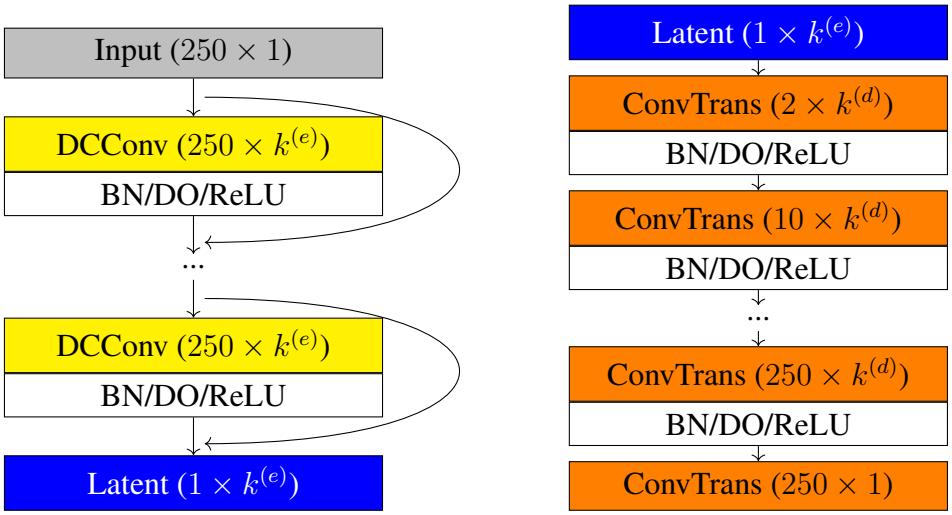


FIGURE 4.1: The Supervised Temporal Autoencoder architecture. A convolutional encoder converts input sequence into a latent representation which is used as input into one or more fully connected layers. *Prediction* produces output for the primary supervised learning task and *Reconstructed* is the reconstructed input sequence, as the auxiliary learning task.



(a) Encoder (b) Decoder

FIGURE 4.2: In 4.2(a), the encoder contains stacks of residual blocks. Each residual block consists of skip connection, dilated causal convolution (abbrev. *DCConv*), batch normalization (Ioffe and Szegedy, 2015), spatial dropout (Tompson et al., 2015) and rectified linear unit activation layers (abbrev. *BN/DO/ReLU*). In 4.2(b), the decoder uses *transposed convolution* layers (abbrev. *ConvTrans*) to reproduce the original sequence from latent representation. $k^{(e)}$ and $k^{(d)}$ are number of filters in convolutional layers of encoder and decoder, respectively. Each convolutional layer may have a different number of filters.

encoder (as depicted in Figure 4.2(a)) is organised into residual blocks (each containing dilated causal convolution, batch normalisation, dropout and ReLU layers) with skip connections between blocks. As the input are time-series, we use 1-D kernels in both the encoder and decoder. We use dilation rates of powers of 2 and allow hyperparameter search to choose between 8, 16 and 32 kernels, and kernel size of 2, 5 and 10, corresponding to daily[7], weekly and fortnightly features, respectively. For each output sequence of the last residual block, we take the last cell of the sequence as the latent representation of the entire sequence (as illustrated in Figure 2.9). The decoder uses *transposed convolutions* (also called *deconvolution*, Long et al., 2015) to recreate the original sequence from the latent representation, as illustrated in Figure 4.2(b). To reduce the hyperparameter search space, both encoder and decoder share the same number of kernels which is kept constant for all dilated convolution layers. In sum,

[7]For kernel size of 2, if the kernel learns values of $\{-1, 1\}$, then the sum product of this kernel with the input corresponds to the difference between the two data points.

STAE differs from conventional autoencoders in using temporal convolutions (as explained in Section 2.5.2 and Figure 2.9), as opposed to variants of recurrent networks such as Seq2Seq in (as discussed in Section 2.5.1; Sutskever et al., 2014) and letting reconstruction to serve only as a secondary task (rather than primary objective).

Next, we define the composite loss used to train STAE. The composite loss is the weighted sum of the prediction loss and reconstruction loss. As preliminaries, B is the size of each minibatch and $K = 250$ is the input sequence length (as defined in Section 4.2.1). Input $\boldsymbol{X} \in \mathbb{R}^{B \times K}$ is a matrix of B price sequences and $\boldsymbol{y} \in \mathbb{R}^{B \times 1}$ is vector of forward 1-month stock returns. $F(\boldsymbol{X}; \boldsymbol{\theta})$ is comprised of three sub-networks, encoder $F^{(e)}(\boldsymbol{X}; \boldsymbol{\theta}^{(e)})$, decoder $F^{(d)}(\boldsymbol{h}; \boldsymbol{\theta}^{(d)})$ and predictor $F^{(p)}(\boldsymbol{h}; \boldsymbol{\theta}^{(p)})$, where $\boldsymbol{h}$ is the latent representation produced by $F^{(e)}$, as depicted in Figure 4.1. For brevity, we use $\boldsymbol{\theta} = \{\boldsymbol{\theta}^{(e)}, \boldsymbol{\theta}^{(d)}, \boldsymbol{\theta}^{(p)}\}$ to denote weights and bias of all three sub-networks as a whole and $\boldsymbol{\theta}^{\{(e),(d),(p)\}}$ to denote weights and biases of encoder, decoder and predictor, respectively. We train network $F(\boldsymbol{X}; \boldsymbol{\theta}^{(e)}, \boldsymbol{\theta}^{(d)}, \boldsymbol{\theta}^{(p)})$ at each t using 10 years of daily prices preceding t. Further details on construction of the training set is provided in Section 4.3.1. The network is trained with *composite loss*:

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{y}; \boldsymbol{\theta}^{(e)}, \boldsymbol{\theta}^{(d)}, \boldsymbol{\theta}^{(p)}) = \ell^{(p)}(\boldsymbol{X}, \boldsymbol{y}; \boldsymbol{\theta}^{(e)}, \boldsymbol{\theta}^{(p)}) + \omega \ell^{(r)}(\boldsymbol{X}; \boldsymbol{\theta}^{(e)}, \boldsymbol{\theta}^{(d)}),$$

where $\ell^{(p)}$ is *prediction loss* (primary learning objective), $\ell^{(r)}$ is *reconstruction loss* (auxiliary learning objective), $\omega \in [0, 1]$ is the weight on $\ell^{(r)}$, and $\boldsymbol{\theta}^{(p)}, \boldsymbol{\theta}^{(e)}, \boldsymbol{\theta}^{(d)}$ are weights for predictor-, encoder- and decoder-part of the network, respectively. For brevity, we denote composite loss, prediction loss and reconstruction loss as simply $\mathcal{L}$, $\ell^{(p)}$ and $\ell^{(r)}$. We use *quadratic loss* for both prediction and reconstruction losses:

$$\ell^{(p)} = \frac{1}{B} \sum_{i=1}^{B} \left[y_i - F^{(p)}(F^{(e)}(\boldsymbol{x}_i; \boldsymbol{\theta}^{(e)}); \boldsymbol{\theta}^{(p)}) \right]^2$$

$$\ell^{(r)} = \frac{1}{B} \frac{1}{K} \sum_{i=1}^{B} \sum_{j=1}^{K} \left[\boldsymbol{x}_{i,j} - \hat{\boldsymbol{x}}_{i,j} \right]^2,$$

where $\boldsymbol{x}_i$ is the i-th row of matrix $\boldsymbol{X}$, $x_{i,j}$ is the j-th entry in the sequence $\boldsymbol{x}_i$, and,

$$\hat{\boldsymbol{x}}_{i,j} = F^{(d)}(F^{(e)}(\boldsymbol{x}_i; \boldsymbol{\theta}^{(e)}); \boldsymbol{\theta}^{(d)}).$$

Thus, $\boldsymbol{\theta}^{(e)}$ is influenced by both the prediction loss and reconstruction loss. With the loss function defined, the network is trained using SGD and early stopping (as discussed in Section 2.2). We use $\ell^{(p)}$ as the early stopping criterion rather than the composite loss as we are concerned with the best prediction performance. As the primary and auxiliary tasks have different convergence rates, using the composite loss as early stopping criterion may cause our predictor to under or overfit. We choose optimal ω as part of the hyperparameter search and expect ω to have similar behaviour to other regularisation techniques. A low ω will lead to under-regularisation and STAE will converge to TCN. A high ω will force the network to place too much focus on reproducing the input sequence and thus the prediction performance will deteriorate.

There are two distinct advantages of STAE in this context. Firstly, STAE can improve generalisation without resorting to function-form constraints in N-BEATS (as discussed in Section 4.1, function-form constraints are introduced into N-BEATS via user-defined basis functions). Secondly, by inspecting the reconstructed input from the decoder, the user can make sense of the features retained by the network and thus provide interpretability.

4.3.1 Data and experimental setup

Our empirical results are based on U.S. stock prices from CRSP. We construct TRI for each stock, adjusted by stock splits/consolidations and inclusive of dividends. We create a proxy of the Russell 3000 index by taking the 3,000 largest stocks in the U.S. at the end of every June. The same set of stocks are tracked for twelve months until the next rebalance (unless they are delisted). This broad universe ensures that there is sufficient breadth for the network to learn from but also excludes stocks with very low capitalisation that are unlikely to be investable by institutional investors. To the best of our knowledge, this universe is also broader than existing literature on time-series stock return prediction. Our dataset spans from 1984 to 2020. We use the initial 10 years, split into 7 years of training and 3 years of validation, for hyperparameter tuning. Then, for every January, we train a new network using 10 years of prices for stocks within the index in every month. The 10-year rolling window provides the network with sufficient data for training and ensures timeliness of the training set (in contrast

to an expanding window approach). The same network is used for prediction throughout the year. This provides us with 26 years of out-of-sample predictions for evaluation. Note that the results presented in this chapter differs to the published paper Wong et al. (2021) as the dataset is extended from 2018 to 2020, the models are re-trained with the addition of LSTM and transformers for comparison.

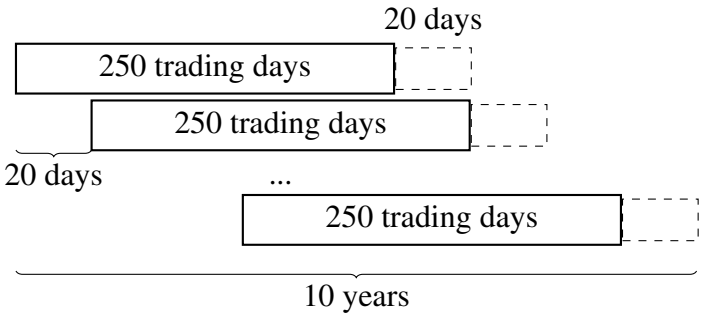


FIGURE 4.3: Training data is created by taking blocks of $250 + 20$ days of TRI.

Next, we will first describe pre-processing procedures for sequences $\boldsymbol{X}$, then expected return $\boldsymbol{y}$. Recall that in Section 4.3, we have defined sequence length $K = 250$ and y is forward 20-day return. In the 10-year rolling window, there are $\lfloor (252 \times 10 - 270)/20 \rfloor = 112$ cross-sections[8]. Let $\mathbf{T} = \{t - 20 \times (i-1) | i = 1, \ldots, 112\}, \mathsf{t} \in \mathbf{T}$ be *period counter* and,

$$\boldsymbol{x}^*_{\mathsf{t}-20,i} = \{\log u_{\mathsf{t}-K-19,i}, \log u_{\mathsf{t}-K-18,i}, \ldots, \log u_{\mathsf{t}-20,i}\}, \tag{4.1}$$

be a K-length price sequence for stock i. Log-transformation is performed in Equation (4.1) to stabilise variance (a common procedure for compounding economic time-series, Box and Jenkins, 1976; Lütkepohl and Xu, 2012). The median value of each sequence is then removed to centre the sequence[9],

$$\mu^{(\text{med})}_{\mathsf{t}-20,i} = \text{med}(\boldsymbol{x}^*_{\mathsf{t}-20,i})$$
$$\boldsymbol{x}'_{\mathsf{t}-20,i} = \boldsymbol{x}^*_{\mathsf{t}-20,i} - \mu^{(\text{med})}_{\mathsf{t}-20,i},$$

[8] Assuming an average of 252 trading days per year.

[9] Otherwise, stocks with high TRI values will denominate the training dataset once the training set is standardised.

where med is the median function. Values of all sequences $\boldsymbol{X}'_t = \{\boldsymbol{x}_{\mathsf{t}-20,i} | (i \in \mathrm{N} : i \leq N) \wedge (\mathsf{t} \in \mathbf{T})\}$ are standardised[10],

$$\boldsymbol{X}_t = \frac{\boldsymbol{X}'_t - \bar{\boldsymbol{X}}'_t}{\sigma(\boldsymbol{X}'_t)},$$

where $\bar{\boldsymbol{X}}'_t$ and $\sigma(\boldsymbol{X}'_t)$ are mean and standard deviation computed over all values of $\boldsymbol{X}'_t$, respectively. Standardised sequences $\boldsymbol{X}_t$ are then used as input into the network. Expected return is the forward 20-day return and cross-sectionally standardised,

$$\begin{aligned} y'_{\mathsf{t},i} &= \log u_{\mathsf{t}} - \log u_{\mathsf{t}-20} \\ \boldsymbol{y}'_{\mathsf{t}} &= \{y'_{\mathsf{t},i}\}_{i=1}^{N} \\ \boldsymbol{y}_{\mathsf{t}} &= \frac{\boldsymbol{y}'_{\mathsf{t}} - \bar{\boldsymbol{y}}'_{\mathsf{t}}}{\sigma(\boldsymbol{y}'_{\mathsf{t}})}. \end{aligned}$$

The training dataset is the pooled dataset, comprising of $\mathcal{D}_t = \{(\boldsymbol{x}_{\mathsf{t}-20,i}, y_{\mathsf{t},i}) | i = 1, \ldots, N \wedge \mathsf{t} \in \mathbf{T}\}$ standardised input-output pairs. This description is illustrated in Figure 4.3. $\boldsymbol{X}_t$ is standardised as a whole (i.e., elementwise) to preserve relative volatility of sequences[11]. For $\boldsymbol{y}_t$, the mean return of the cross-section represents the market return which may be difficult to forecast. $\boldsymbol{y}_{\mathsf{t}}$ is standardised cross-sectionally to remove the market return[12]. This treatment of $\boldsymbol{y}_{\mathsf{t}}$ is consistent with prior works (e.g., Fischer and Krauss, 2018). In effect, the neural network learns to predict a *score* drawn from $\mathrm{N}(0, 1)$. Predictions of the network $\hat{\boldsymbol{y}}_t$ are transformed

[10] Note that before standardisation, values are first winsorised at 1 % to remove outliers. This is applied to both $\boldsymbol{X}'_t$ and $\boldsymbol{y}'_t$.

[11] Consider two sequences, one where daily returns have standard deviation of 10 % and another has standard deviation of 1 %. Standardising $\boldsymbol{X}'_t$ as a whole preserves both the shape of the sequence (e.g., upward or downward trending) and relative volatility, while standardising each sequence individually does not preserve relative volatility and standardising by date does not preserve shape of the sequence.

[12] In our portfolio selection problem, we are only concerned with relative performance between stocks. Thus, it is safe to remove the market return (i.e., the cross-sectional mean). In Section 3.5.4, we have shown that this formulation improves the neural network's ability to predict stock returns.

back into the original distribution (in units of return) by,

$$\mu_t^{(\mathcal{D})} = \frac{1}{|\mathsf{T}|}\sum_{\mathsf{t}\in\mathsf{T}} \bar{\boldsymbol{y}}'_{\mathsf{t}}$$
$$\sigma_t^{(\mathcal{D})} = \frac{1}{|\mathsf{T}|}\sum_{\mathsf{t}\in\mathsf{T}} \sigma(\boldsymbol{y}'_{\mathsf{t}})$$
$$\hat{\boldsymbol{y}}'_t = \hat{\boldsymbol{y}}_t \sigma_t^{(\mathcal{D})} + \mu_t^{(\mathcal{D})}.$$

As noted in Section 4.2.2, we compare STAE to MOM12[13], TCN, LSTM, transformers and N-BEATS. TCN and STAE use mostly identical hyperparameter ranges. For N-BEATS, we use trend and generic models, and search over a range of polynomial dimensions. For LSTM, we search over the number of LSTM layers, followed by a fully-connected layer. For transformers, we search over the complexity of the multi-head attention block, followed by a fully-connected layer. Appendix A4 outlines the full sets of hyperparameters of each model. We train 10 networks for each model and compare the average performance. To gauge performance, we compare average cross-sectional MSE (Equation (2.4)), IC (Equation (3.2)), mean decile return (Equation (3.4)) and Sharpe ratio (Equation (3.3)).

4.3.2 Main empirical results

We start by discussing what the STAE "sees". Figure 4.4 records the input to the models (standardised log TRI of Facebook Inc.) and the reconstructed sequences at various reconstruction loss weights. Note that auxiliary loss weight $\omega = 1$ means both prediction and reconstruction have equal weight and will thus be influenced by what the predictor sees as important. There is a bias towards zero at the beginning of the sequence. This is due to causal padding which appends zeros to the start of the sequence. Thus, with a kernel size of 5, the first convolution involves 4 zeros and the first value of the sequence. Across the various ω, the reconstructed sequence tracks the overall shape of the true sequence reasonably well, with the exception of the local minima during 2018. Based on Figure 4.4, we interpret that STAE sees a general upward sloping trend.

[13] MOM12 is return over 11 months. We convert it into a monthly forecast by dividing by 11.

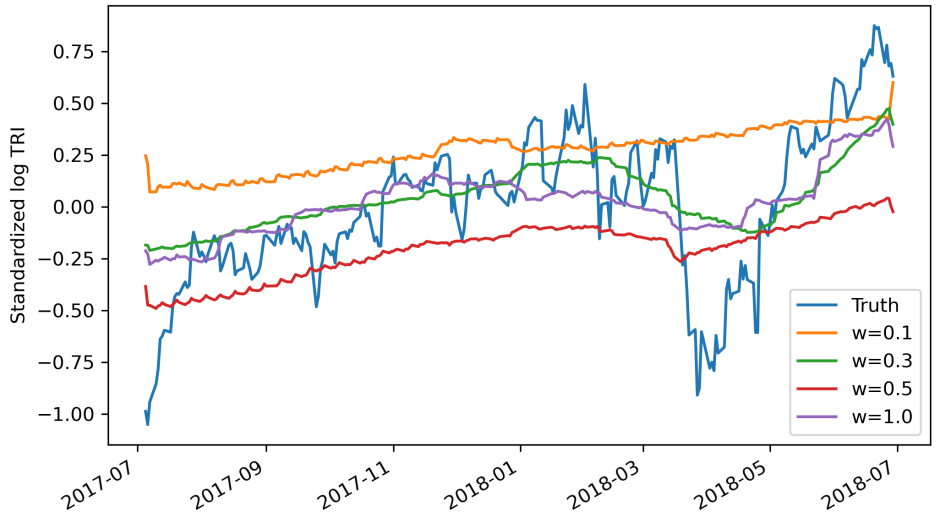


FIGURE 4.4: Standardised log TRI of Facebook Inc. and reconstructed time-series at various ω.

TABLE 4.1: **Main results**: Mean forecasting performance (of 10 networks scored individually) and performance of the ensemble (abbrev. *Ens.*) over the out-of-sample period (1994–2020). Decile return is mean difference in monthly returns of top and bottom deciles based on ensemble forecasts. Sharpe ratio is calculated as annualised decile returns divided by annualised standard deviation of decile returns. Best values in **bold**.

Metric	MOM12	N-BEATS	LSTM	Transformer	TCN	STAE
Mean IC (%)	1.77	1.76	2.11	2.25	1.74	**2.76**
Std Dev of IC (%)		0.32	0.41	0.28	0.91	**0.25**
Mean MSE	0.0205	0.0176	0.0176	0.0176	**0.0175**	0.0176
Ens. IC (%)	1.77	2.20	3.20	2.61	2.92	**3.36**
Ens. MSE	0.0205	0.0176	0.0175	0.0176	**0.0175**	0.0176
Decile Return (%)	0.67	0.57	**1.11**	0.57	0.92	1.05
Sharpe Ratio	0.25	0.25	**0.58**	0.22	0.40	0.45

Next, we turn to forecast accuracy. IC (introduced in Section 3.2.1) is our primary performance measure and is a widely used performance metric in investment management (Grinold and Kahn, 1999; Fabozzi et al., 2011b). We present two types of IC to illustrate the effects of ensembling. First, *mean IC*, given as the average IC of the 10 networks in the ensemble (computed for each network individually, then average is taken). Second, *ensemble IC* (denoted Ens. IC), given as the IC of the ensemble forecasts of the 10 networks (forecasts of the 10 networks are first averaged to produce the ensemble forecasts, then IC is computed).

The IC measure in Chapter 3 corresponds to ensemble IC. Similarly, we also report mean and ensemble MSE for the average of the 10 networks and the ensemble, respective. Consistent with Chapter 3, we also report decile return spread (Equation (3.4)) of the ensemble forecasts and Sharpe ratio of the decile return spread (Equation (3.3)). Table 4.1 records out-of-sample performance over 1994 to 2020. Overall, STAE achieved the highest IC for both the ensemble forecast (3.36 %) and each individual network (on average, at 2.76 %). LSTM is ranked second on IC, at 3.20 % for the ensemble and 2.11 % for the average over 10 networks. Transformer scored second on mean IC at 2.25 % (over 10 networks) but IC of the ensemble is only ranked fourth, at 2.61 %. Mean monthly decile return for LSTM is 1.11 %, marginally higher than STAE at 1.05 %. Sharpe ratio is also marginally higher at 0.58, compared to STAE at 0.45. For practitioners, IC, decile returns and Sharpe ratio are important performance metrics (Sharpe ratio and decile returns are related as Sharpe ratios are derived from decile returns, and are used by Gu et al., 2020). However, decile returns focus on top and bottom 10 % of forecasts without accounting for the middle 80 % of the distribution of forecasts. There are investment strategies that rely on less extreme return forecasts. In general, IC provides a more complete pictures of prediction performance by incorporating all forecasts. The higher IC of STAE reflects better ranking of stocks across the whole distribution, despite having similar decile returns. Both STAE and LSTM are economically meaningfully better than MOM12, with IC of 1.77 %, mean decile return of 0.67 % and Sharpe ratio 0.25. Comparing STAE to TCN, STAE is better on IC, mean decile return and Sharpe ratio. All 5 machine learning models achieve similar MSE of 0.0175–0.0176. Both TCN and LSTM appear to benefit more from ensembling, with IC of the ensemble forecast being 50 % to 60 % higher than the average IC of the individual models. This is compared to an increase of only 22 % for STAE and 16 % for transformer. In the case of STAE, we speculate that by regularising the network using an autoencoder, the networks are slightly more correlated to each other and thus reducing the benefits of ensembling. This is explored in more details in Section 4.3.4. Finally, N-BEATS has not performed well in our test. Due to the complexity of hyperparameter combinations, it is possible that the optimal hyperparameters lie outside of the search range.

Figure 4.5 records cumulative decile returns of MOM12, STAE, TCN, LSTM, transformer and N-BEATS. Cumulative returns of STAE, TCN and LSTM are significantly higher than

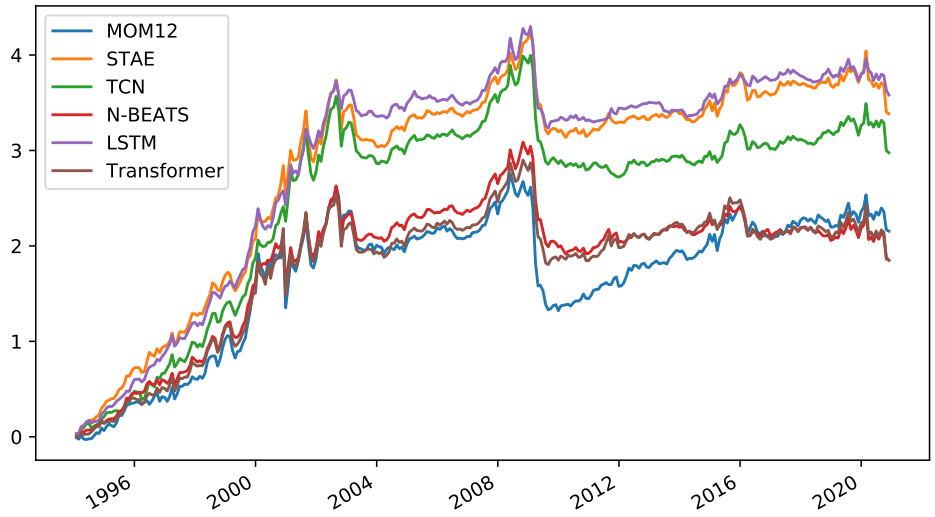


FIGURE 4.5: Cumulative decile returns based on ensemble forecasts of each model. Decile returns are calculated as mean top decile returns less mean bottom decile returns.

MOM12, transformer and N-BEATS. All 6 strategies experienced a "crash" in March 2009, as the U.S. market rebound from the depth of the global financial crisis. This is to be expected, as the input into the models are stocks' own price history and does not include information about the prevailing economic environment. A similar but smaller crash is noted at the end of the Dot-com bubble in 2003. One notable feature is the lower forecast efficacy of all models after the Dot-com bubble[14] We hypothesise that market efficacy has improved following the wide spread adoption of computers since the early 2000s. This constitutes *concept drift* and is discussed in Section 1.5.

TABLE 4.2: **Validation results**: Mean forecasting performance (of 10 networks) over the validation period (1991–1993). MSE is based for standardised returns and are not comparable to Table 4.1. Best values in **bold**.

Metric	**MOM12**	**N-BEATS**	**LSTM**	**Transformer**	**TCN**	**STAE**
Mean IC (%)	5.45	7.07	6.38	7.15	6.99	**8.50**
Mean MSE		0.9953	0.9963	0.9951	0.9955	**0.9929**

Next, we examine forecasting performance on the validation set, recorded in Table 4.2. STAE leads other models on mean IC (over 10 networks) by a large margin, scoring $8.50\,\%$. TCN,

[14]Cumulative returns in Figure 4.5 show relatively steeper inclines until 2003, then a more benign profile after 2003.

LSTM, transformer, N-BEATS and MOM12 scored $6.99\,\%$, $6.38\,\%$, $7.15\,\%$, $6.99\,\%$ and $5.45\,\%$, respectively. MSE is also the lowest for STAE compared to other models. Note that MSE here is computed using cross-sectionally standardised monthly returns (as used in network training and validation), not raw returns in Table 4.1. Comparing IC of MOM12 in the validation set to the out-of-sample set, IC fell from $5.45\,\%$ to $1.77\,\%$. Similarly for STAE, mean IC fell from $8.50\,\%$ to $2.76\,\%$. This indicates a general decline in return predictability and is consistent with our observations in Figure 4.5.

4.3.3 Explaining the predictions of STAE

In this section, we examine the predictions of STAE in relations to two well-known anomalies in finance literature — the *momentum* and *reversal* effects.

Jegadeesh and Titman (1993) found two patterns in U.S. stock prices — a medium term trending effect (termed *momentum*, as discussed in Section 4.1), and a short term reversal effect (MOM1) where stocks that rose (fell) the most over the current month tend to reverse in the subsequent month. If the only pattern that exists in prices is momentum, then we expect predictions produced by the neural network to be highly correlated with MOM12. Conversely, if the only pattern that exists in stock prices is reversal (i.e., stock prices are oscillating within a range), then neural network predictions will be correlated with MOM1. We conjecture that the two patterns can be conceptualised as alternating periods of trending and reversal patterns, as illustrated in Figure 4.6.

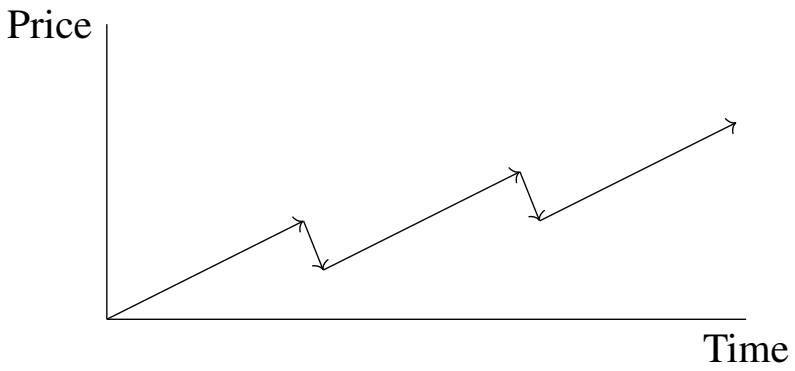


FIGURE 4.6: A hypothetical illustration of momentum and reversal patterns in stocks.

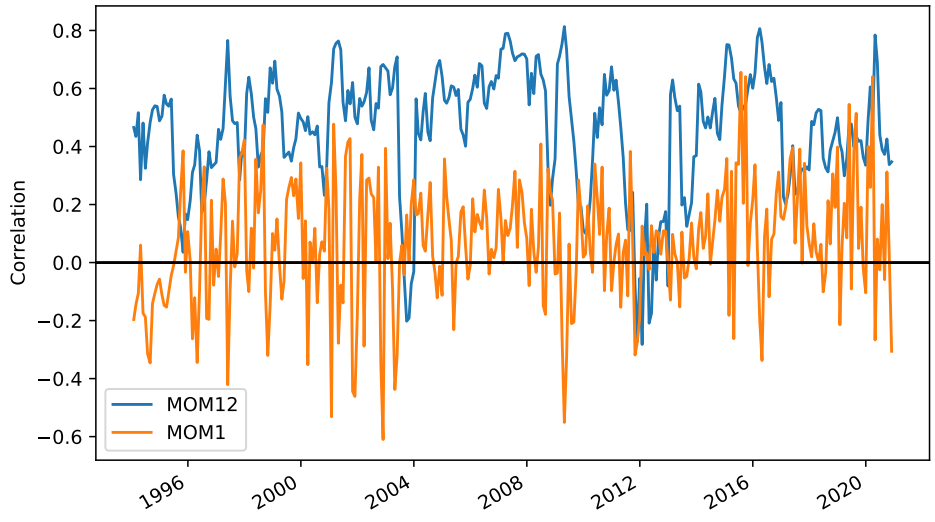


FIGURE 4.7: Cross-sectional correlations of the ensemble prediction of STAE to MOM12 and MOM1.

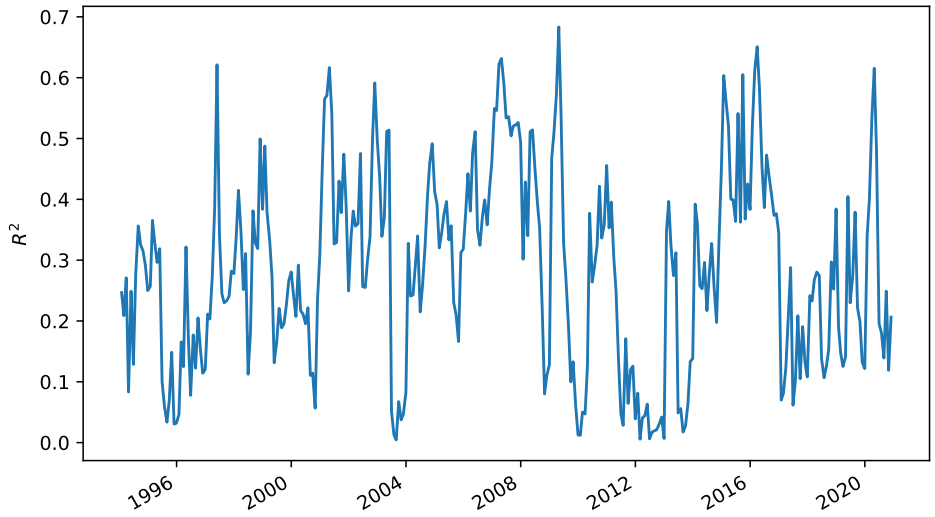


FIGURE 4.8: R^2 of the cross-sectional regression: $\hat{y}_i^{(STAE)} = \beta_0 + \beta_1 MOM12_i + \beta_2 MOM1_i$. Mean R^2 is 28 %.

Momentum and reversal patterns are simple 12-month and 1-month change in price, respectively. We argue that if a neural network were to learn complex, non-linear patterns from stock prices directly, the resultant predictions will be correlated with both MOM12 and MOM1. Moreover, when regressing ensemble predictions of STAE ($\hat{y}_i^{(STAE)}$) on MOM12 and MOM1 scores,

$$\hat{y}_i^{(STAE)} = \beta_0 + \beta_1 MOM12_i + \beta_2 MOM1_i,$$

we would expect the R^2 to be relatively low. Figure 4.7 records the cross-sectional correlations of ensemble predictions of STAE to MOM12 and MOM1. Mean correlations to MOM12 and MOM1 are 0.46 and 0.07, respectively. This suggests that STAE's predictions tend to be driven by trends, as shown by the relatively higher correlation to MOM12 than MOM1. Correlations to both momentum and reversal patterns are highly variable over time. Maximum (minimum) correlations to MOM12 and MOM1 are 0.81 (-0.28) and 0.66 (-0.61), respectively. We note that correlation to MOM12 tends to be lower after a market downturn, namely the end of 2003 (right after the Dot-com bubble) and 2011-12 (after global financial crisis). However, we do not observe a fall in correlation in 2020 during the pandemic. This may be because the 10-year rolling window is chronologically split into 7 years of training data and 3 years of validation data (used for early stopping). Patterns observed during a crisis are only visible to the network 3 years after they occurred. This underlies the reason we are yet to observe a divergence in correlation to MOM12 in 2020. Figure 4.8 records R^2 of regressing ensemble predictions of STAE on MOM12 and MOM1. Mean R^2 is relatively low, at $28\,\%$, indicating that STAE is extracting non-trivial patterns from stock prices that cannot be explained by simple trend and reversal patterns.

4.3.4 Further analysis of the reconstruction task

In this section, we provide further analysis on the regularisation effects of the reconstruction task.

The benefit of ensembling can be illustrated by analysing the expected loss of the ensemble predictor (Goodfellow et al., 2016),

$$\begin{aligned} \mathrm{E}\left[\left(\frac{1}{U}\sum_{i=1}^{U}\epsilon_i\right)^2\right] &= \frac{1}{U^2}\,\mathrm{E}\left[\sum_{i=1}^{U}\left(\epsilon_i^2 + \sum_{j\neq i}\epsilon_i\epsilon_j\right)\right] \\ &= \frac{v}{U} + \frac{U-1}{U}c. \end{aligned} \tag{4.2}$$

where U is number of predictors in the ensemble, $\epsilon_i \sim \mathrm{N}(0, v)$ is the error incurred by model i, which is assumed to be drawn from a $\mathrm{N}(0, v)$ distribution, and c is the expected

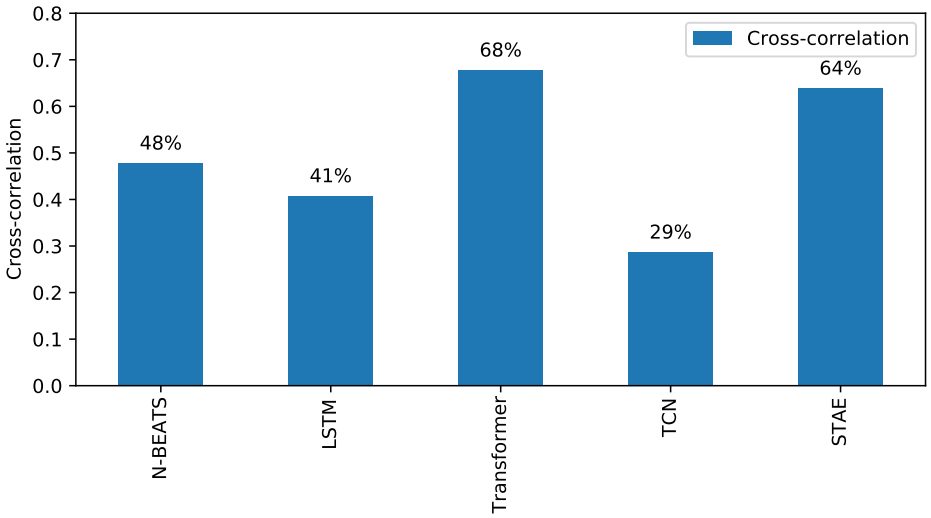


FIGURE 4.9: Mean cross-correlation of every network to every other network in the ensemble (of 10) for each model used in Section 4.3.2.

covariance between any two models of the ensemble. Equation (4.2) shows that expected loss of the ensemble predictor is lower bound by the variance of individual models scaled by size of the ensemble. That is, if each model of the ensemble is independent of all other models, then expected loss of the ensemble predictor decreases logarithmically by size of the ensemble. This "diversification" benefit is offsetted by positive correlations between models in the ensemble. The higher the correlation between models of the ensemble, the higher the expected loss of the ensemble. Due to random weight initialisation and non-convexity, every neural network will be different, even though they are trained on the training set. In Table 4.1, we observe that ensembling has a greater positive impact on TCN and LSTM than STAE and transformer. This is due to higher cross-correlation between networks within each ensemble for STAE and transformer, as shown in Figure 4.9. The mean cross-correlations for STAE and transformer are 0.64 and 0.68, respectively. These are significantly higher than LSTM and TCN, at 0.41 and 0.29. In the case of STAE, we hypothesise that the auxiliary learning task is imposing a non-parametric structure on the representation of the sequence. Thus, predictions by different networks are more correlated.

Next, we investigate whether higher weight assigned to the auxiliary task increases correlation and its impact on forecast accuracy. In this experiment, we fix the encoder kernel size to 2 with

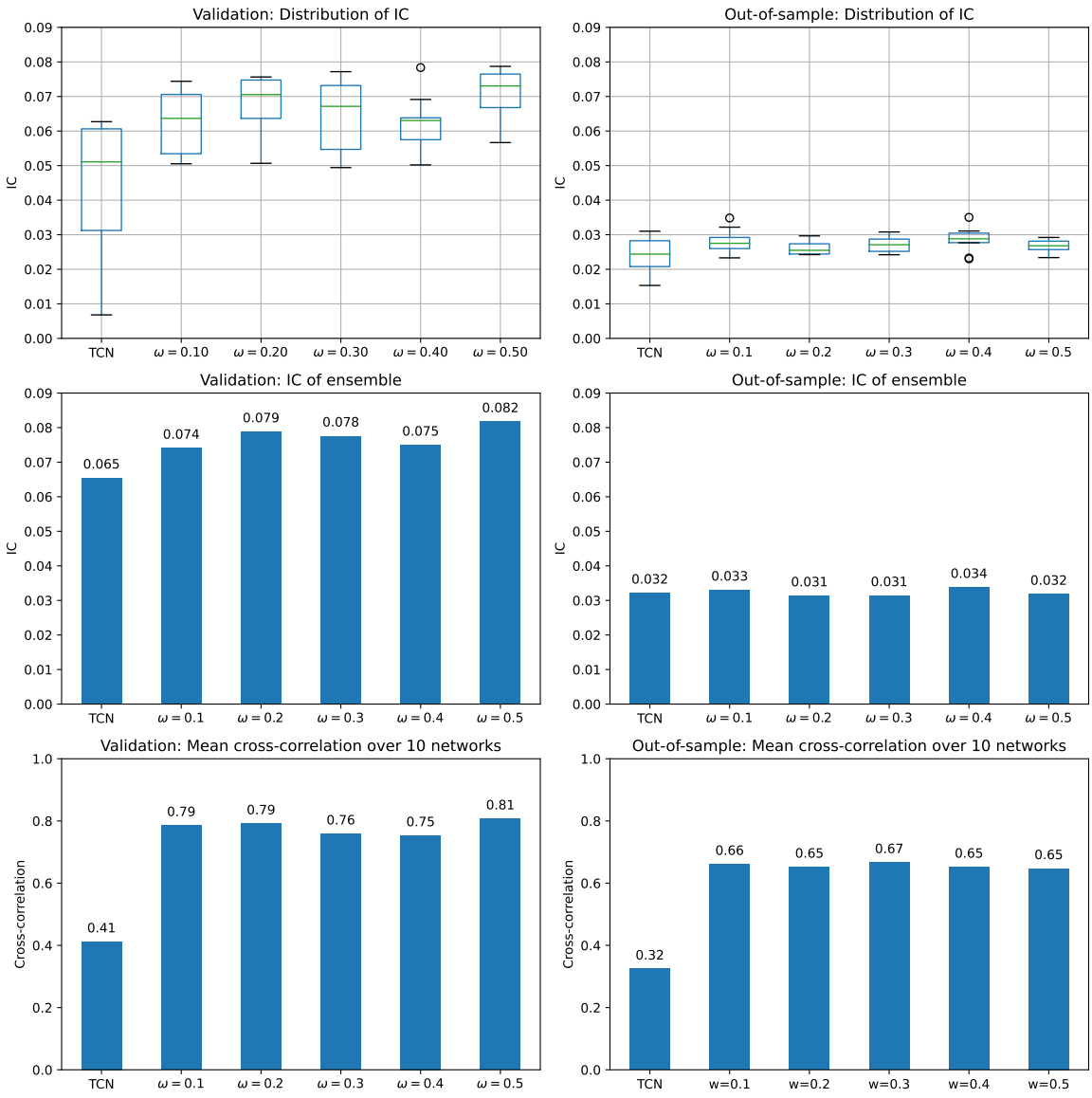


FIGURE 4.10: **Top row**: Distribution of IC for each network of the ensemble for TCN and STAE at different auxiliary loss weights (ω). **Middle row**: IC of the ensemble predictor for TCN and STAE. **Bottom row**: Mean cross-correlation between the predictions of each network of the ensemble.

16 filters, and only vary auxiliary loss weight $\omega \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. We compare STAE at various ω against TCN (i.e., $\omega = 0$) on IC of each network (in an ensemble of 10), IC of the ensemble predictor (i.e., mean prediction of the 10 networks), and average cross-correlation of each network to every other network in the ensemble. This experiment is performed on the validation set and in the out-of-sample set. Figure 4.10 records results of the experiment. In

the top row, we observe that even a small weight to the auxiliary task (at 0.1) is beneficial to IC. All values of ω achieved better median IC (over ensemble of 10) than TCN but the trend between IC and ω is not monotonic. The range of IC varies widely for TCN in both the validation set and the out-of-sample set. As all networks use the same hyperparameters, this indicates that TCN is prone to being stuck in unfavourable local minimas. In the middle row, we observe that all values of ω achieved better mean IC for the ensemble prediction than TCN in the validation set. All models achieved similar IC in the out-of-sample set. As discussed in Section 4.3.2, we conjecture that increasing market efficiency has placed an upper bound on the information content of a stock's own price history. Finally, in the bottom row, we observe that STAE has materially higher cross-correlation between networks in the ensemble than TCN, even at low ω. Cross-correlation does not appear to escalate with ω. Based on these observations, we conclude that the auxiliary task is beneficial to the prediction task, even at a low weight. Finance literature has documented evidence of declining stock return predictability (e.g., Brogaard and Zareei, 2022) — an empirical finding we have also confirmed in Chapter 3 and this chapter. Main stream finance theory (the efficient market hypothesis, as discussed in Section 1.3) relates stock return predictability to the market's information processing efficiency. Hypothesising that market efficiency is improving over time, information content of price history is higher in the validation set (1991–1993) than out-of-sample set (1994–2020), a higher ω appears to be more beneficial when information content is high.

4.4 Conclusion

In this chapter, we propose to use an autoencoder to regularise a TCN for time-series stock return forecasting. We argue that this is beneficial to the supervised learning task as the convolutional filters are constrained to learn features that are useful for reconstruction of the original input and for prediction. Thus, representation sharing will reduce the likelihood of the filters learning spurious features and improve generalisation in noisy environments such as financial markets. We propose STAE, by augmenting TCN with a convolutional decoder for the auxiliary task and show that STAE provides better forecasting performance than TCN

in predicting U.S. stock returns using a time-series of TRI. The reconstructed input by the decoder also assists the user in interpreting the features learnt by the network. We show that neural networks can learn features from (transformed) price series directly, eliminating the need for handcrafted features.

There are two potential extensions to this work. On the topic of improving financial time-series forecasting, in this work, we have demonstrated that STAE outperforms other neural network architectures in forecasting stock returns relying solely on a stock's own price history. We observe a degradation in predictability over time, which we attribute to improving market efficiency and declining information content of prices. Future work can investigate providing the neural network with more information about the stock, such as its size, *OHLCV* (opening price, day's high, day's low, closing price and traded volume), CAPM beta (as discussed in Section 1.3) and measures of business performance[15]. In particular, we observe a sharp fall in decile returns when the market turns (e.g., in March 2009 and April 2020). This is an example of exogenous shock. Informing the neural network with the prevailing market condition may potentially improve its ability to anticipate turning points. To combat concept drift, observations can be time-weighted, or models can be trained in an online manner (such as using the OES algorithm as introduced in Chapter 3). For clarity, in this chapter, we have focused solely on the time-series forecasting in noisy environment problem. We hypothesise that regularising a neural network using an autoencoder has general applicability in other noisy learning environments, outside of financial time-series forecasting. In Section 6.2, we discuss ways of combining the STAE introduced in this chapter with the OES algorithm to train neural networks that can adapt to time-varying DGP and remain robust to noise.

On the topic of improving neural network forecasting in general noisy environments, we have demonstrated that the addition of an auxiliary reconstruction task helped regularise a neural network. We observe that the auxiliary task increased cross-correlation between networks in the ensemble, which decreased the effectiveness of ensembling. Potential ways to decrease cross-correlation are to use *bagging* (Hastie et al., 2020), where both features and time periods are randomly dropped to increase diversity within the ensemble, or different look

[15]Similar to the inputs used in Chapter 3, such as accounting measures of profitability and firm valuation. However, differing to Chapter 3, we can provide a time-series of these metrics instead of just the cross-section.

back windows. The auxiliary task enforces a non-parametric functional form on the latent representation of the sequence, similar to imposing a linear trend shape constraint in linear models. A potential improvement is to combine STAE with attention (Vaswani et al., 2017), where the auxiliary task provides the non-parametric overall trend and attention is applied on deviations from the trend. This decomposition combines a "noise-robust" component with an attention-component that focuses on small intricacies. We also hypothesise that the auxiliary task would also benefit LSTM and transformers. Thus, a positive finding in using a supervised LSTM autoencoder would add to the body of evidence that an auxiliary reconstructon task is beneficial to learning in financial markets.

So far in this thesis, we have examined both cross-sectional and time-series forecasting of stock returns using neural networks. In both applications, outputs of the neural network are point estimates conditional on the input. However, if we were to "bet" on the predictions of a neural network, we need to ask — *how confident are we in the predictions?* In Chapter 5, we will examine methods of incorporating elements of statistical models to provide both the conditional mean and conditional variance of the predictions.

CHAPTER 5

Quantifying neural network uncertainty under volatility clustering

---

Time-series with time-varying variance pose a unique challenge to uncertainty quantification methods. Time-varying variance, such as volatility clustering as seen in financial time-series, can lead to large mismatch between predicted uncertainty and forecast error. Building on recent advances in neural network uncertainty quantification literature, we extend and simplify Deep Evidential Regression and Deep Ensembles into a unified framework to deal with uncertainty quantification under the presence of volatility clustering. We show that a Scale Mixture Distribution is a simpler alternative to the Normal-Inverse-Gamma prior that provides favorable complexity-accuracy trade-off. To illustrate the performance of our proposed approach, we apply it to two sets of financial time-series exhibiting volatility clustering: cryptocurrencies and U.S. equities.

5.1 Introduction

Asset returns are known to exhibit irregular bursts of high volatility that cluster in time (termed *volatility clustering*; Cont, 2001). This poses a challenge to practitioners during portfolio construction which involves the trade-off of return and risk. To motivate the discussion, consider the following simple thought experiment. Suppose an investor has a model that can perfectly forecast next day's asset returns and that the investor's goal is to maximise terminal wealth. Then, on each day, the most rational decision would be to place all of the investor's wealth into the asset with the highest expected return on the next day. Next, suppose that the investor's model is a noisy estimator of future asset returns. Then, the investor may choose to diversify across multiple assets. This intuition serves as the basis

of mean-variance portfolio optimisation (Equation (1.8)) discussed in Section 1.4 and has led to the development of various models for optimal bet allocation that depend on some measures of risk, such as Kelly criterion (where optimal bet size is proportional to expected return divided by variance of expected return[1] which can be replaced by forecast uncertainty; Kelly, 1956; Byrnes and Barnett, 2018) and Bayesian-based portfolio optimisation (Black and Litterman, 1991). Forecast uncertainty can also serve as advanced warning to protect the portfolio from increasing risk. For example, if forecast uncertainty reaches a certain threshold, an investor could purchase portfolio insurance (e.g., put options which allow the investor to sell stocks to the issuer of the options at a pre-agreed price) or liquidate positions to reduce risk.

Forecast uncertainty has an important role in many applications. Such quantity is easy to obtain for statistical models such as linear regression. However, *classical* neural networks for regression problems are typically trained using MSE and provide point estimates for the mean prediction conditional on the input without regards for the conditional variance (see Goodfellow et al., 2016). As a modeller (in our case, an investor), one is concerned with *predictive uncertainty* (Gawlikowski et al., 2021). This is the total uncertainty around a point estimate. Predictive uncertainty can be decomposed into (Gruber et al., 2023): *aleatoric uncertainty*, and *epistemic uncertainty*. Aleatoric uncertainty originates from the stochastic relationship between input variable X taking value x and output variable Y (Gruber et al., 2023). As long as the conditional distribution of $Y|x$ is not degenerate (i.e., Y cannot be perfectly predicted), there will always be aleatoric uncertainty. Aleatoric uncertainty does not typically depend on sample size. By contrast, epistemic uncertainty is attributable to the model and typically scales inversely with sample size (Meinert et al., 2022). Epistemic uncertainty can be further decomposed into *model uncertainty*, which relates to the correct specification of the model, and *parametric uncertainty*, which relates to the correct estimation of model parameters (Sullivan, 2015; Gruber et al., 2023). Epistemic uncertainty refers to the part of predictive uncertainty that is *reducible* through additional information (e.g., more observations and additional variables). In practice, a clear separation between aleatoric and

[1] Note that this differs to Sharpe ratio, which is return divided by standard deviation of return. Kelly criterion is scaled by variance.

epistemic uncertainties is often impossible. To illustrate, consider the (fair) dice rolling experiment, commonly considered to be a process of pure randomness. However, if the initial position and each rotation of the dice can be measured, then it is possible to predict the outcome of each dice roll (Hora, 1996; Gruber et al., 2023). Thus, what is truly aleatoric (i.e., unpredictability of dice roll) and what is epistemic (i.e., initial position and rotation of the dice are merely missing variables) may be difficult to disentangle from a philosophical perspective.

Traditionally, neural network uncertainty quantification requires the use of Bayesian methods or evaluation of the model in unseen data (Meinert et al., 2022). A Bayesian neural network (BNN) is a full probabilistic interpretation of neural network, by placing priors on network weights and inducing a distribution over a parametric set of functions (MacKay, 1992; Neal, 1996; Gal, 2016). Modern BNNs can be trained using MCMC (e.g., the Metropolis-Hastings algorithm; Hastings, 1970) and *Variational Inference* techniques (Jospin et al., 2022). Jospin et al. (2022) notes four advantages of using BNNs over classical neural networks, with two being relevant to uncertainty quantification. First, Bayesian methods provide a natural approach to uncertainty quantification and are better *calibrated* than classical neural networks (Mitros and Namee, 2019; Kristiadi et al., 2020; Ovadia et al., 2019; Jospin et al., 2022). Second, BNN allows distinguishing between epistemic uncertainty and aleatoric uncertainty. However, despite their advantages, MCMC-based methods are computationally expensive (Quiroz et al., 2019). Thus, limiting the applicability of BNNs.

Recent advances (see Gawlikowski et al., 2021 for a recent survey) have focused on predicting the conditional distribution that is most likely to have generated the data and thus bridging the gap between BNNs and classical neural networks. In particular, using a neural network to generate parameters of a conditional distribution that is assumed to have generated the data (Lakshminarayanan et al., 2017; Amini et al., 2020) offers an attractive trade-off between adequately quantifying uncertainty and avoiding the computational cost of a full Bayesian treatment. In Lakshminarayanan et al. (2017) (the *Ensemble* method, also know as *Deep Ensembles*), regression target y is assumed to be drawn from $y \sim \mathrm{N}(\mu, \sigma^2)$, where N is the Normal distribution, μ is the expectation of y and σ^2 models aleatoric uncertainty. In this

setup, σ^2 is incapable of quantifying epistemic uncertainty. Lakshminarayanan et al. (2017) addressed this by using an ensemble of neural networks with randomly initialised weights. Each network settles in a different local minima and produces different μ and σ^2 for the same input. The variance of μ across the ensemble thus provides an estimate of epistemic uncertainty. Addressing this shortcoming, Amini et al. (2020) (the *Evidential* method, also know as *Deep Evidential Regression*) proposed to place an evidential prior[2], the NIG, on μ, σ^2. In this construct, prediction μ is assumed to be drawn from the priors: $\mu \sim \mathrm{N}(\gamma, \sigma^2\nu^{-1})$ and $\sigma^2 \sim \mathrm{InvGam}(\alpha, \beta)$, where σ^2 remains as an estimate of aleatoric uncertainty, InvGam (or IG) is the Inverse-Gamma distribution and $\sigma^2\nu^{-1}$ (with $\sigma^2 \sim \mathrm{InvGam}(\alpha, \beta)$) is estimated epistemic uncertainty. Epistemic uncertainty is linked to aleatoric uncertainty via ν, which is learnt from the data. The marginal distribution of a Normal likelihood with NIG prior is the Student's t-distribution. This mimics a Bayesian setup and circumvents the costly computational burden of MCMC methods by analytically integrating out unobserved variables. Ensemble and Evidential require only minimal modifications to a conventional neural network architecture — requiring only the NLL function of the marginal distribution as loss function and a new output layer. Evidential has been applied to navigation (Liu et al., 2021; Cai et al., 2021; Singh et al., 2022) and medical fields (Soleimany et al., 2021; Li and Liu, 2022), and has been extended into the multi-task learning domain (Oh and Shin, 2022). Multivariate models related to Evidential include the *Natural Posterior Network* which also uses a conjugate prior (NIG for regression problems and Dirichlet for categorical classification problems; Charpentier et al., 2021), and *Regression Prior Networks* which uses a Normal-Wishart prior (Malinin et al., 2020).

However, more recent works have highlighted weaknesses of the Evidential method. *Scoring rules* are a class of loss functions that measure the discrepancy between a predicted distribution and the observed distribution (Gneiting and Raftery, 2007). A scoring rule is *proper* if the score is maximised when the discrepancy is minimised, and is *strictly proper* if the maximum is unique. Thus, strictly proper scoring rules provide attractive loss functions for scoring probabilistic forecasts. Evidential can be interpreted as a hierarchical method with a prior

[2] In contrast to conventional priors in Bayesian inference where the modeller has to specify the parameters of the prior distribution, the evidential prior (e.g., NIG in Evidential) learns these hyperparameters from the data. Note that NIG is a conjugate prior to the Normal distribution (Bernardo and Smith, 2000).

distribution that controls the data distribution. Bengs et al. (2023) argues that in order for hierarchical methods such as Evidential to comply with the requirements of proper scoring rules, rather than training on observable values of y, the predictor must be trained on the imaginary distribution around each observation that depicts its uncertainty, which cannot possibly exist. This requirement stems from the definition of proper scoring, which requires the learner be scored against the "ground truth". As α and β relate to the prior distribution in Evidential, they are not directly observed as data. Thus, hierarchical methods that estimate both the prior and likelihood parameters lack theoretical guarantees on the robustness of their estimated distributions. Similarly, Meinert et al. (2022) argued that unlike aleatoric uncertainty, epistemic uncertainty has no "ground truth" and is difficult to estimate objectively. To motivate this argument, Meinert et al. (2022) used the example of points lined up perfectly in a straight line. If one point is perturbed such that the points no longer form a straight line. Without relying on a-priori assumptions, it is impossible to perform point-wise separation of aleatoric and epistemic uncertainties (i.e., whether the single deviation is due to noise or the correctness of the linear model and its estimated slope). The marginal t-distribution of NIG is overparameterised, which leads to the finding that it is possible to minimise the NLL irrespective of ν (interpreted as "strength of the data" in Amini et al., 2020). As a further critique of the network architecture, we note that all four hyperparameters of Evidential are derived from the same latent representation outputted by the last hidden layer. The four hyperparameters can have vastly different scales (e.g., in our motivating application, γ is in scale of 0.01, while ν is in scale of 10). We consider this feature to be a weakness of these approaches as the latent representation has to provide a sufficiently rich encoding to linearly derive all hyperparameters of the distribution. Nonetheless, successful applications of Evidential on real world datasets has led Meinert et al. (2022) to conclude that Evidential is a heuristic to Bayesian methods and may be appropriate for applications that aim to capture both aleatoric and epistemic uncertainties but do not demand an accurate distinction between them, such as our motivating application.

In this work, we are concerned with neural network uncertainty quantification for time-series that exhibit *time-varying variance*, such as time-series of asset returns. We combine and extend Ensemble and Evidential into a framework (the *Combined* method) for quantifying

predictive uncertainty of this class of time-series. We propose to formulate the problem using the SMD, a simpler alternative to the NIG prior, to address some of the shortcomings highlighted by Meinert et al. (2022) and Bengs et al. (2023). In SMD, a sole Gamma prior is placed on the scaling factor of variance of the Normal distribution, rather than both the mean and variance in Evidential. This is motivated by our asset return forecasting application, where the mean is typically close to zero (in scale of 0.01) and thus uncertainty is negligible, and volatility is significantly larger (standard deviation in scale of 0.1). This is consistent with fitting a return series with models such as Generalised Autoregressive Conditional Heteroskedasticity (GARCH) (Bollerslev, 1986) in which the mean process is typically assumed zero or first order autoregressive (Carroll and Kearney, 2009). Integrating out the scaling factor of SMD results in a marginal t-distribution, of which its variance indicates the predictive uncertainty. Epistemic uncertainty is assumed to be the difference between variance of the marginal t-distribution and variance of the assumed Normal data distribution. This simplification trades off granular attribution of aleatoric and epistemic uncertainties afforded by the NIG prior but allows the reduction of the number of effective parameters by one and resolves the overparameterisation of NIG, as highlighted by Meinert et al. (2022). We also propose a novel architecture to model parameters of the marginal distribution using disjoint subnetworks, rather than a single output layer as in Ensemble and Evidential. We show through an ablation study in Section 5.4.3 that this is crucial to forecasting predictive uncertainty that closely tracks forecast error when the time-series exhibit volatility clustering. As both forecast accuracy and estimation of predictive uncertainty are important to our motivating application, we incorporate model averaging into our Combined method and show that it significantly improves forecast accuracy without significantly changing the estimated predictive uncertainty. This work also provides a template for uncertainty quantification in time-series that exhibit volatility clustering, such as time-series of asset returns.

To illustrate our contributions, we apply our proposed method to cryptocurrency and U.S. equities time-series forecasting. Cryptocurrencies are an emerging class of digital assets. They are highly volatile and frequently exhibit price bubbles (Fry and Cheah, 2016; Hafner, 2018; Chen and Hafner, 2019; Núñez et al., 2019; Petukhina et al., 2021), with large volumes of high frequency data (e.g., prices in hourly intervals) freely available from major exchanges.

This makes cryptocurrencies an ideal testbed for uncertainty quantification methodologies in financial applications. Given the extreme levels of volatility, we view cryptocurrencies as one of the most challenging datasets for this type of application. A comparison in U.S. equities is also provided which illustrates performance in conventional financial time-series. In the rest of this paper, we first describe the setup of our motivating application (asset return forecasting) in Section 5.2.1 and review of related works in Section 5.2.2. We describe our proposed framework in Section 5.3. Data description and empirical results of applying Ensemble, Evidential and Combined on cryptocurrency are presented in Section 5.4.1 and U.S. equities in Section 5.4.2. An ablation study analysing the benefits of each of our proposed enhancements is presented in Section 5.4.3. Whilst this paper is focused on uncertainty quantification in time-series that exhibit volatility clustering, in Appendix A7.1, we also provide a direct comparison to Evidential and Ensemble using the UCI benchmark datasets (non-time-series), as previously analysed in Hernández-Lobato and Adams (2015), Gal and Ghahramani (2016), Lakshminarayanan et al. (2017), and Amini et al. (2020). Finally, concluding remarks are provided in Section 5.5.

5.2 Preliminaries

5.2.1 Problem setup

The basic setup of the problem in this chapter follows that of Chapter 4. At every period $t \in \{1, \ldots, T\}$, an investor observes price history up to t and uses the preceding $\{K \in \mathbb{Z} \,|0 < K < t\}$ period returns to forecast one-step ahead returns. Similar to Chapter 4, we define an asset's return at time t as the log difference in price $r_t = \log p_t - \log p_{t-1}$ and, consistent with empirical findings in finance literature (Pesaran and Timmermann, 1995; Cont, 2001), we assume that the DGP is time-varying:

$$r_t \sim \mathrm{N}(\mu_t, \sigma_t^2). \tag{5.1}$$

Let $\boldsymbol{\zeta}_t = (\mu_t, \sigma_t^2)$ be parameters of the assumed DGP, $\boldsymbol{x}_{t-1} = \{r_{t-K}, r_{t-K+1}, \ldots, r_{t-1}\}$ be a K-length input sequence[3] using returns up to $t-1$ and $y_{t-1} = r_t$ be forward one period return. The training dataset is comprised of $\mathcal{D}_t = \{(\boldsymbol{x}_{q-1}, y_{q-1}) | q \in \mathbb{N} : q \leq t\}$ input-output pairs[4] and is essentially a set of sequences formed with a K-length sliding window and their corresponding regression targets. Our goal is to forecast y_t (which corresponds to r_{t+1}). At each t, the investor's goal is to solve the optimisation problem[5],

$$\boldsymbol{\theta}_t = \underset{\boldsymbol{\theta}^*}{\operatorname{argmin}} - \sum_{q=K}^{t-1} \log \mathrm{p}(y_q | F(\boldsymbol{x}_q; \boldsymbol{\theta}^*)), \tag{5.2}$$

where $F(\boldsymbol{x}; \boldsymbol{\theta})$ is a neural network with input $\boldsymbol{x}$ and parameters $\boldsymbol{\theta}$, $\boldsymbol{\theta} = \bigcup_{\ell=1}^{L} \{\boldsymbol{W}^{(\ell)}, \boldsymbol{b}^{(\ell)}\}$ is the set of network weights and biases and, in this context, $\mathrm{p}(y | F(\boldsymbol{x}; \boldsymbol{\theta}))$ is the likelihood of observing y based on the outputs of neural network $F(\cdot; \cdot)$ and the assumed marginal distribution. In other words, the investor is concerned with recovering the parameters $\hat{\boldsymbol{\zeta}}_t = (\hat{\mu}_t, \hat{\sigma}_t^2) \coloneqq F(\boldsymbol{x}_t; \boldsymbol{\theta}_t)$ that are most likely to have generated the observed data. In this setup, $\hat{\sigma}_t^2$ can be interpreted as an estimate of aleatoric uncertainty and is an estimate of the contemporaneous variance of the DGP at time t.

There are two parts to this problem. The first part concerns uncertainty quantification specifically for time-series that exhibit volatility clustering and is the primary focus of this work. The second part concerns advancing methods of uncertainty quantification across general applications. In Appendix A7.1, we show that our proposed approach can still benefit non-time-series problems in spite of it being designed to deal with a series of data points indexed in time order and exhibiting volatility clustering.

[3] For illustrative purposes, we have stated that the sequence only contains returns r_t. However, as discussed in Section 5.3.2, we also include squared returns r_t^2 as part of the input sequence.

[4] Note that at each portfolio selection period t, the training set can at most contain data up to $t-1$ as we have not yet observed r_{t+1}.

[5] For clarity, the case of a single asset is shown. At each t, there are N assets and the dataset is typically in a $t \times N$ layout. It is easy to see the generalisation of Equation 5.2 over N assets, where the average loss is calculated over $(t-K-1) \times N$ instances.

5.2.2 Related work

Recent advances in neural network uncertainty quantification, such as Ensemble and Evidential, have focused on outputting parameters of the assumed data distribution. As these works were originally proposed for non-time-series problems, in discussing these works, we have left out time index t but note that in our motivating application, variables are indexed by t (e.g., the assumed DGP in Equation (5.1)). The neural networks are trained using procedures similar to *maximum likelihood estimation*. In Ensemble (Lakshminarayanan et al., 2017), regression target y is assumed to be drawn from $y \sim \mathrm{N}(\mu, \sigma^2)$, where μ is the forecast of y and σ^2 models aleatoric uncertainty. The output layer of the neural network is modified to output $\boldsymbol{\zeta} = (\mu, \sigma^2)$, and the network is trained using the Gaussian NLL. As this formulation is incapable of quantifying epistemic uncertainty, Lakshminarayanan et al. (2017) used an ensemble of neural networks with randomly initialised weights to provide an empirical estimate of epistemic uncertainty. Addressing this, Amini et al. (2020) proposed to place an *evidential prior*, the NIG distribution, on the model parameters μ, σ^2 of the Normal data distribution:

$$\begin{aligned} \text{Data} &: y \sim \mathrm{N}(\mu, \sigma^2) \\ \text{NIG prior} &: \mu \sim \mathrm{N}(\gamma, \sigma^2 \nu^{-1}), \quad \sigma^2 \sim \mathrm{InvGam}(\alpha, \beta), \end{aligned} \tag{5.3}$$

where μ is assumed to be drawn from a Normal prior distribution with unknown mean γ and scaled variance $\sigma^2\nu^{-1}$, ν is a scaling factor for σ^2, and shape $\alpha > 1$ and scale $\beta > 0$ parameterise the Inverse-Normal (IG) distribution[6]. We require $\alpha > 1$ to ensure the mean of the marginal distribution is finite.

In this construct, parameters of the posterior distribution of y is $\boldsymbol{\zeta} = (\gamma, \nu, \alpha, \beta)$. Epistemic uncertainty is reflected by the uncertainty in μ, which is assumed be a fraction of σ^2 and is itself assumed to be drawn from an IG distribution. This fraction is controlled by ν, which is learnt from the data and, in an abstract sense, varies according to the amount of information in the data. Parameter ν is interpreted as the number of virtual observations for the mean

[6]Time index t has been omitted for brevity and legibility. Note that variables in this section are indexed by time for each asset: $\{y_t, r_t, \mu_t, \sigma_t^2, \gamma_t, \nu_t, \alpha_t, \beta_t\}$.

parameter μ. In other words, ν virtual instances of μ are assumed to have been observed in determining the prior variance of μ (Jordan, 2009; Amini et al., 2020).

For the NIG prior in 5.3, the marginal distribution of μ after integrating out σ^2 is a non-standardised Student's t-distribution (denoted St; Bernardo and Smith, 2000),

$$\begin{aligned} \mathrm{p}(\mu|\gamma,\nu,\alpha,\beta) &= \int_{\sigma^2=0}^{\infty} \mathrm{p}_{\mathrm{N}}(\mu|\gamma,\sigma^2\nu^{-1})\mathrm{p}_{\mathrm{IG}}(\sigma^2|\alpha,\beta)\,\mathrm{d}\sigma^2 \\ &= \mathrm{St}\left(\gamma,\frac{\beta}{\nu\alpha},2\alpha\right), \end{aligned} \tag{5.4}$$

using the fact that $\sigma^2 \sim \mathrm{InvGam}(\alpha,\beta)$ corresponds to $\sigma^{-2} \sim \mathrm{Gam}(\alpha,\beta)$. Hence, assigning a $\mathrm{Gam}(\alpha,\beta)$ prior to precision σ^{-2} in (Equation (5.3)) gives the Normal-Gamma (NG) prior and is equivalent to assigning $\mathrm{InvGam}(\alpha,\beta)$ to σ^2 which gives the NIG prior. The variance of this t-distribution is $\frac{\beta}{\nu(\alpha-1)}$. Predictions based on the NIG prior can be computed as (Amini et al., 2020),

$$\begin{aligned} \text{Prediction} &: \mathrm{E}[\mu] = \gamma \\ \text{Aleatoric uncertainty} &: \mathrm{E}[\sigma^2] = \tfrac{\beta}{\alpha-1} \\ \text{Epistemic uncertainty} &: \mathrm{Var}[\mu] = \tfrac{\beta}{\nu(\alpha-1)}. \end{aligned} \tag{5.5}$$

The marginal variance $\mathrm{Var}[\mu]$ refers to the variance of the marginal t-distribution in (Equation (5.4)) for the NIG prior. We note that ν can also be interpreted as a factor that attributes uncertainty between aleatoric uncertainty ($\frac{\beta}{\alpha-1}$) and epistemic uncertainty ($\frac{\beta}{\nu(\alpha-1)}$). If $\nu = 1$, then total uncertainty is evenly split between aleatoric and epistemic uncertainties.

Whilst not the focus of Amini et al. (2020), we note that epistemic uncertainty can be further decomposed approximately into uncertainties attributable to parameters μ and σ^2. Parameter $\mu|\sigma^2$ is normally distributed with $\mathrm{Var}[\mu|\sigma^2] = \sigma^2/\nu$ (from Equation (5.3)), and $\mathrm{E}[\sigma^{-2}] = \mathrm{E}[\frac{1}{\sigma^{-2}}] \approx \frac{1}{\mathrm{E}[\sigma^{-2}]} = \alpha/\beta$ (from the Gamma distribution of σ^{-2}). This leads to $\mathrm{Var}[\mu|\sigma^2] \approx \frac{\beta}{\nu\alpha}$,

$$\begin{aligned} \text{Model } \mu \text{ uncertainty} &: \mathrm{Var}[\mu|\sigma^2] \approx \tfrac{\beta}{\nu\alpha} \\ \text{Model } \sigma^2 \text{ uncertainty} &: \mathrm{Var}[\mu] - \mathrm{Var}[\mu|\sigma^2] \approx \tfrac{\beta}{\nu\alpha(\alpha-1)}, \end{aligned} \tag{5.6}$$

where the difference between the marginal and conditional variances of μ gives the uncertainty of σ^2.

In this construct, the marginal distribution of y after integrating out μ and σ^2 is a non-standardised Student's t-distribution (Amini et al., 2020),

$$\begin{aligned}\mathrm{p}(y|\gamma,\nu,\alpha,\beta) &= \int_{\sigma^2=0}^{\infty}\int_{\mu=-\infty}^{\infty} \mathrm{p}_{\mathrm{N}}(y|\mu,\sigma^2)\mathrm{p}_{\mathrm{NIG}}(\mu,\sigma^2|\gamma,\nu,\alpha,\beta)\,\mathrm{d}\mu\,\mathrm{d}\sigma^2 \\ &= \mathrm{St}\left(y;\gamma,\frac{\beta(1+\nu)}{\nu\alpha},2\alpha\right). \end{aligned} \tag{5.7}$$

Variance of this t-distribution is $\frac{\beta(1+\nu)}{\nu(\alpha-1)}$, which corresponds to the sum of epistemic and aleatoric uncertainties,

$$\mathrm{Var}[y] = \frac{\beta}{\alpha-1} + \frac{\beta}{\nu(\alpha-1)} = \frac{\beta(1+\nu)}{\nu(\alpha-1)}. \tag{5.8}$$

The corresponding NLL of Equation (5.7) is (Amini et al., 2020),

$$\begin{aligned}\mathcal{L}_{\mathrm{NIG}}(y|\boldsymbol{\zeta}) = &\tfrac{1}{2}\log\left[\tfrac{\pi}{\nu}\right] - \alpha\log\left[2\beta(1+\nu)\right] \\ &+ (\alpha+\tfrac{1}{2})\log\left[(y-\gamma)^2\nu + 2\beta(1+\nu)\right] + \log\left[\frac{\Gamma(\alpha)}{\Gamma(\alpha+\frac{1}{2})}\right]. \end{aligned} \tag{5.9}$$

Equation (5.9) mimics a Bayesian setup, granting classical neural networks the ability to estimate both epistemic and aleatoric uncertainty, and offers an intuitive interpretation of the model mechanics — due to uncertainty in the model parameters, the tails of the marginal likelihood are heavier than a Normal distribution. This has the effect of regularising the network and provides an avenue of estimating epistemic uncertainty. As the distribution of asset returns has heavy tails (Cont, 2001), we argue that the marginal t-distribution also provides a better fit of the data. The implementation is remarkably simple — Equation (5.9) replaces MSE as the loss function (for a regression problem) and the final layer of the network is replaced with a layer that simultaneously outputs four parameters of the marginal distribution. Clearly, modelling of γ and ν by the neural network is direct as they correspond to mean and degrees of freedom of the t-distribution. By contrast, scale of the t-distribution in Equation (5.7) is modelled through a more complex structure ($\frac{\beta(1+\nu)}{\nu\alpha}$), which reflects the two sources of uncertainty in Equation (5.6) with two additional neural network outputs: α and β.

They are the shape and scale parameters of the prior Gamma distribution for precision σ^{-2} which describe distinct characteristics of epistemic uncertainty.

As discussed in Section 5.1, aleatoric and epistemic uncertainties are difficult to disentangle. More recent works have questioned the accuracy of methods, such as Evidential, that directly estimate epistemic uncertainty through minimising the NLL of an assumed marginal distribution. Bengs et al. (2023) argues that a strictly proper loss function for Evidential involves scoring against the distribution around each observation, which cannot possibly exist. Thus, there is no theoretical guarantee that the estimated epistemic uncertainty by Evidential is reliable. Moreover, Meinert et al. (2022) notes that Equation (5.7) is overparameterised, as it is possible to minimise Equation (5.9) irrespective of ν, by: $\frac{\partial}{\partial \nu}\mathcal{L}_{\text{NIG}} = 0$, if $\beta\nu = \frac{1}{1+\nu^{-1}}$ and sending $\nu \to 0$. This is because Equation (5.7) is, by definition, a projection of the NIG distribution, and thus is unable to unfold all of its degrees of freedom unambiguously (Meinert et al., 2022). Through simulation data, Meinert et al. (2022) showed that over the course of neural network training, the estimated ν was related to speed of convergence. Thus, the estimated ν, which controls the ratio of epistemic uncertainty to aleatoric uncertainty, may not be accurate. We note that this is also evident in Equation (5.7), as ν appears in both the numerator and denominator of the scale parameter of the t-distribution in the form of $1 + \frac{1}{\nu}$. Thus, ν relates ambiguously to the the scale parameter of the t-distribution. Motivated by this observation, we propose a simpler formulation, which we detail in Section 5.3.1.

5.3 Uncertainty quantification under volatility clustering

5.3.1 Modelling forecast uncertainty using a scale mixture distribution

As discussed in Section 5.2.2, Evidential provides the ability to perform granular attribution of uncertainty to various parts of the model (e.g., Equation (5.5) and (5.6)). However, this ability comes at the cost of model complexity and the estimated epistemic uncertainty may not be reliable (as discussed in Section 5.2.2). We sought to propose a simpler formulation of the problem than Evidential while offering the ability to quantify predictive uncertainty,

which is the type of forecast uncertainty that we are most concerned about in our motivating application.

We propose to simplify the model by formulating the problem as a SMD[7] (Andrews and Mallows, 1974),

$$y \sim \mathrm{N}(\gamma, \sigma^2\nu^{-1}), \quad \nu \sim \mathrm{Gam}(\alpha, \beta), \tag{5.10}$$

where $\nu > 0$ is the scaling factor, Gam is the Gamma distribution, and $\alpha > 1$ and $\beta > 0$ are the shape and scale parameters of the Gamma distribution, respectively. Our proposed formulation effectively omits the prior on μ and places a prior on ν, the scaling factor of σ^2. We argue that uncertainty of variance can be modelled through either σ^2 or ν. In here, y is assumed to be drawn from $\mathrm{N}(\gamma, \sigma^2\nu^{-1})$, with mean γ and unknown variance $\sigma^2\nu^{-1}$ where ν is a latent variable that introduces uncertainty into the variance of the assumed Normal distribution of y. This allows flexibility to inflate the variance (by minimising ν without inflating σ^2) so as to capture the extremities of the distribution. Relative to Equation (5.7), σ^2 replaces ν in the parameter set when taking the SMD approach as σ^2 has a richer interpretation — it directly indicates the scale of the conditional data distribution. Note that in Equation (5.10), placing a Gamma prior on ν is equivalent to $\sigma^{-2} \sim \mathrm{Gam}(\alpha, \beta)$ as ν and σ^{-2} are indistinguishable in $\sigma^2\nu^{-1}$. However, this is distinct from using a NG prior as there is no Normal prior on μ in Equation (5.10).

The marginal distribution of a Normal distribution with unknown variance (Equation (5.10)) is a non-standardised t-distribution (derivation is provided in Appendix A6),

$$\begin{aligned} \mathrm{p}(y|\gamma, \sigma^2, \alpha, \beta) &= \int_{\nu=0}^{\infty} \mathrm{p}_{\mathrm{N}}(y|\gamma, \sigma^2\nu^{-1})\mathrm{p}_{\mathrm{G}}(\nu|\alpha, \beta)\,\mathrm{d}\nu \\ &= \mathrm{St}\left(y; \gamma, \frac{\sigma^2\beta}{\alpha}, 2\alpha\right). \end{aligned} \tag{5.11}$$

Analogous to Equation (5.7), the shape parameter of this marginal Student's t-distribution is 2α. Equation (5.11) is similar to Equation (5.4) with y replacing μ, and can be interpreted

[7]Time index t has been omitted for brevity and legibility. Note that variables in this section are indexed by time for each asset: $\{y_t, \gamma_t, \sigma_t^2, \nu_t, \alpha_t, \beta_t\}$. We use the same notations in Equation (5.10) as Equation (5.3) where the symbols have the same meaning to improve comparability.

as the Normal distribution being "stretched out" into a heavier tailed distribution due to the uncertainty in its variance. Jointly, $\boldsymbol{\zeta} = (\gamma, \sigma^2, \alpha, \beta)$ are parameters of the SMD distribution and are outputs of the neural network. This has the effect of regularising the mean estimate (γ) and, similar to NIG, provides the ability to handle heavy tails of the distribution that characterise asset returns.

The corresponding NLL of Equation (5.11) (derivation is provided in Appendix A6) is,

$$\mathcal{L}_{\text{SMD}}(y|\boldsymbol{\zeta}) = \log\left[\frac{\Gamma(\alpha)}{\Gamma(\alpha+\frac{1}{2})}\right] + \tfrac{1}{2}\log[2\pi\sigma^2\beta] + (\alpha+\tfrac{1}{2})\log\left[\frac{(y-\gamma)^2}{2\sigma^2\beta}+1\right]. \tag{5.12}$$

Then, $\mathcal{L}_{\text{SMD}}$ is used in place of the marginal likelihood function in Equation (5.2), in which the neural network learns to output parameters in $\boldsymbol{\zeta}$. In Equation (5.10), conditional on the scaling factor ν, the data is normal with variance given by the scale of the marginal t-distribution ($\frac{\sigma^2\beta}{\alpha}$). This variance gives the uncertainty of the data. Since the predictive uncertainty given by the variance of the marginal t-distribution contains both epistemic and aleatoric uncertainties, the difference between predictive and data uncertainties gives the epistemic uncertainty. This is illustrated in Equation (5.13) below:

$$\begin{aligned}
\text{Prediction} &: \mathrm{E}[y] = \gamma \\
\text{Aleatoric uncertainty} &: \mathrm{E}[\tfrac{\sigma^2}{\nu}] \approx \tfrac{\sigma^2\beta}{\alpha} \\
\text{Predictive uncertainty} &: \mathrm{Var}[y] = \tfrac{\sigma^2\beta}{\alpha}\cdot\tfrac{2\alpha}{2\alpha-2} = \tfrac{\sigma^2\beta}{\alpha-1} \\
\text{Epistemic uncertainty} &: \mathrm{Var}[y] - \mathrm{E}[\tfrac{\sigma^2}{\nu}] \approx \tfrac{\sigma^2\beta}{\alpha-1} - \tfrac{\sigma^2\beta}{\alpha} = \tfrac{\sigma^2\beta}{\alpha(\alpha-1)}.
\end{aligned} \tag{5.13}$$

Recall that the result in Equation (5.11) can be interpreted as a Normal distribution being stretched out into a heavier tailed t-distribution when variance is unknown. Kurtosis of the t-distribution is controlled by the shape parameter (2α). In analysing Equation (5.11) and (5.13), we argue that α is analogous to "virtual observations" (ν) in NIG. Epistemic uncertainty $\frac{\sigma^2\beta}{\alpha(\alpha-1)}$ is smaller than aleatoric uncertainty $\frac{\sigma^2\beta}{\alpha}$ by a factor of $\frac{1}{\alpha-1}$, when $\alpha > 2$. Thus, as α increases, both epistemic uncertainty and scale of the marginal t-distribution monotonically decrease. Importantly, epistemic uncertainty also drops relative to aleatoric uncertainty, as the t-distribution converges to the Normal distribution on increasing α. This

stands in contrast to the model with NIG prior (Equation (5.7)), where increasing evidence ν does not monotonically lead to a decrease in scale of the t-distribution.

Our proposed SMD formulation addresses some of the concerns of Meinert et al. (2022) and Bengs et al. (2023). There are essentially three free parameters in Equation (5.11) as $\sigma^2\beta$ together should be treated as one. Parameter β is a redundant parameter as it exists as a product together with σ^2 in both the marginal NLL (Equation (5.12)) and in all three uncertainty measures (Equation (5.13)). Parameters σ^2 and β indicate scales of the Normal and Gamma distributions, respectively. Together, they contribute to the scale of the marginal t-distribution. The number of parameters can be reduced by either reparameterising $\sigma^2\beta$ as a single parameter, or by setting $\alpha = \beta$, which we consider as the more intuitive choice. SMD encapsulates several well-known distributions as special cases. According to Andrews and Mallows (1974) and Choy and Chan (2008), in the case of $\alpha = \beta$, then Equation (5.10) is a Student's t-distribution with 2α degrees of freedom, and is Cauchy if $\alpha = \beta = 1$. If $\alpha \neq \beta$, Equation (5.10) gives the Pearson Type VII (PTVII) distribution which can be re-expressed as a Student's t-distribution in Equation (5.11). As epistemic uncertainty is estimated by the heavy tails of the t-distribution, we can, without loss of generality, set $\alpha = \beta$ and reformulate Equation (5.11) as,

$$\mathrm{p}(y|\gamma, \sigma^2, \alpha) = \mathrm{St}\left(y; \gamma, \sigma^2, 2\alpha\right), \tag{5.14}$$

and the marginal NLL (Equation (5.12)) as,

$$\mathcal{L}_{\text{PTVII}}(y|\gamma, \sigma^2, \alpha, \alpha) = \log\left[\frac{\Gamma(\alpha)}{\Gamma(\alpha + \frac{1}{2})}\right] + \tfrac{1}{2}\log[2\pi\sigma^2\alpha] + (\alpha + \tfrac{1}{2})\log\left[\frac{(y-\gamma)^2}{2\sigma^2\alpha} + 1\right]. \tag{5.15}$$

Comparing Equation (5.14) to the marginal t-distribution of using a NIG prior (Equation 5.7), parameters of this model relate directly to parameters of the t-distribution instead of hyper-parameters of the prior distribution. Hence, mitigating the concerns of Bengs et al. (2023) on hierarchical models and Meinert et al. (2022) on unresolved degrees of freedom. Thus, we argue that SMD offers an attractive trade-off between model complexity and granularity, occupying the middle ground between Ensemble (no prior) and Evidential (prior on both mean and variance).

5.3.2 Architecture of the neural network

For the main application of this work, uncertainty quantification of financial time-series forecasts, we propose a novel architecture for the modelling of distribution parameters, as illustrated in Figure 5.1. To predict $\hat{y}_t$, time-series inputs of both *returns* $(r_{t-K+1}, \ldots, r_t)$ and *log-transformed squared returns* $(\log[r^2_{t-K+1}], \ldots, \log[r^2_t])$ are fed into one or more LSTM layers (Hochreiter and Schmidhuber, 1997). We log-transform squared returns to reduce skewness. The LSTM layers convert each time-series into a latent representation. The latent representation is then fed into four subnetworks, where each subnetwork is comprised of one or more fully connected layers and applies non-linear transformations on the latent representation. This allows the network to model complex relationships between the parameters in $\boldsymbol{\zeta}$ and the sequence. During training, the four parameters outputted by the network and the observed y are fed into the loss function (Equation (5.12)) to compute loss value and gradients, which are backpropagated through the network for weight updates. As noted in Section 5.3.1, we can set $\alpha = \beta$ and reduce the number of subnetworks to three. In other words, the

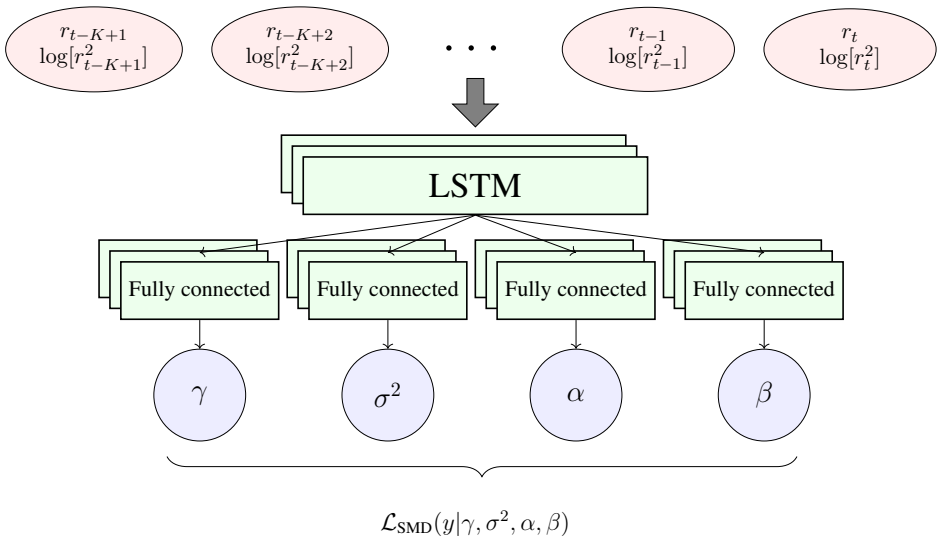


FIGURE 5.1: Input sequence (shaded in red) is passed into one or more LSTM layers. Output from the LSTM layers is then fed into four subnetworks of one or more fully connected layers with ReLU activation. The final layer of each subnetwork is a fully-connected layer with linear activation. Softplus is applied to σ^2, α and β to ensure positivity. During training, the four output values of the neural network together with the observation y are fed into the loss function (Equation (5.12)) to compute loss and gradients.

network architecture illustrated in Figure 5.1 can be modified to output three parameters: $\boldsymbol{\zeta} = (\gamma, \sigma^2, \alpha)$. We have kept β to be comparable to Evidential but provide empirical results in Appendix A8 using the UCI dataset (the same benchmark dataset used in Lakshminarayanan et al., 2017 and Amini et al., 2020, and discussed in Appendix A7.1) to show that the two networks are indeed equivalent. In the following, we explore the proposed design of the architecture in detail.

Lakshminarayanan et al. (2017) and Amini et al. (2020) introduced the `Gaussian` and `NormalInverseGamma` layers as the final layer of a neural network. These final layers output parameters of the posterior distribution. Let $\boldsymbol{a} \in \mathbb{R}^{H^{(I)}}$ be the input vector of the final layer with $H^{(I)}$ dimensions and $H^{(O)}$ be the dimension of the output layer. In the case of the `NormalInverseGamma` layer, $H^{(O)} = 4$. The `NormalInverseGamma` layer outputs,

$$\begin{aligned} \boldsymbol{\zeta} &= \mathrm{O}(\boldsymbol{a}; \boldsymbol{\theta}) = \boldsymbol{a}^{\mathsf{T}} \cdot \boldsymbol{W}^{(O)} + \boldsymbol{b}^{(O)} \\ \gamma = \zeta_1, \quad \nu &= \zeta_2, \quad \alpha = \zeta_3, \quad \beta = \zeta_4, \end{aligned} \tag{5.16}$$

where O denotes the `NormalInverseGamma` output layer, $\{\zeta_{1,\ldots,4}\}$ are 1st, ..., 4th elements of vector $\boldsymbol{\zeta}$, $\boldsymbol{W}^{(O)} \in \mathbb{R}^{H^{(I)} \times H^{(O)}}$ and $\boldsymbol{b}^{(O)} \in \mathbb{R}^{H^{(O)}}$ are weights and bias of the output layer, respectively. Each dimension of $\boldsymbol{\zeta}$ corresponds to each of γ, ν, α and β.

Outputs of the `NormalInverseGamma` layer are linear transformations of a common input $\boldsymbol{a}$ (Equation (5.16)). We argue that this construct is too restrictive for complex applications, such as in quantifying uncertainty of financial time-series forecasts, as detailed in Section 5.4.1. We propose to model each of the four parameters of SMD with its own subnetwork of one or more fully connected layers. This allows for a more expressive modelling of $\boldsymbol{\zeta}$, where each parameter may have complex, non-linear relationships with the input.

Additionally, we enforce constraints on $\sigma^2 > 0$, $\alpha > 1$ and $\beta > 0$ by applying softplus transformation with a constant term, $z' = \log(1 + \exp(z)) + c$, where $z \in \{\sigma^2, \alpha, \beta\}$ and c is the minimum value of the respective parameters. The transformed values constitute the final output of the network: $\boldsymbol{\zeta}' = \{\gamma, (\sigma^2)', \alpha', \beta'\}$. In Section 5.4.1, we show that this modification vastly improves quantification of forecast uncertainty of financial time-series.

For other network architectures, we argue that the same approach can be applied. In the case of a feedforward network, we recommend having at least one common hidden layer that reduces the input to a single latent representation. The latent representation is then passed to individual subnetworks for specialisation. We argue that the common hidden layer allows information sharing across the four parameters, while having no common hidden layer (i.e., if the input is fed into the four disjoint stacks of hidden layers directly) will prevent sharing of information across the stacks.

Machine learning models are typically trained using pooled dataset of historical observations. As such, they learn the average uncertainty within the historical data. However, as noted in Section 5.1, asset returns exhibit time-varying volatility clustering patterns. Thus, we expect predictive uncertainty to be correlated with time-varying variance of the DGP. In other words, predictive uncertainty is high when σ_t^2 of the DGP is high and the model is "surprised" by the volatility. To inform the neural network of the prevailing volatility environment, we propose to include the log of squared returns $\{\log(r_{t-K+1}^2), \ldots, \log(r_t^2)\}$ as part of the input matrix. This follows from the use of squared returns in volatility forecasting literature (Brownlees et al., 2011) and allows the neural network to infer the prevailing volatility environment.

Model averaging, as a special case of ensembling, is a well studied statistical method for improving predictive power of estimators (Breiman, 1996; Goodfellow et al., 2016), and has previously been shown to improve accuracy of financial time-series forecasting (in Chapter 4) and sequential predictions (Raftery et al., 2010). As accuracy of both return forecast accuracy and predictive uncertainty are important in our motivating application, we propose to incorporate model averaging to improve return forecasts at the cost of higher predictive uncertainty estimates. For an ensemble of M models, we compute the ensemble forecast $\tilde{y}$ and predictive variance $\mathrm{Var}[\tilde{y}]$ as,

$$\tilde{y} = \frac{1}{M}\sum_{i=1}^{M}\hat{y}_i, \quad \mathrm{Var}[\tilde{y}] = \frac{1}{M}\sum_{i=1}^{M}(\hat{y}_i^2 + \mathrm{Var}[\hat{y}_i]) - \tilde{y}^2, \tag{5.17}$$

where $\hat{y}_i$ and $\mathrm{Var}[\hat{y}_i]$ are mean and predictive variance of model i, respectively. In Equation (5.17), $\mathrm{E}[\hat{y}^2] > \mathrm{E}[\hat{y}]^2$ (by Jensen's inequality). Thus, predictive uncertainty of the

ensemble will be higher than estimated using the marginal t-distribution alone. In Section 5.4.3, we show that model averaging resulted in significant predictive performance improvement and, despite the higher uncertainty estimates, resulted in the lowest NLL.

Popular tools for modelling time-varying volatility are Autoregressive Conditional Heteroskedasticity (ARCH) (Engle, 1982) and GARCH models. GARCH, when applied to stock returns, assumes the same DGP as Equation (5.1). Time-varying variance σ_t^2 is modelled using an ARMA model (Box et al., 1994). Parameter μ_t can assume a fixed value (e.g., sample mean or 0) or modelled using time-series models such as ARMA (leading to the ARMA-GARCH formulation). In our proposed framework, squared returns are provided as inputs to LSTM in similar spirit to the autoregressive terms of squared returns in GARCH. However, our proposed framework also has few differences to ARMA-GARCH. A neural network offers greater flexibility in modelling and can automatically discover interaction effects between returns and volatility. For example, higher volatility is negatively correlated with future asset returns (known as the *leverage effect*; Cont, 2001). By contrast, modelling of interaction effects in additive models (such as GARCH) requires explicit specification by the user. LSTM can also be interpreted as having dynamic autoregressive orders (as opposed to fixed orders in GARCH). The input and forget gates of LSTM allow the network to control the extent of long-memory depending on features of the time-series. Multi-step ahead forecasting is an iterative process for ARMA-GARCH and forecast errors may compound. LSTM is able to predict multi-step ahead directly. In Section 5.4.1, we apply our framework to forecast forward 1-month U.S. stock returns using daily returns. Nonetheless, we do not directly compare against ARMA-GARCH models for two reasons. First, in this work, we are focused on advancing uncertainty quantification methodologies for neural networks. We argue that several of our advances can be beneficial to both time-series and non-time-series datasets (as demonstrated in Appendix A7.1). Second, we lean on the plethora of literature in comparing LSTM to ARMA-variants (e.g., Siami-Namini et al., 2018) and ARCH-variants (e.g., Liu et al., 2019).

For ease of comparison, we outline the differences of our method to Ensemble (Lakshminarayanan et al., 2017) and Evidential (Amini et al., 2020) in Table 5.1.

TABLE 5.1: A comparison of Combined to Deep Ensemble and Deep Evidential regressions. *Output layer* refers to the structure of output layer(s) of the network that outputs the parameters of the likelihood function.

Method	**Ensemble**	**Evidential**	**Combined**
Prior	None	NIG	Gamma
Ensemble	Yes	No	Yes
Likelihood	Gaussian	Student's t	Student's t
Output layer	Single layer μ, σ^2	Single layer $\gamma, \nu, \alpha, \beta$	Multi-layer $\gamma, \sigma^2, \alpha, \beta$

5.4 Experiments

Our proposed framework is primarily focused on advancing uncertainty quantification in time-series exhibiting volatility clustering. In this chapter, we detail experiment results in our motivating application — time-series forecasting and uncertainty quantification on cryptocurrency and U.S. equities time-series datasets, to illustrate the benefits of our proposed method. Nonetheless, SMD parameterisation, modelling distribution parameters using subnetworks and ensemble predictions can also be applied to general applications of prediction uncertainty quantification. In Appendix A7.1, we also compare our method to Ensemble and Evidential using the UCI benchmark dataset. This is intended to provide readers with a direct comparison to the results published in Lakshminarayanan et al. (2017) and Amini et al. (2020), demonstrating the benefits of our proposed improvements in non-time-series datasets.

5.4.1 Uncertainty quantification in cryptocurrency time-series forecasting

In this section, we will first describe the cryptocurrency dataset, then present empirical results on cryptocurrencies. Further confirmatory experiments on more conventional financial time-series (U.S. equities) is presented in Section 5.4.2. The same neural network architectures are used in the two datasets, with hyperparameters tuned independently. The hyperparameters used are recorded in Appendix A5.

Our cryptocurrency dataset consists of hourly returns downloaded from Binance over July 2018 to December 2021, for 10 of the most liquid, non-*stablecoin*[8] cryptocurrencies. Tickers for these cryptocurrencies are BTC, ETH, BNB, NEO, LTC, ADA, XRP, EOS, TRX and ETC, denominated in USDT[9]. Following Chapter 3 and 4, we use IC (Equation (3.2); cross-sectionally computed for each t for all 10 cryptocurrencies, then averaged over time) as a measure of predictive accuracy, in addition to RMSE and NLL. Data from July 2018 to June 2019 are used for hyperparameter tuning, chronologically split into $70\,\%$ training and $30\,\%$ validation. Data from July 2019 to December 2021 are used for out-of-sample testing. Networks are trained every 30 days using an expanding window of data from July 2018, which is preferred over a rolling window approach used in Chapter 4 due to the small sample size of the cryptocurrency dataset. Each input sequence consists of 10 days of hourly returns r and squared returns $\log(r^2)$ (i.e., each input sequence is a matrix with dimensions 240×2), and are used to predict forward one hour return (i.e., units of analysis and observation are both hourly). Network topology consists of LSTM layers, followed by fully connected layers with ReLU activation and the corresponding output layers of Ensemble and Evidential. For Combined, we use four subnetworks as illustrated in Figure 5.1. As discussed in Section 5.1, we consider uncertainty quantification in cryptocurrencies to be especially challenging due to their high volatility. Note that in this section and Section 5.4.2, "forecast uncertainty" and "uncertainty forecast" refer to estimated predictive uncertainty (i.e., sum of epistemic and aleatoric uncertainties) for simplicity.

At this point, it is useful to remind readers that prior literature have found both datasets to exhibit time-varying variance (e.g., Cont, 2001; Hafner, 2018), which is also visible in Figure 5.2. We start with the main empirical results on cryptocurrency time-series forecasting, recorded in Table 5.2. We observe that Combined has the highest average IC, lowest RMSE and NLL in the cryptocurrency dataset. This indicates that Combined has higher cross-sectional predictive efficacy (as measured by IC) and is able to better forecast uncertainty of

[8] Stablecoins are cryptocurrencies that are pegged to real world assets (e.g., U.S. Dollar). As such, they exhibit lower volatility than other non-pegged cryptocurrencies.

[9] *Tether* (USDT) is a stablecoin that is pegged to USD. It has the highest market capitalisation amongst the USD-linked stablecoins (Lipton, 2021).

the time-series prediction. Evidential has better (higher) IC and (lower) RMSE but worse (higher) NLL than Ensemble.

Next, Figure 5.2, compares predicted uncertainty and actual prediction error of the three methods to actual volatility of Bitcoin (BTC/USDT), the cryptocurrency with the highest market capitalisation, and Cardano Ada (ADA/USDT), a cryptocurrency with relatively smaller market capitalisation and higher volatility. Volatility forecasts are often compared with observed volatility (typically computed over a look back window) to evaluate forecast performance. However, the true instantaneous volatility of an asset (i.e., σ^2 in Equation (5.1)) is unobservable (Ge et al., 2022). Thus, in the top row of Figure 5.2, we use the standard deviation of hourly returns computed over each day as a proxy for σ^2. In rows 2–4, for Bitcoin, we aggregate hourly forecasts to daily data points by computing the daily RMSE of return forecasts $\sqrt{\frac{1}{24}\sum_{k=0}^{23}(y_{t-k}-\hat{y}_{t-k})^2}$ (denoted $\sqrt{(y-\hat{y})^2}$) computed from hourly return forecasts, and the daily root mean predictive uncertainty $\sqrt{\frac{1}{24}\sum_{k=0}^{23}\mathrm{Var}(\hat{y}_{t-k})}$ (denoted $\sqrt{\mathrm{Var}(\hat{y})}$), for each $t = 24, 48, 72, \ldots, T$ (note that t for cryptocurrency is in hourly units). Comparing the top row of Figure 5.2 to the root return forecast error of row 2-4 (blue line), we observe that forecast error spikes when volatility of the asset spikes. This is expected, as the spike in volatility leads to large forecast errors. Comparing the bottom three rows of Figure 5.2, which correspond to Combined, Ensemble and Evidential, respectively. We observe that Combined's predicted uncertainty of $\hat{\mu}$ tracks actual forecast error much more closely than Evidential and Ensemble. This appears to be especially true during periods of elevated volatility (e.g., during March 2020), which are important to investors. Overestimation

TABLE 5.2: Comparing Ensemble, Evidential and Combined on average IC, RMSE and NLL for cryptocurrencies time-series forecasts. Average result and standard deviation over 10 trials for each method. Best method for each dataset is highlighted in **bold**.

Metric	Ensemble	Evidential	Combined
IC (%)	2.78 ± 1.09	3.94 ± 1.84	$\mathbf{9.87 \pm 3.17}$
RMSE (%)	0.874 ± 0.022	0.874 ± 0.003	$\mathbf{0.867 \pm 0.001}$
NLL	-3.74 ± 0.10	-3.24 ± 0.02	$\mathbf{-4.14 \pm 0.01}$

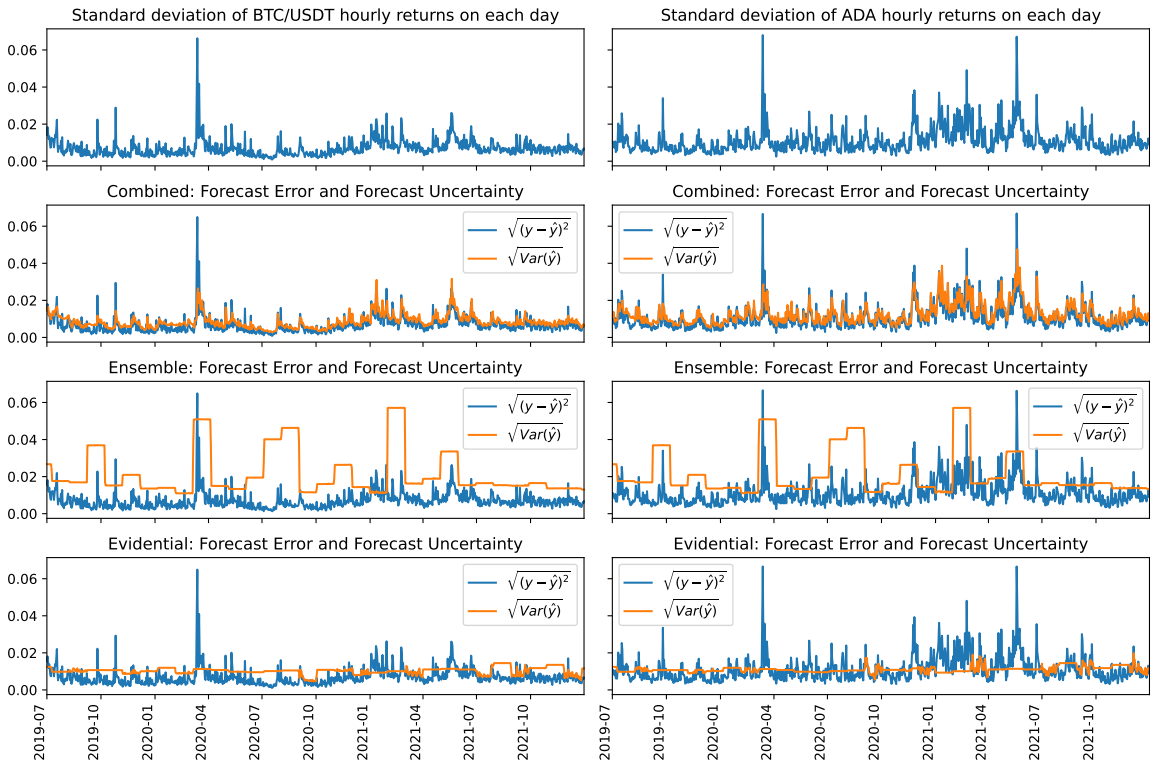


FIGURE 5.2: **First row**: Standard deviation of hourly return of BTC/USDT and ADA/USDT on each day. **Second–fourth rows**: Actual prediction error and predicted uncertainty $\mathrm{Var}(\hat{y})$ of Combined, Ensemble and Evidential for BTC/USDT (left column) and ADA/USDT (right column), respectively. Square root of the average squared error and uncertainty over each day shown.

of predictive uncertainty is severe for Ensemble in Bitcoin, where predictive uncertainty can sometimes be significantly higher than observed forecast error.

Note that the "block-like" appearances of uncertainty forecasts of both Ensemble and Evidential are due to periodic training (monthly for cryptocurrencies and yearly for U.S. equities) and the failure to generalise the prevailing volatility environment. During training, the optimiser updates network weights $\boldsymbol{W}$ and bias $\boldsymbol{b}$ (which is analogous to the intercept in linear models). When the network fails to generalise, it minimises the loss function by updating the bias rather than the weights. Thus, outputting the same constant that do not vary with the input, until the network is re-trained in the following month. This produces the block-like appearances of Ensemble and Evidential, and is indicative of the network setup (e.g., no separate modelling of

hyperparameters) being unsuitable to this class of problems. Lastly, Evidential underestimates forecast error during heightened volatility (e.g., March 2020) and overestimates forecast error under periods of low volatility (e.g., July 2020). In Appendix A9, we investigate the addition of separate modelling of distribution hyperparameters for Evidential, and conclude that both squared returns and separate hyperparameter modelling are required to achieve uncertainty forecasts that closely tracks time-varying volatility. We observe similar visual characteristics in the predicted uncertainty of other cryptocurrencies for all three methods.

5.4.2 Further results on U.S. equities

In this section, we provide empirical results of a further study on a conventional financial time-series dataset, quantifying forecast uncertainty in U.S. equities. Mimicking the S&P 500 index universe, the dataset consists of daily returns downloaded from CRSP over 1984 to 2020, for the 500 largest stocks[10] listed on NASDAQ, NYSE and NYSE American. Data from 1984 to 1993 are used for hyperparameter tuning, while 1994 to 2020 are used for out-of-sample testing. The network is refitted every January using a rolling 10-year window. We retain the same hyperparameter tuning setup to the cryptocurrency dataset, each input sequence consists of 240 trading days (approximately one-year) of daily returns r and squared returns $\log(r^2)$ (rather than 250 trading days in Chapter 4), forecasting forward 20-day (approximately one-month) return and its uncertainty. Given that 240 days cover $95\,\%$ of the 252 trading days per year, we do not expect this choice to have a material impact on the experiment results when compared to Chapter 4. Note that the unit of analysis is monthly and unit of observation is daily. One-month is a popular forecast horizon for U.S. equities in literature (e.g., Gu et al., 2020 and is used in Chapter 3 and 4), which motivated our choice of forecast horizon. The basic setup is similar to the financial time-series forecasting experiment in Chapter 4. The same models as the cryptocurrency experiment are used with separate hyperparameter tuning. Further details on hyperparameters are provided in Appendix A5.

Table 5.3 records the empirical results on U.S. equities. Again, we observe that Combined has the highest IC, and lowest RMSE and NLL out of the three methods. This demonstrates

[10]The list of stocks is refreshed every June, keeping the same stocks until the next rebalance.

TABLE 5.3: Comparing Ensemble, Evidential and Combined on average IC, RMSE and NLL for U.S. equities. Average result and standard deviation over 10 trials for each method. Best method for each dataset is highlighted in **bold**.

Metric	Ensemble	Evidential	Combined
IC (%)	0.40 ± 0.66	0.09 ± 0.93	$\mathbf{1.22 \pm 0.65}$
RMSE (%)	9.426 ± 0.044	9.433 ± 0.033	$\mathbf{9.379 \pm 0.020}$
NLL	-1.65 ± 0.17	-0.82 ± 0.03	$\mathbf{-1.71 \pm 0.01}$

the usefulness of Combined in quantifying forecast uncertainty in both time-series with extreme volatility (e.g., cryptocurrencies) and in conventional financial time-series. IC in U.S. equities are materially lower for all three methods compared to the cryptocurrency dataset. We hypothesise that this is due to both the difference in forecast horizon and maturity of the U.S. market.

Figure 5.3 compares the predicted uncertainty and actual prediction error of the three methods to actual volatility of Chevron Corp., a major U.S. oil producer, and IBM, a major U.S. technology company. As the unit of analysis is monthly, we plot the absolute error between observed monthly returns and predicted returns (denoted $|y - \hat{y}|$) in the top row of Figure 5.3, and square-root of forecast uncertainty (denoted $\sqrt{\text{Var}(\hat{y})}$) in rows 2–4 for Combined, Ensemble and Evidential, respectively. We observe similar results as the cryptocurrency experiment in the bottom three rows of Figure 5.3. Predicted uncertainty of Combined is observed to track actual forecast error more closely than Ensemble, especially during the three market crashes — the Dot-com bubble (2000–01), U.S. recession over 2008–09 and the 2020 pandemic. For Chevron, we observe an additional spike of volatility during the 2015 oil shock. Evidential produced uncertainty forecasts that are visually similar to Combined, but block-like features can still be seen in 1999 and 2012. Ensemble's predicted uncertainty for IBM jumped cover 2000–01, coinciding with a period of elevated volatility for the stock. In Figure 5.3, Ensemble exhibited less block-like appearance than in Figure 5.2. This indicates that Ensemble achieved better generalisation performance on the U.S. equities dataset than on the cryptocurrency dataset. However, Ensemble's predicted uncertainty for Chevron saw the same block-like jump which did not coincide with higher volatility of the stock. We hypothesise that generalisation for Ensemble is still problematic on the U.S. equities dataset

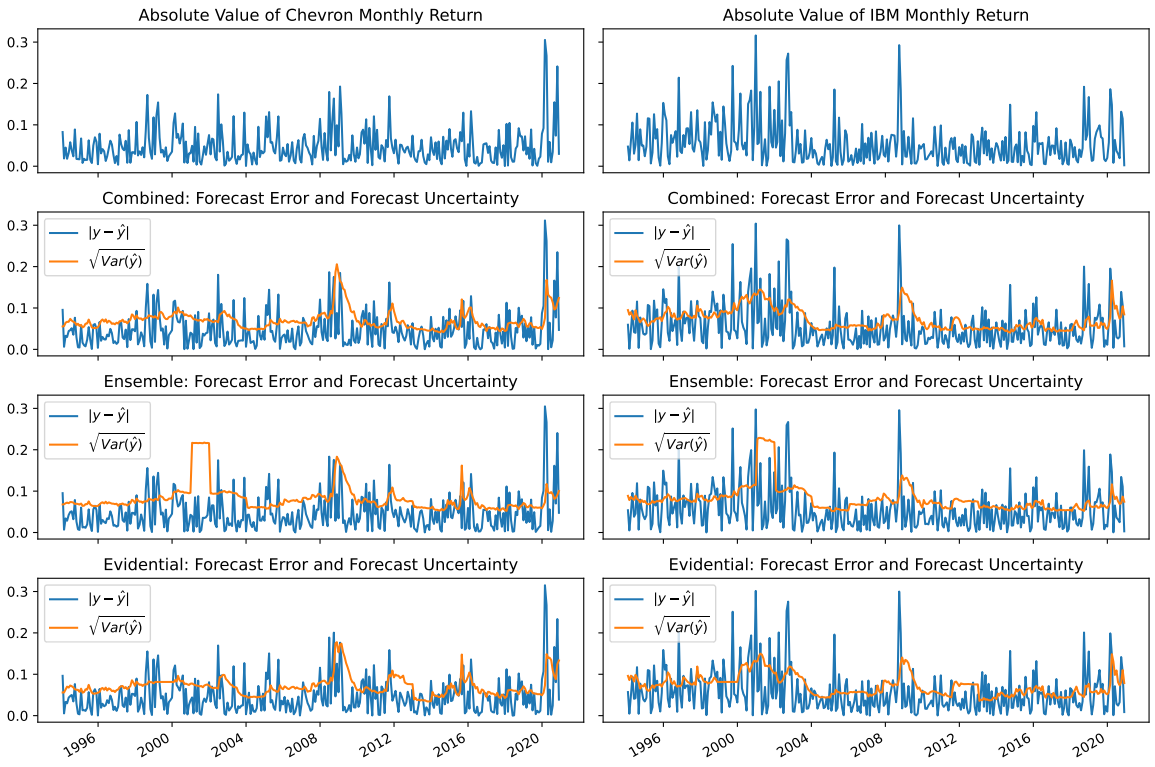


FIGURE 5.3: **First row**: Absolute monthly returns of Chevron (left) and IBM (right). **Second–fourth rows**: Actual prediction error and predicted uncertainty $\mathrm{Var}(\hat{y})$ of Combined, Ensemble and Evidential for Chevron and IBM, respectively. Square root of the monthly forecast error and forecast uncertainty shown.

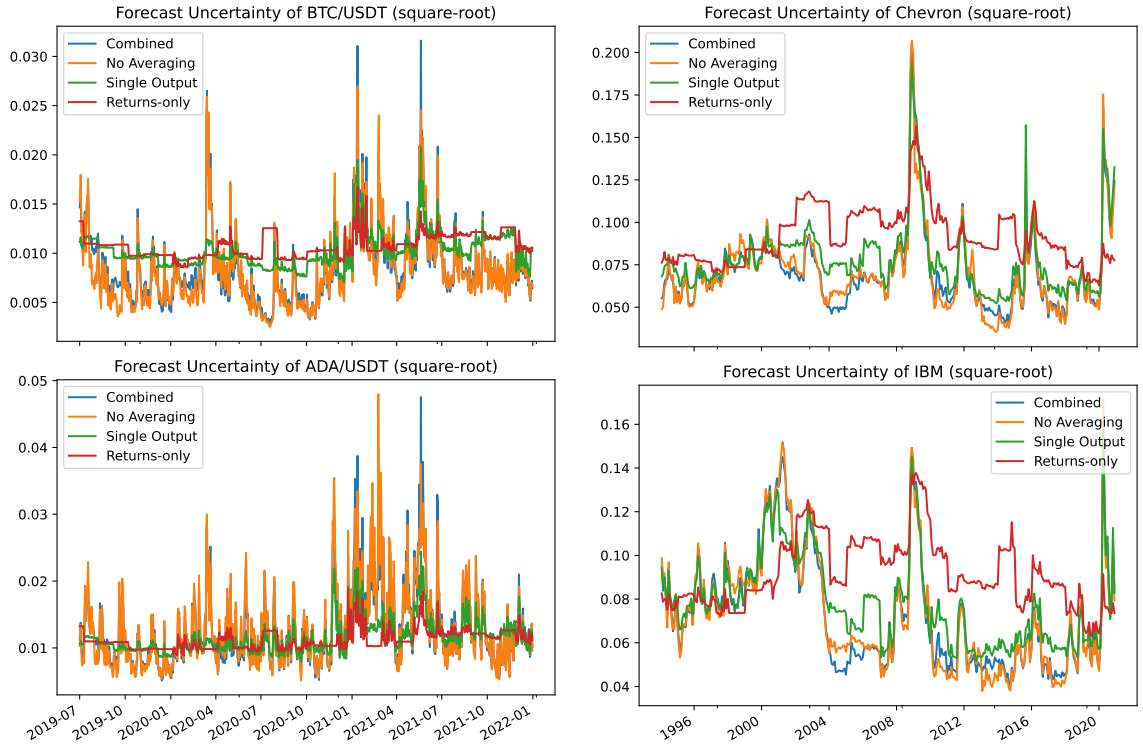
and that Ensemble failed to generalise the impact of heightened volatility environment on different stocks.

5.4.3 Ablation study

Next, we test the effects of removing each of the following for Combined: 1) model averaging; 2) single output layer for all distribution parameters (same as Evidential); 3) using return time-series only (i.e., no squared returns). The results are recorded in Table 5.4 and in Figure 5.4. As discussed in Section 5.3.2, model averaging (Equation (5.17)) will lead to higher predictive uncertainty estimates. Comparing results in Table 5.4 to the main results in Table 5.2 and Table 5.3, we observe that model averaging has a large negative impact on IC and NLL. IC

TABLE 5.4: **Ablation studies**: In each column, we remove model averaging (*No Averaging*), separate modelling of distribution parameters (*Single Output*) and using return time-series only (*Returns-only*) from Combined for cryptocurrencies (left) and U.S. equities (right), respectively. Average result and standard deviation over 10 trials are reported for each method. Note that cryptocurrency returns are hourly and U.S. stock returns are monthly.

	Cryptocurrency			U.S. equities		
Metric	No Averaging	Single Output	Returns Only	No Averaging	Single Output	Returns Only
IC (%)	4.48 ± 2.80	8.23 ± 2.91	10.46 ± 2.04	0.92 ± 0.65	1.87 ± 1.06	1.21 ± 0.73
RMSE (%)	0.868 ± 0.001	0.872 ± 0.002	0.866 ± 0.002	9.392 ± 0.020	9.398 ± 0.029	9.384 ± 0.046
NLL	-3.35 ± 0.01	-4.04 ± 0.02	-3.95 ± 0.02	-0.88 ± 0.01	-1.63 ± 0.04	-1.34 ± 0.04



(a) Uncertainty of BTC/USDT and ADA/USDT

(b) Uncertainty of Chevron and IBM

FIGURE 5.4: Square-root of predicted uncertainty $\mathrm{Var}(\hat{y})$ of Combined, without model averaging (*No Averaging*), single output layer (*Single Output*) and using returns only (*Returns-only*).

is $55\,\%$ and $25\,\%$ lower for cryptocurrencies and U.S. equities, respectively. While NLL is higher by 0.8 in both cases (lower is better), indicating a worse overall fit. However, it

does not appear to impede the network's ability to model time-series forecast uncertainty (as observed in Figure 5.4). Moreover, comparing Combined (with model averaging) to No Averaging (without model averaging) in Figure 5.4, we observe very similar estimated predictive uncertainties with and without model averaging (as the orange and blue lines track each other closely). This indicates a favorable trade-off between significantly improved forecast performance and practically the same predictive uncertainty estimates. Using a single output layer for all distribution parameters leads to marginally worse NLL. IC is lower in cryptocurrencies but marginally higher in U.S. equities. While using returns only leads to marginally higher IC but marginally lower on NLL in cryptocurrency, and lower IC and NLL in U.S. equities. From Figure 5.4, the block-like appearances indicate that both using single output layer and using returns only result in the network failing to closely track time-varying variance of the DGP. This suggests that both squared returns and separate modelling of distribution parameters are required to model time-varying forecast uncertainty.

5.5 Conclusions

Our motivating application of portfolio selection depends on both forecasts and forecast uncertainties. This is a challenging problem due to both the low signal-to-noise ratio in financial markets (Gu et al., 2020) and the presence of volatility clustering. To this end, we present a method for the simultaneous forecasting asset returns and modelling of forecast uncertainty in presence of volatility clustering. Our proposed method extends and simplifies the work of Lakshminarayanan et al. (2017) and Amini et al. (2020). We propose to use a SMD (which uses a Gamma prior for scale uncertainty ν) as a simpler alternative to a NIG prior, in which a Normal prior is placed on μ and an Inverse-Gamma prior on σ^2). Parameters of SMD are modelled using separate subnetworks. Together with ensembling and the use of second order of returns as inputs, we show that our proposed method can successfully model time-varying variance of the DGP, while providing superior forecasting performance than two state-of-the-art neural network uncertainty quantification methods — Evidential and Ensemble. This is illustrated through the successful quantification of forecast uncertainty of two financial time-series datasets: cryptocurrency and U.S. equities. Our proposed SMD

formulation offers an avenue to resolve some of the criticisms of Meinert et al. (2022) and Bengs et al. (2023). In particular, our SMD parameterisation has three effective parameters and thus does not have any unresolved degrees of freedom. We can set $\alpha = \beta$, which leads to a marginal t-distribution where the three distributional parameters $(\gamma, \sigma^2, \alpha)$ relate directly to the location, scale and shape of the t-distribution, without the need of a hierarchical model. In this formulation, epistemic uncertainty is assumed to be the difference between the predictive (t-distributed) and aleatoric (Normal-distributed) uncertainties. This assumption provides for a simpler model but lacks the granular attribution between aleatoric and epistemic uncertainties afforded by the NIG prior in Evidential. However, as Meinert et al. (2022) has pointed out, the granular control comes at the cost of an unresolved degree of freedom. Thus, users are encouraged to weigh the trade-offs in choosing a method to deploy. This also makes for a potential future research direction. We show empirically that our method is able to accurately predict forecast errors, similar to the success Evidential demonstrated in other real world applications (e.g., see Liu et al., 2021; Soleimany et al., 2021; Cai et al., 2021; Singh et al., 2022; Li and Liu, 2022). From a finance application perspective, forecast uncertainty can be used to size bets, or as advanced warning to protect the portfolio from downside risk. For example, if forecast uncertainty reaches a certain threshold, an investor could purchase portfolio insurance (e.g., put options) or liquidate positions to reduce risk. The ability to attribute epistemic and aleatoric uncertainties may also allow for more advanced portfolio optimisation techniques to be developed in future research (e.g., place different risk aversions on the two sources of uncertainties). Lastly, uncertainty quantification in time-series applications is a relatively under-explored area of literature. We believe this work can lead to further advancements of uncertainty quantification in complex time-series.

CHAPTER 6

Conclusion

Neural networks have made tremendous strides across many domains over the past decade, from mastering the game of Go[1], self-driving cars, medical diagnosis, to the personal assistant in smart phones carried by millions of people worldwide. Financial markets have proved to be a challenging problem for econometricians and statisticians. As discussed in Section 1.5, financial markets can suffer from endogenous and exogenous shocks, the distribution of asset returns has heavy tails, signal-to-noise ratio is low and, most inconveniently, the data generation process changes over time. Given the success neural networks have achieved in other domains, this begs the question — can neural networks advance the state-of-the-art in financial market applications? At the conclusion of this thesis, we believe we are a step closer to the opening of the floodgates but with much more still to be done. In the rest of this chapter, we will first outline our contributions to literature, then discuss future research directions on this topic.

6.1 Contributions to machine learning in portfolio management

In this thesis, we have provided an overview of the mechanics of a quantitative investment process, outlined the relevant finance theory that underpins stock return predictability (or the lack of), discussed a number of challenges in applying conventional statistical and machine learning tools in financial markets, identified several potential ways machine learning can be

[1] *Go* is an abstract board game and is considered as the most challenging of classic games for artificial intelligence (Silver et al., 2016).

used to improve the quantitative investment process, and proposed three distinct advances to deep learning that covers three related applications in financial market predictions.

In Chapter 3, we address the time-varying DGP problem in fnancial markets in a cross-sectional prediction application of neural networks. We proposed the Online Early Stopping algorithm for training neural networks online. Online training of a neural network is an online optimisation problem. In classical online optimisation literature, online optimisation algorithms are analysed in terms of *regret* — a measure of performance loss compared to a theoretical (but unattainable) optimum. We provide a worst-case performance bound that conforms with the notion of regret in a non-convex optimisation context, by showing that the OES algorithm can achieve tracking performance no worse than a function of the variance of the DGP. This provides an intuitive interpretation of the worst-case performance of the algorithm, where its ability to track a moving DGP is bounded by the variance of the DGP (i.e., the higher the time variability of the DGP, the more difficult it is to track and the higher the expected loss). We compare OES to a static neural network[2] and the DTS-SGD, a state-of-the-art non-convex online optimisation algorithm, in simulated data and showed superior performance in tracking a time-varying DGP. We highlight the usefulness of OES to practitioners by comparing OES to the static neural network used in Gu et al. (2020), demonstrating competitive performance and the ability to track changes in financial markets over time. In particular, we show that the OES-trained network reacts quicker (relative to the static network) to market downturns and recoveries, such as the global financial crisis and the 2015 oil shock. Finally, we show that the ensemble prediction of the static network and OES-trained network delivered the best prediction performance. Thus, we argue that OES can be a useful tool for practitioners in predicting cross-sectional stock returns.

In Chapter 4, we address the low signal-to-noise problem in financial markets in a time-series prediction application of neural networks. We propose the Supervised Temporal Autoencoder architecture, using a supervised autoencoder to regularise a temporal convolutional network. We argue that due to the low signal-to-noise in financial markets, machine learning should focus on "robust" learning as opposed to "deep" learning. The proposed supervised autoencoder

[2]In this context, a static neural network refers to a network that is trained using all available data and does not vary with time.

imposes a non-parametric functional form on the latent representation of the input sequence. We argue that this is more flexible than a fixed parametric functional form (e.g., a linear time trend). The reconstruction task provides interpretability, allowing users to inspect the smoothed reconstructed sequence to visualise the features retained by the neural network. We provide a template for financial time-series forecasting directly using price series, alleviating the need for handcrafted features. In the application test, we compare STAE to momentum, a prominent stock return predictor documented in finance literature, and showed material improvement in predictive performance — a finding that is economically meaningful to practitioners. We establish a benchmark of sequential neural network architectures in financial time-series forecasting, demonstrating superior predictive performance of STAE over TCN, LSTM and transformers. We show that the addition of the auxiliary task, even at a small weight, is beneficial to the prediction task. We document declining predictive performance of momentum, an asset pricing anomaly in finance literature, and predictions of neural networks. We conjecture that markets are becoming increasingly efficient and that information content of stock prices has decreased over time.

In Chapter 5, we advance the state-of-the-art in incorporating *risk*, in the form of predictive uncertainty, into neural network forecasts. Risk forecasting is of paramount importance in portfolio optimisation and can influence optimal bet sizes. We combine and extend two state-of-the-art methods, Ensemble (Lakshminarayanan et al., 2017) and Evidential (Amini et al., 2020), into a unified framework for quantifying time-series forecast uncertainty of neural networks. The unified framework consists of four improvements. Firstly, we propose to use the SMD instead of the NIG prior used in Evidential, arguing that SMD is simpler and offers superior numerical properties than the NIG prior, which would allow a first-order optimiser (such as SGD) to more easily traverse the loss landscape[3]. Secondly, we argue that in cases where the input has complex relations with the distribution hyperparameters (such as in financial time-series forecasting), it is beneficial to afford the neural network of the flexibility to non-linearly transform the common latent representation of the input sequence generated by the hidden layers (i.e., convolutional or recurrent layers). This is in contrary

[3]Loss landscape refers to the hyperplane spanned by network parameters (Li et al., 2018). A smooth and convex loss landscape can be easily traversed by a first-order optimiser. Conversely, a highly non-convex loss landscape with saddle points and many local minima will be difficult to traverse.

to the output layers used in Ensemble and Evidential, which compute hyperparameters of the distribution as linear combinations of the latent representation. Thirdly, we propose to incorporate ensembling, which was shown to significantly improve forecast accuracy in Chapter 4. Lastly, we propose to incorporate the second moment of returns to inform the network of the prevailing volatility environment, which will directly affect forecast uncertainty. We provide evidence of the benefits of the framework and each of the four improvements using the UCI benchmark datasets, and in cryptocurrencies and U.S. equities forecasts. In particular, using the UCI dataset and an identical network topology, we show that SMD delivers superior uncertainty quantification performance compared to the NIG prior. We believe forecast uncertainty will be a useful input into the portfolio optimisation process, such as for determining optimal bet size (high forecast uncertainty attracts a lower limit) or for scaling the risk model (Equation (1.4)).

6.2 Future research

As discussed in Chapter 1, financial markets represent one of the most challenging areas for the application of machine learning. In this section, we detail several potential advances of machine learning in future research.

Financial markets are endogenous. That is, one's own trading leaves a trail of footprints on asset prices (by incurring market impact and perturbing the share price). The same patterns may also be discovered by other investors which leave the same footprints. Financial markets are also impacted by exogenous shocks such as pandemics, wars and recessions. Thus, time-varying models have an important role in portfolio management. In Chapter 3, we have introduced OES and shown that, in theory, it offers superior predictive performance over a stationary model. We have not investigated realistic performance in a portfolio setting, after accounting for transaction costs and portfolio constraints (e.g., limits on how much the portfolio can bet on any single stock or industry). It is conceivable that actual realisable benefits of a time-varying model are concave with respect to the time-variability of the model due to higher trading. A model with moderate time-variability is likely better than a stationary

model after transaction costs. However, a highly time-varying model may not be better than a moderately time-varying model if transaction costs outstrip further improvements in tracking the time-varying DGP more closely. Thus, future research on this topic can investigate portfolio-level impacts of time-varying models and consider regularisations of OES, such as smoothing of regret in Hazan et al. (2017).

Efficient trading has been discussed in Section 1.6.4. Whilst not addressed in this thesis, this is a worthy topic within the broader domain of machine learning in portfolio management and is an essential component of autonomous trading systems. Stocks exhibit various intraday patterns, such as U-shaped volume distribution (higher at the beginning and end) throughout the trading day (Wood et al., 1985; Jain and Joh, 1988; McInish and Wood, 1992; Eaves and Williams, 2010), which is also associated with similar U-shaped intraday volatility (Lockwood and Linn, 1990; Eaves and Williams, 2010). Volume and volatility patterns form useful inputs to any model that aims to minimise market impact by predicting expected volume during the day. Future work can leverage recent advances in deep reinforcement learning[4] (François-Lavet et al., 2018), combining with new features to extend the work by Nevmyvaka et al. (2006) on using reinforcement learning for optimal trade execution. Another potential direction is to extend the work by Webber (2017), in incorporating concept drifts (Gama et al., 2014) into deep reinforcement learning to address time-varying financial markets (similar to our work in Chapter 3).

Extracting information from text has been discussed in Section 1.6.4. There is a large swathe of text information about companies, such as management's discussion of business performance in the annual report, may contain useful information for predicting future return. A significant portion of a financial analyst's job is to transform textual information about a company, such as the company's strategy and the competitive landscape, into future revenue and earnings expectations (Damodaran, 2006). For example, a biotechnology company developing a life-saving drug may see significant revenue in the future but is currently loss-making. Recently,

[4]Reinforcement learning is the task of learning a policy (i.e., sequence of actions) in an environment in order to maximise cumulative rewards (Bishop, 2006; Murphy, 2012; François-Lavet et al., 2018). This requires estimating future expected reward for each action. Deep reinforcement learning is to use neural networks to estimate future reward.

Araci (2019) used the BERT model (which was trained using the Wikipedia corpus, Devlin et al., 2019) and retrained the final layers using financial news articles to learn a finance-specific language model. I argue that the resultant model understands the grammar used in financial text, but is intrinsically devoid of understanding of the context. In the biotechnology firm example, an article may discuss the drug that the company is developing, but not the financial implications. Such second order effect is inferred from the context. Advances in this domain may combine natural language progressing and concept learning (Mitchell, 1997) in the context of financial markets.

In Chapter 4, we show that it is possible to learn predictive patterns directly from the share price time-series. Potential improvements to both the application of financial time-series forecasting and the method of learning in noisy environments are discussed in Section 4.4. A natural extension of this work is to combine cross-sectional forecasting and time-series forecasting, where the neural network is provided with time-series of all features relating to the company, such as stock prices, company financials and social media sentiment. Learning weak signals from such a large and diverse feature set will pose a significant challenge. However, I am convinced that if the financial industry were to advance towards highly tailored, stock-specific models, the advances will be reminiscent of the model described above. Such model can leverage methods of supervised autoencoding as described in Chapter 4, where the autoencoder performs dimensionality reduction which may assists with processing from a large feature set.

In Chapter 5, we have proposed a framework for quantifying uncertainty in financial time-series predictions. We suggest that forecast uncertainty can be used to determine bet sizes and serves as an input into the portfolio construction process. The quantified uncertainty is a scalar value that is specific to the stock. However, uncertainty may be correlated between stocks. Thus, a natural extension is to produce variance-covariance-style uncertainty that captures uncertainty covariance between a cohort of stocks. The uncertainty covariance matrix can then substitute or supplement the conventional variance-covariance matrix of asset returns used in mean-variance optimisation, as the latter neglects parameter uncertainty in return forecasts. Variance-covariance matrices are subject to the *curse of dimensionality*. For

example, consider the Russell 3000 index used in Chapter 4. Estimating a variance-covariance matrix for this universe involves estimating $3000 \times 3000 = 9\,\text{million}$ values. As discussed in Section 2.5.2, one of the advantages of CNN over fully connected neural network in image recognition problems is *parameter sharing*, which greatly reduces the number of parameters required by representing common patterns with a small number of parameters organised in a kernel. This property of convolution layers may offer a viable avenue to solve the curse of dimensionality problem in uncertainty covariance estimation.

The three advances introduced in this thesis relates to: online learning in a cross-sectional prediction context (Chapter 3), noise-robust learning in a time-series prediction context (Chapter 4), and forecast uncertainty quantification (Chapter 5). All three topics play important roles in quantitative investing. We argue that the three advances can be combined into a unified framework to simultaneously forecast returns, provide predictive uncertainty, and adapt to changes in the DGP of financial markets. We propose the following neural network architecture which can be evaluated in future work. The network is comprised of eight subnetworks:

- Subnetwork 1: Fully connected layers to process cross-sectional firm features (as per Chapter 3).
- Subnetwork 2: An encoder of LSTM layers to process daily stock returns (as per Chapter 5).
- Subnetwork 3: A decoder with fully connected layers which reconstruct all firm features using the latent representation outputted by Subnetwork 1.
- Subnetwork 4: A decoder of LSTM layers which reconstruct the daily return sequence using the latent representation outputted by Subnetwork 2.
- Subnetwork 5: Fully connected layers which combine the output of Subnetwork 1 and 2.
- Subnetwork 6-8: Outputs parameters γ, σ^2 and α of the SMD to simultaneously estimate both returns and predictive uncertainty.

The loss function is the NLL of the marginal t-distribution of SMD, plus reconstruction error of both firm features (output of Subnetwork 3) and time-series of returns (output of Subnetwork 4). To adapt to time-varying DGP, the network can be trained using OES. However, we envisage three potential challenges with this approach. First, as the network is quite large, one may encounter difficulties in training the entire network simultaneously. To solve this, one may employ *transfer learning* in training the two autoencoders for firm features (Subnetwork 1 and 3) and return series (Subnetwork 2 and 4). Transfer learning has been successfully applied in natural language processing, where a language model (typically an encoder-decoder) is trained on a large corpus of text to predict the next word, given the preceding words. The pre-trained language model is then fine tuned on downstream tasks such as sentiment analysis and question-answering (Ruder et al., 2019; Han et al., 2021). In a similar vein, we can first pre-train each encoder-decoder pair (Subnetwork 1 and 3, and Subnetwork 2 and 4) on encoding and decoding firm features and return series, respectively. The pre-trained encoder-decoder pairs can then be used in the final amalgamated network to perform return prediction. Second, the OES algorithm involves training on a the $t-2$ cross-section and validating on the $t-1$ cross-section. For such a large network, one may find that one cross-section contains insufficient data to train the network. To solve this, one may expand the number of periods used to train the network. However, we caution that expanding the look back window will lead to the algorithm to fit the average DGP in the look back window. Thus, losing its ability to closing track the time-varying DGP. Third, the scales of the three components of the loss function are different. Thus, care must be taken during hyperparameter search to determine the optimal weight given reconstructing firm features and return series. The proposed network architecture outputs both the return forecast and predictive uncertainty, which can then be used in downstream portfolio optimisation tasks.

Finally, in this thesis, we have proposed three advances that address various subtopics of applying deep learning to portfolio management, with much more still to be done. Deep learning has contributed to the advances of numerous fields of science. One particular advancement is DeepMind's *AlphaFold*[5], a machine learning system that can predict the

[5]DeepMind is a subsidiary of Alphabet Inc. that focuses on machine learning research. https://www.deepmind.com/research/highlighted-research/alphafold.

structure of over 200 million proteins and promises to speed up drug development. As a trained bioinformatician, I find this development exciting for the field of medical research and sobering for the finance industry. The main impediments to leaps in applying deep learning to financial markets are well discussed in Section 1.5. We, as finance practitioners, can dream that one day deep learning will shine some light on this dark corner of social science.

Bibliography

Balaji Lakshminarayanan, AlexanderPritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In Guyon et al. (2017), pages 6405–6416. ISBN 9781510860964.

Alexander Amini, Wilko Schwarting, Ava Soleimany, and Daniela Rus. Deep evidential regression. In Larochelle et al. (2020), pages 14927–14937.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105, Lake Tahoe, NV, USA, 2012. Curran Associates, Inc.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Bengio and LeCun (2015). URL http://arxiv.org/abs/1409.1556.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In Grauman et al. (2015), pages 1–9. doi: 10.1109/CVPR.2015.7298594.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In Grauman et al. (2015), pages 815–823. doi: 10.1109/CVPR.2015.7298682.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Lourdes Agapito, Tamara Berg, Jana Kosecka, and Lihi Zelnik-Manor, editors, *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR 2016, pages 770–778, Las Vegas, NV, USA, 2016. IEEE.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech*

and Signal Processing, ICASSP 2013, pages 6645–6649. IEEE, 2013. doi: 10.1109/ICASSP.2013.6638947.

Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, ICML'13. JMLR.org, 2013.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML'08, pages 160–167. ACM, 2008. ISBN 9781605582054. doi: 10.1145/1390156.1390177.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In Ghahramani et al. (2014), pages 3104–3112.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. ISSN 1476-4687.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. In *arXiv*, 2016.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alexander Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *arXiv*, 2016. URL https://arxiv.org/abs/1609.03499.

Eunhee Kang, Junhong Min, and Jong Chul Ye. Wavenet: a deep convolutional neural network using directional wavelets for low-dose x-ray ct reconstruction. *Medical Physics*, 44:360–375, 10 2017. doi: 10.1002/mp.12344.

Othmane Mounjid and Charles-Albert Lehalle. Improving reinforcement learning algorithms: towards optimal learning rate policies. In *arXiv*, 2021.

Jeremy D. Turiel and Tomaso Aste. Peer-to-peer loan acceptance and default prediction with artificial intelligence. *Royal Society Open Science*, 7(6):191649, 2020. doi: 10.1098/rsos.191649. URL https://royalsocietypublishing.org/doi/abs/10.1098/rsos.191649.

Richard Grinold and Ronald Kahn. *Active Portfolio Management: A Quantitative Approach for Producing Superior Returns and Controlling Risk*. McGraw-Hill Education, 1999.

Nga Pham. The australian superannuation system. Technical report, Monash University, Victoria, Australia, 2019. URL https://www.monash.edu/__data/assets/pdf_file/0016/2010553/The-Australian-superannuation-system_v3.pdf.

Shihao Gu, Bryan Kelly, and Dacheng Xiu. Empirical asset pricing via machine learning. *The Review of Financial Studies*, 33(5):2223–2273, 02 2020. ISSN 0893-9454. doi: 10.1093/rfs/hhaa009.

Pamela Peterson Drake and Frank J. Fabozzi. *Financial Instruments, Markets, and Intermediaries*, chapter 2, pages 13–35. John Wiley & Sons, Inc., 2010.

Frank J. Fabozzi, Frank J. Jones, Robert R. Johnson, and Pamela P. Drake. *Fundamentals of Common Stock*, chapter 8, pages 207–227. Volume 1 of Fabozzi and Markowitz (2011), 2011a. ISBN 9781118267028.

Yakov Amihud. Illiquidity and stock returns: cross-section and time-series effects. *Journal of Financial Markets*, 5(1):31–56, 2002. ISSN 1386-4181.

Aswath Damodaran. Lecture notes in corporate finance, Feb 2022.

Eugene F. Fama. Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*, 25(2):383–417, 1970.

Harry Markowitz. Portfolio selection. *Journal of Finance*, 7(1):77–91, 1952. ISSN 00221082, 15406261.

William F. Sharpe. Capital asset prices: A theory of market equilibrium under conditions of risk. *Journal of Finance*, 19(3):425–442, 1964. ISSN 00221082, 15406261.

Michael C. Jensen. The performance of mutual funds in the period 1945-1964. *Journal of Finance*, 23(2):389–416, 1968. ISSN 00221082, 15406261. URL http://www.jstor.org/stable/2325404.

Robert Oerter. *The Theory of Almost Everything: The Standard Model, the Unsung Triumph of Modern Physics*. Plume, 1 edition, 2006.

The Nobel Foundation. The sveriges riksbank prize in economic sciences in memory of alfred nobel 1990, Oct 1990. URL https://www.nobelprize.org/prizes/economic-sciences/1990/summary/. 2022-09-01.

Rolf W. Banz. The relationship between return and market value of common stocks. *Journal of Financial Economics*, 9(1):3–18, 1981.

Dennis Stattman. Book values and stock returns. In *The Chicago MBA: A Journal of Selected Papers*, volume 4, pages 25–45, 1980.

Barr Rosenberg, Kenneth Reid, and Ronald Lanstein. Persuasive evidence of market inefficiency. *Journal of Portfolio Management*, 11(3):9–16, Spring 1985. URL http://ezproxy.lib.uts.edu.au/login?url=https://search-proquest-com.ezproxy.lib.uts.edu.au/docview/195568751?accountid=17095. Name - New York Stock Exchange; Copyright - Copyright Euromoney Institutional Investor PLC Spring 1985; Last updated - 2015-05-25.

Campbell R. Harvey, Yan Liu, and Heqing Zhu. ... and the cross-section of expected returns. *The Review of Financial Studies*, 29(1):5–68, 2016.

Eugene F. Fama and James D. MacBeth. Risk, return, and equilibrium: Empirical tests. *Journal of Political Economy*, 81(3):607–637, 1973. doi: 10.1086/260061.

Andrew Alford, Robert Jones, and Terence Lim. *Quantitative Equity Portfolio Management*, chapter 11, pages 287–306. Volume 1 of Fabozzi and Markowitz (2011), 2011. ISBN 9781118267028.

Guofu Zhou and Frank J. Fabozzi. *Factor Models*, chapter 5, pages 103–124. Volume 1 of Fabozzi and Markowitz (2011), 2011. ISBN 9781118267028.

Jeffrey Marc Wooldridge. *Introductory Econometrics: A Modern Approach*. South-Western, 4th edition, 2008. ISBN 9780324581621.

Rama Cont. Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative Finance*, 1:223–236, 2001.

Sara Salinas and Michelle Castillo. Facebook just suffered its worst day ever. *CNBC*, 2018. URL https://www.cnbc.com/2018/07/26/facebook-is-on-pace-for-its-worst-day-ever.html.

Mohamed A. El-Erian. Facebook just suffered its worst day ever. *Bloomberg*, 2021. URL https://www.bloomberg.com/opinion/articles/2021-01-30/gamestop-gme-short-squeeze-who-will-surrender-first.

Anders Johansen and Didier Sornette. Endogenous versus exogenous crashes in financial markets. In *SSRN*, 2002. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=344980.

Yahoo! Finance. Gamestop corp. (gme) stock historical prices & data – yahoo finance, Mar 2022a. URL https://au.finance.yahoo.com/quote/GME/history?p=GME. 2022-03-07.

Ruiqiang Song, Min Shu, and Wei Zhu. The 2020 global stock market crash: Endogenous or exogenous? *Physica A: Statistical Mechanics and its Applications*, 585:126425, 2022. ISSN 0378-4371. doi: https://doi.org/10.1016/j.physa.2021.126425.

Yahoo! Finance. Devon energy corporation (dvn) stock historical prices & data – yahoo finance, Mar 2022b. URL https://au.finance.yahoo.com/quote/DVN/history?p=DVN. 2022-03-07.

Guy P. Nason. Stationary and non-stationary time-series. *Statistics in Volcanology*, 1:129–142, 2006.

João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):44:1–44:37, March 2014. ISSN 0360-0300. doi: 10.1145/2523813.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Jeffrey C. Schlimmer and Richard H. Granger. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, 1986. doi: 10.1007/BF00116895.

Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996. doi: 10.1007/BF00116900.

M. Hashem Pesaran and Allan Timmermann. Predictability of stock returns: Robustness and economic significance. *Journal of Finance*, 50:1201–1228, 1995.

Peter Bossaerts and Pierre Hillion. Implementing statistical criteria to select return forecasting models: What do we learn? *Review of Financial Studies*, 12(2):405–428, 06 1999. ISSN 0893-9454. doi: 10.1093/rfs/12.2.405. URL https://doi.org/10.1093/rfs/12.2.405.

Timotheos Angelidis, Athanasios Sakkas, and Nikolaos Tessaromatis. Stock market dispersion, the business cycle and expected factor returns. *Journal of Banking & Finance*, 59: 265–279, 2015. ISSN 0378-4266. doi: https://doi.org/10.1016/j.jbankfin.2015.04.025.

R. David McLean and Jeffrey Pontiff. Does academic research destroy stock return predictability? *Journal of Finance*, 71(1):5–32, 2016. doi: 10.1111/jofi.12365.

Xi Dong, Qi Liu, Lei Lu, Bo Sun, and Hongjun Yan. Anomaly discovery and arbitrage trading. In *SSRN working paper*, 2020. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2431498.

Narasimhan Jegadeesh and Sheridan Titman. Returns to buying winners and selling losers: Implications for stock market efficiency. *Journal of Finance*, 48(1):65–91, 1993.

Clifford S. Asness, Tobias J. Moskowitz, and Lasse Heje Pedersen. Value and momentum everywhere. *Journal of Finance*, 68(3):929–985, 2013. ISSN 00221082, 15406261.

Melody Y. Huang, Randall R. Rojas, and Patrick D. Convery. Forecasting stock market movements using google trend searches. *Empirical Economics*, 59:2821–2839, 2020. ISSN 1435-8921. doi: https://doi.org/10.1007/s00181-019-01725-1.

Lily Fang and Joel Peress. Media coverage and the cross-section of stock returns. *Journal of Finance*, 64(5):2023–2052, 2009.

Alois Weigand. Machine learning in empirical asset pricing. *Financial Markets and Portfolio Management*, 33:93–104, 2019.

Marcial Messmer. Deep learning and the cross-section of expected returns. In *SSRN*, 2017. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3081555.

Masaya Abe and Hideki Nakayama. Deep learning for forecasting stock returns in the cross-section. In *arXiv*, 2018. URL https://arxiv.org/abs/1801.01777.

Baruch Lev and Anup Srivastava. Explaining the recent failure of value investing. In *SSRN*, 2019. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3442539.

Eugene F. Fama and Kenneth R. French. The cross-section of expected stock returns. *Journal of Finance*, 47(2):427–465, 1992. ISSN 00221082, 15406261. URL http://www.jstor.org/stable/2329112.

Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1996.

Jonathan Brogaard and Abalfazl Zareei. Machine learning and the stock market. In *SSRN*, 2022. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3233119.

Shihao Gu, Bryan Kelly, and Dacheng Xiu. Autoencoder asset pricing models. *Journal of Econometrics*, 222(1):429–450, 2021. ISSN 0304-4076. Annals Issue:Financial Econometrics in the Age of the Digital Economy.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2 edition, 2020.

Luyang Chen, Markus Pelger, and Jason Zhu. Deep learning in asset pricing. In *arXiv*, 2021. URL https://arxiv.org/abs/1904.00745.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Ghahramani et al. (2014), pages 2672–2680.

Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Anomaly discovery and arbitrage trading. In *arXiv*, 2021. URL https://arxiv.org/abs/2106.04560.

Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In Guyon et al. (2017), pages 6232–6240. ISBN 9781510860964.

John F. Kolen and Stefan C. Kremer. *Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies*, pages 237–243. Wiley-IEEE Press, 2001.

David Rolnick, Andreas Veit, Serge Belongie, and Nir Shavit. Deep learning is robust to massive label noise. In *arXiv*, 2018. URL https://arxiv.org/abs/1705.10694.

Milad Moradi, Kathrin Blagec, and Matthias Samwald. Deep learning models are not robust against noise in clinical text. In *arXiv*, 2021. URL https://arxiv.org/abs/2108.12242.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, January 2014. ISSN 1532-4435.

Giorgio Patrini, Alessandro Rozza, Aditya Krishna Menon, Richard Nock, and Lizhen Qu. Making deep neural networks robust to label noise: A loss correction approach. In Yanxi Liu, James M. Rehg, Camillo J. Taylor, and Ying Wu, editors, *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR 2017, pages 2233–2241, Honolulu, HI, USA, 2017. IEEE.

Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Proceedings of the 13th European Conference on Computer Vision*, ECCV 2014, pages 94–108. Springer International Publishing, 2014. ISBN 978-3-319-10599-4.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(76):2493–2537, 2011.

Omer Sezer, Ugur Gudelek, and Murat Ozbayoglu. Financial time series forecasting with deep learning : A systematic literature review: 2005—2019. *Applied Soft Computing*, 90: 106–181, 02 2020. doi: 10.1016/j.asoc.2020.106181.

Kai Chen, Yi Zhou, and Fangyan Dai. A lstm-based method for stock returns prediction: A case study of china stock market. In Maria Prandini, editor, *Proceedings of the 2015 IEEE International Conference on Big Data*, Big Data '15', pages 2823–2824, Santa Clara, CA, USA, 2015. IEEE.

Rosa Altilio, Giorgio Andreasi, and Massimo Panella. *A Classification Approach to Modeling Financial Time Series*, pages 97–106. Springer International Publishing, Cham, Switzerland, 2019.

Yves Meyer. *Wavelets and Operators*, volume 1 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1993.

Hongju Yan and Hongbing Ouyang. Financial time series prediction based on deep learning. *Wireless personal communications*, 102(2):683–700, 2017.

Zhixi Li and Vincent Tam. Combining the real-time wavelet denoising and long-short-term-memory neural network for predicting stock indexes. In *2017 IEEE Symposium Series on Computational Intelligence*, SSCI, pages 1–8, Honolulu, HI, USA, 2017. IEEE.

John L. Kelly. A new interpretation of information rate. *Bell System Technical Journal*, 35(4): 917–926, 1956.

Tim Byrnes and Tristan Barnett. Generalized framework for applying the kelly criterion to stock markets. *International Journal of Theoretical and Applied Finance*, 21(05):1–13, 2018. doi: 10.1142/S0219024918500334.

Fischer Black and Robert Litterman. Global portfolio optimization. *Financial Analysts Journal*, 48(5):28–43, Sep 1992.

John Mitros and Brian Mac Namee. On the validity of bayesian neural networks for uncertainty estimation. In *arXiv*, 2019. URL https://arxiv.org/abs/1912.01530.

Matias Quiroz, Robert Kohn, Mattias Villani, and Minh-Ngoc Tran. Speeding up MCMC by efficient data subsampling. *Journal of the American Statistical Association*, 114(526): 831–843, 2019. doi: 10.1080/01621459.2018.1448827.

Gabriele Ranco, Darko Aleksovski, Guido Caldarelli, Miha Grčar, and Igor Mozetič. The effects of twitter sentiment on stock price returns. *PloS one*, 10:1–21, 09 2015. doi: 10.1371/journal.pone.0138441.

Wataru Souma, Irena Vodenska, and Hideaki Aoyama. Enhanced news sentiment analysis using deep learning methods. *Journal of Computational Social Science*, 2:33–46, 01 2019. doi: 10.1007/s42001-019-00035-x.

Ran El-Yaniv, Amos Fiat, Richard M. Karp, and G. Turpin. Optimal search and one-way trading online algorithms. *Algorithmica*, 30(1):101–139, 05 2001. doi: 10.1007/s00453-001-0003-0.

Lili Dworkin, Michael Kearns, and Yuriy Nevmyvaka. Pursuit-evasion without regret, with an application to trading. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the*

31st International Conference on Machine Learning, volume 37, pages 1521–1529, Bejing, China, 2014. JMLR.org.

Yuriy Nevmyvaka, Yi Feng, and Michael Kearns. Reinforcement learning for optimized trade execution. In William Cohen and Andrew Moore, editors, *Proceedings of the 23rd International Conference on Machine Learning*, volume 2006 of *ACM International Conference Proceeding Series*, pages 673–680, Pittsburgh, PA, USA, 06 2006. ACM.

Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, 11(3–4):1–156, 2018. doi: 10.1561/2200000071.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359—-366, jul 1989. ISSN 0893-6080.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989. ISSN 1435-568X. doi: 10.1007/BF02551274. URL https://doi.org/10.1007/BF02551274.

Alexander Bain. *Mind and Body: The Theories of their Relation*. D. Appleton and Company, New York, NY, USA, 1873.

William James. *The Principles of Psychology*. H. Holt and Company, New York, NY, USA, 1890.

Frank Rosenblatt. The perceptron: A probalistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

Frank Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington D.C., USA, 1961.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, pages 318–362. MIT Press, Cambridge, MA, USA, 1986a. ISBN 026268053X.

Eran Malach, Gilad Yehudai, Shai Shalev-Shwartz, and Ohad Shamir. The connection between approximation, depth separation and learnability in neural networks. In Mikhail Belkin and Samory Kpotufe, editors, *Conference on Learning Theory*, COLT 2021, pages 3265–3295, Boulder, CO, USA, 2021. PMLR.

Shan Sung Liew, Mohamed Khalil-Hani, and Rabia Bakhteri. Bounded activation functions for enhanced training stability of deep neural networks on visual pattern recognition problems. *Neurocomputing*, 216:718–734, 2016. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2016.08.037.

Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, ICCV 2009, pages 2146–2153, Kyoto, Japan, 2009. IEEE.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 27th International Conference on Machine Learning*, ICML'10, pages 807–814. JMLR.org, 2010.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. In *6th International Conference on Learning Representations*, ICLR 2018, Vancouver, BC, Canada, 2018. OpenReview.net.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations*, ICLR 2016, 2016.

Solomon Kullback and Richard Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951. doi: 10.1214/aoms/1177729694.

Kevin Wainwright and Alpha C. Chiang. *Fundamental Methods of Mathematical Economics*. McGraw Hill, fourth edition, 2005.

Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping from saddle points - online stochastic gradient for tensor decomposition. In Peter Grünwald, Elad Hazan, and Satyen Kale, editors, *Proceedings of The 28th Conference on Learning Theory*, COLT 2015, pages 797–842, Paris, France, 2015. JMLR.org.

Chi Jin, Rong Ge, Praneeth Netrapalli, Sham M. Kakade, and Michael I. Jordan. How to escape saddle points efficiently. In Precup and Teh (2017), pages 1724–1732.

Bobby Kleinberg, Yuanzhi Li, and Yang Yuan. An alternative view: When does SGD escape local minima? In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80, pages 2698–2707, Stockholm, Sweden, 2018. PMLR.

Nelson Morgan and Hervé A. Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 630–637. Morgan-Kaufmann, 1990.

Russell Deryl Reed. Pruning algorithms-a survey. *Transactions on Neural Networks*, 4(5): 740–747, 1993. ISSN 1045-9227. doi: 10.1109/72.248452.

Lutz Prechelt. *Early Stopping - But When?* Springer-Verlag, London, UK, 1998. ISBN 3-540-65311-2.

Maren Mahsereci, Lukas Balles, Christoph Lassner, and Philipp Hennig. Early stopping without a validation set. *CoRR*, abs/1703.09580, 2017.

Jonas Sjöberg and Lennart Ljung. Overtraining, regularization and searching for a minimum, with application to neural networks. *International Journal of Control*, 62(6):1391–1407, 1995. doi: 10.1080/00207179508921605.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011. ISSN 1532-4435.

Geoffrey Hinton and Tijmen Tieleman. Lecture 6.5 - rmsprop, coursera: Neural networks for machine learning, 2012. URL http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Bengio and LeCun (2015).

Timothy Dozat. Incorporating nesterov momentum into adam. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations*, ICLR 2016, 2016.

Yurii Nesterov. A method of solving a convex programming problem with convergence rate $o(1/k^2)$. *Soviet Mathematics Doklady*, 27:372–376, 1983.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Dasgupta and McAllester (2013), pages 1139–1147.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the 13th*

International Conference on Artificial Intelligence and Statistics, volume 9 of *AISTATS 2010*, pages 249–256. PMLR, 2010.

David M. Bradley. *Learning in Modular Systems*. PhD thesis, The Robotics Institute, Carnegie Mellon University, 2010.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision*, ICCV, pages 1026–1034. IEEE, 2015. doi: 10.1109/ICCV.2015.123.

Siddharth Krishna Kumar. On weight initialization in deep neural networks. In *arXiv*, 2017. URL https://arxiv.org/abs/1704.08863.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Bach and Blei (2015), pages 448–456.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In Bengio et al. (2018).

Jonas Moritz Kohler, Hadi Daneshmand, Aurélien Lucchi, Thomas Hofmann, Ming Zhou, and Klaus Neymeyr. Exponential convergence rates for batch normalization: The power of length-direction decoupling in non-convex optimization. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, volume 89 of *AISTATS 2019*, pages 806–815. PMLR, 2019.

Tim Salimans and Durk P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 901–909, Red Hook, NY, USA, 2016. Curran Associates, Inc.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, chapter 8, pages 318—-362. MIT Press, Cambridge, MA, USA, 1986b. ISBN 026268053X.

Chung-Cheng Chiu, Tara N. Sainath, Yonghui Wu, Rohit Prabhavalkar, Patrick Nguyen, Zhifeng Chen, Anjuli Kannan, Ron J. Weiss, Kanishka Rao, Ekaterina Gonina, Navdeep Jaitly, Bo Li, Jan Chorowski, and Michiel Bacchiani. State-of-the-art speech recognition

with sequence-to-sequence models. In Dan Schonfeld, Pascale Fung, and Nam Ik Cho, editors, *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4774–4778, Calgary, AB, Canada, 2018. IEEE. doi: 10.1109/ICASSP.2018.8462105.

Yunus Emre Cebeci. A recurrent neural network model for weather forecasting. In *2019 4th International Conference on Computer Science and Engineering*, UBMK, pages 591–595. IEEE, 2019. doi: 10.1109/UBMK.2019.8907196.

Natasha Jaques, Shixiang Gu, Richard E. Turner, and Douglas Eck. Generating music by fine-tuning recurrent neural networks with reinforcement learning. In *5th International Conference on Learning Representations*, ICLR 2017. OpenReview.net, 2017.

Xiumin Li, Lin Yang, Fangzheng Xue, and Hongjun Zhou. Time series prediction of stock price using deep belief networks with intrinsic plasticity. In *Proceedings of the 29th Chinese Control And Decision Conference*, CCDC 2017, pages 1237–1242. IEEE, 2017. doi: 10.1109/CCDC.2017.7978707.

Arzoo Katiyar and Claire Cardie. Nested named entity recognition revisited. In Marilyn A. Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL-HLT 2018, pages 861–871, New Orleans, LA, USA, 2018. Association for Computational Linguistics. doi: 10.18653/v1/n18-1079.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In Dasgupta and McAllester (2013), pages 1310–1318.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. In *arXiv*, 2018. URL https://arxiv.org/abs/1803.01271.

Yoshua Bengio, Patrice Simard, , and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9 (8):1735—-1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735.

Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition.

IEEE Transactions on Pattern Analysis and Machine Intelligence, 31(5):855–868, 2009. doi: 10.1109/TPAMI.2008.137.

Nal Kalchbrenner, Lasse Espeholt, Aäron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. In *arXiv*, 2016. URL https://arxiv.org/abs/1609.03499.

Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In Grauman et al. (2015), pages 648–656.

Yann LeCun. *Modeles connexionnistes de l'apprentissage*. Université de Paris VI, June 1987. PhD thesis.

H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4–5):291–294, September 1988. ISSN 0340-1200. doi: 10.1007/BF00332918.

Geoffrey E. Hinton and Richard S. Zemel. Autoencoders, minimum description length and helmholtz free energy. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, pages 3–10, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

Pierre Baldi and K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2(1):53–58, January 1989. ISSN 0893-6080. doi: 10.1016/0893-6080(89)90014-2.

Lei Le, Andrew Patterson, and Martha White. Supervised autoencoders: Improving generalization performance with unsupervised regularizers. In Bengio et al. (2018), pages 107–117.

Brecht Evens, Puya Latafat, Andreas Themelis, Johan Suykens, and Panagiotis Patrinos. Neural network training as an optimal control problem: An augmented lagrangian approach. In Maria Prandini, editor, *Proceedings of the 60th IEEE Conference on Decision and Control*, CDC, pages 5136–5143, Fairmont, TX, USA, 2021. IEEE.

Shai Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2012. ISSN 1935-8237. doi: 10.1561/2200000018.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144, New York, NY, USA, 2016. ACM.

Keith Ambachtsheer. Profit potential in an almost efficient market. *Journal of Portfolio Management*, 1(1):84, FALL 1974.

Frank J. Fabozzi, James L. Grant, and Raman Vardharaj. *Common Stock Portfolio Management Strategies*, chapter 9, pages 229–270. Volume 1 of Fabozzi and Markowitz (2011), 2011b. ISBN 9781118267028.

Sergul Aydore, Tianhao Zhu, and Dean P. Foster. Dynamic local regret for non-convex online forecasting. In Wallach et al. (2019), pages 7980–7989.

Barbara Rossi and Atsushi Inoue. Out-of-sample forecast tests robust to the choice of window size. *Journal of Business & Economic Statistics*, 30(3):432–453, 2012.

Christoph Bergmeir, Rob Hyndman, and Bonsoo Koo. A note on the validity of cross-validation for evaluating autoregressive time series prediction. *Computational Statistics & Data Analysis*, 120:70–83, 4 2018. doi: 10.1016/j.csda.2017.11.003.

Yujie Liu, Hongbin Dong, Xingmei Wang, and Shuang Han. Time series prediction based on temporal convolutional network. In Simon Xu, Yongbin Wang, Mingyong Shi, Wenqiang Shang, Jiefeng Liu, and Kailong Zhang, editors, *2019 IEEE/ACIS 18th International Conference on Computer and Information Science*, ICIS 2019, pages 300–305, Beijing, China, 2019. IEEE.

Subhrajit Samanta, Mahardhika Pratama, Suresh Sundaram, and Narasimalu Srikanth. Learning elastic memory online for fast time series forecasting. *Neurocomputing*, 390:315–326, 2020. ISSN 0925-2312.

Nicolò Cesa-Bianchi, Pierre Gaillard, Gábor Lugosi, and Gilles Stoltz. A new look at shifting regret. In *arXiv*, 2012.

Thomas M. Cover. Universal portfolios. *Mathematical Finance*, 1(1):1–29, 1991. doi: 10.1111/j.1467-9965.1991.tb00002.x.

Elad Hazan, Karan Singh, and Cyril Zhang. Efficient regret minimization in non-convex games. In Precup and Teh (2017), pages 1433–1441.

Jeremiah Green, John R. M. Hand, and X. Frank Zhang. The characteristics that provide independent information about average U.S. monthly stock returns. *The Review of Financial Studies*, 30(12):4389–4436, 03 2017. ISSN 0893-9454. doi: 10.1093/rfs/hhx019.

Ivo Welch and Amit Goyal. A comprehensive look at the empirical performance of equity premium prediction. *The Review of Financial Studies*, 21(4):1455–1508, 03 2008. ISSN 0893-9454. doi: 10.1093/rfs/hhm014.

Joel L. Horowitz, Tim Loughran, and N.E. Savin. The disappearing size effect. *Research in Economics*, 54(1):83–100, 2000. ISSN 1090-9443.

U.S. Securities and Exchange Commission. Microcap stock: A guide for investors. https://www.sec.gov/reportspubs/investor-publications/investorpubsmicrocapstockhtm.html, Sep 2013. Accessed: 2021-01-03.

Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. *Efficient BackProp*, pages 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

Kent Daniel and Tobias J. Moskowitz. Momentum crashes. *Journal of Financial Economics*, 122(2):221–247, 2016. ISSN 0304-405X.

Spyros Makridakis and Michèle Hibon. The M3-competition: results, conclusions and implications. *International Journal of Forecasting*, 16(4):451–476, 2000. ISSN 0169-2070.

Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. Statistical and machine learning forecasting methods: Concerns and ways forward. *PLOS ONE*, 13(3): 1–26, 03 2018. doi: 10.1371/journal.pone.0194889.

Rob J. Hyndman. A brief history of forecasting competitions. *International Journal of Forecasting*, 36(1):7–14, 2020. ISSN 0169-2070. M4 Competition.

George E. P. Box, Gwilym M. Jenkins, and Gregory C. Reinsel. *Time Series Analysis: Forecasting and Control*. Prentice Hall, Englewood Cliffs, N.J., USA, 3 edition, 1994.

Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-BEATS: neural basis expansion analysis for interpretable time series forecasting. In *8th International Conference on Learning Representations*, ICLR 2020, Addis Ababa, Ethiopia, 2020. OpenReview.net.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Guyon et al. (2017),

pages 6000–6010.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL 2019, pages 4171–4186, Online, 2019. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Larochelle et al. (2020), pages 1877–1901.

Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *arXiv*, 2022. URL https://arxiv.org/abs/2205.13504.

J.B. Heaton, Nick Polson, and Jan Hendrik Witte. Deep learning in finance. In *arXiv*, 2016. URL https://arxiv.org/abs/1602.06561.

Jerzy Korczak and Marcin Hemes. Deep learning for financial time series forecasting in a-trader system. In *2017 Federated Conference on Computer Science and Information Systems*, FedCSIS, pages 905–912, Prague, Czech Republic, 2017. IEEE.

Eugene F. Fama. Random walks in stock market prices. *Financial Analysts Journal*, 21(5): 55–59, 1965. ISSN 0015198X.

Amado Peiró. The distribution of stock returns: international evidence. *Applied Financial Economics*, 4(6):431–439, 1994. doi: 10.1080/758518675.

Michael Isichenko. *Quantitative Portfolio Management: The Art and Science of Statistical Arbitrage*. Wiley, 2021.

Jining Yan, Lin Mu, Lizhe Wang, Rajiv Ranjan, and Albert Y. Zomaya. Temporal convolutional networks for the advance prediction of enso. *Scientific Reports*, 10(1), 2020.

Rui Dai, Shenkun Xu, Qian Gu, Chenguang Ji, and Kaikui Liu. Hybrid spatio-temporal graph convolutional network: Improving traffic prediction with navigation data. In *Proceedings*

of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 3074–3082, Virtual Event, CA, USA, 2020. ACM.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. In *arXiv*, 2016. URL https://arxiv.org/abs/1609.08144.

Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on signal processing*, 45(11):2673–2681, 1997.

Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In Lluís Màrquez, Chris Callison-Burch, and Jian Su, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, EMNLP 2015, pages 1412–1421, Lisbon, Portugal, 2015. Association for Computational Linguistics.

S. C. Suddarth and Y. L. Kergosien. Rule-injection hints as a means of improving network performance and learning time. In *Proceedings of the EURASIP Workshop 1990 on Neural Networks*, pages 120–129, Berlin, Heidelberg, 1990. Springer-Verlag. ISBN 3540522557.

Baruch Epstein and Ron Meir. Generalization bounds for unsupervised and semi-supervised learning with autoencoders. In *arXiv*, 2019. URL https://arxiv.org/abs/1902.01449.

Patrick Thiam, Hans A. Kestler, and Friedhelm Schwenker. Multimodal deep denoising convolutional autoencoders for pain intensity classification based on physiological signals. In Maria De Marsico, Gabriella Sanniti di Baja, and Ana L. N. Fred, editors, *Proceedings of the 9th International Conference on Pattern Recognition Applications and Methods*, ICPRAM 2020, pages 289–296. SCITEPRESS, 2020.

Michel Barlaud and Frederic Guyard. A non-parametric supervised autoencoder for discriminative and generative modeling. In *HAL*, September 2020. URL https:

//hal.archives-ouvertes.fr/hal-02937643. working paper or preprint.

Shantipriya Parida, Esau Villatoro-Tello, Sajit Kumar, Mael Fabien, and Petr Motlicek. Detection of similar languages and dialects using deep supervised autoencoders. In Pushpak Bhattacharyya, Dipti Misra Sharma, and Rajeev Sangal, editors, *Proceedings of the 17th International Conference on Natural Language Processing*, ICON 2020. SCITEPRESS, 2020.

M. Hiransha, E. A. Gopalakrishnan, Vijay Krishna Menon, and K. P. Soman. Nse stock market prediction using deep-learning models. *Procedia Computer Science*, 132:1351–1362, 2018. ISSN 1877-0509. International Conference on Computational Intelligence and Data Science.

Rohitash Chandra and Shelvin Chand. Evaluation of co-evolutionary neural network architectures for time series prediction with mobile application in finance. *Applied Soft Computing*, 49:462–473, 2016. ISSN 1568-4946.

Bryan Lim, Stefan Zohren, and Stephen Roberts. Enhancing time-series momentum strategies using deep neural networks. *Journal of Financial Data Science*, 2019. ISSN 2405-9188. doi: 10.3905/jfds.2019.1.015.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In Grauman et al. (2015), pages 3431–3440.

Steven Y. K. Wong, Jennifer Chan, Lamiae Azizi, and Richard Y. D. Xu. Supervised temporal autoencoder for stock return time-series forecasting. In Wing Kwong Chan, Bill Claycomb, and Hiroki Takakura, editors, *Proceedings of the IEEE 45th Annual Computer Software and Applications Conference (COMPSAC'21)*, Madrid, Spain, 2021. IEEE.

George E. P. Box and Gwilym M. Jenkins. *Time series analysis: forecasting and control*. Holden-Day series in time series analysis and digital processing. Holden-Day, San Francisco, rev. ed. edition, 1976. ISBN 0816211043.

Helmut Lütkepohl and Fang Xu. The role of the log transformation in forecasting economic variables. *Empirical Economics*, 42(3):619–638, 2012.

Thomas Fischer and Christopher Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270 (2):654–669, 2018. ISSN 0377-2217.

Fischer Black and Robert B Litterman. Asset allocation: Combining investor views with market equilibrium. *Journal of Fixed Income*, 1(2):7–18, 1991.

Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseo Lee, Matthias Humt, Jianxiang Feng, Anna M. Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, M. Shahzad, Wen Yang, Richard Bamler, and Xiaoxiang Zhu. A survey of uncertainty in deep neural networks. In *arXiv*, 2021. URL https://arxiv.org/abs/2107.03342.

Cornelia Gruber, Patrick Oliver Schenk, Malte Schierholz, Frauke Kreuter, and Göran Kauermann. Sources of uncertainty in machine learning – a statisticians' view. In *arXiv*, 2023. URL https://arxiv.org/abs/2305.16703.

Nis Meinert, Jakob Gawlikowski, and Alexander Lavin. The unreasonable effectiveness of deep evidential regression. In *arXiv*, 2022. URL https://arxiv.org/abs/2205.10060.

T.J. Sullivan. *Introduction to Uncertainty Quantification*. Number 63 in Texts in Applied Mathematics. Springer International Publishing, Cham, Germany, 1st edition, 2015. ISBN 3-319-23395-5.

S. C. Hora. Aleatory and epistemic uncertainty in probability elicitation with an example from hazardous waste management: Treatment of aleatory and epistemic uncertainty. *Reliability engineering & system safety*, 54(2–3):217–223, 1996. ISSN 0951-8320.

David J. C. MacKay. A practical bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 1992. ISSN 0899-7667. doi: 10.1162/neco.1992.4.3.448.

Radford M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag, Berlin, Heidelberg, 1996. ISBN 0387947248.

Yarin Gal. *Uncertainty in Deep Learning*. University of Cambridge, 2016. PhD thesis.

Wilfred Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

Laurent Valentin Jospin, Hamid Laga, Farid Boussaid, Wray Buntine, and Mohammed Bennamoun. Hands-on bayesian neural networks — a tutorial for deep learning users. *IEEE computational intelligence magazine*, 17(2):29–48, 2022.

Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. Being bayesian, even just a bit, fixes overconfidence in relu networks. In *arXiv*, 2020. URL https://arxiv.org/abs/2002.10118.

Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D. Sculley, Sebastian Nowozin, Joshua V. Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In Wallach et al. (2019), pages 14003–14014.

José M. Bernardo and Adrian F. M. Smith. *Bayesian theory*. John Wiley & Sons Ltd., 2000.

Zhijian Liu, Alexander Amini, Sibo Zhu, Sertac Karaman, Song Han, and Daniela L. Rus. Efficient and robust lidar-based end-to-end navigation. In Yu Sun, editor, *2021 IEEE International Conference on Robotics and Automation*, ICRA, pages 13247–13254, Xi'an, China, 2021. IEEE.

Peide Cai, Hengli Wang, Huaiyang Huang, Yuxuan Liu, and Ming Liu. Vision-based autonomous car racing using deep imitative reinforcement learning. *IEEE Robotics and Automation Letters*, 6(4):7262–7269, 2021. doi: 10.1109/LRA.2021.3097345.

Sandeep Kumar Singh, Jaya Shradha Fowdur, Jakob Gawlikowski, and Daniel Medina. Leveraging graph and deep learning uncertainties to detect anomalous maritime trajectories. *IEEE Transactions on Intelligent Transportation Systems*, 23(12):23488–23502, 2022. doi: 10.1109/TITS.2022.3190834.

Ava P. Soleimany, Alexander Amini, Samuel Goldman, Daniela Rus, Sangeeta N. Bhatia, and Connor W. Coley. Evidential deep learning for guided molecular property prediction and discovery. *ACS central science*, 7(8):1356–1367, 2021. ISSN 2374-7943.

Hao Li and Jianan Liu. 3D high-quality magnetic resonance image restoration in clinics using deep learning. In *arXiv*, 2022. URL https://arxiv.org/abs/2111.14259.

Dongpin Oh and Bonggun Shin. Improving evidential deep learning via multi-task learning. In *Thirty-Sixth AAAI Conference on Artificial Intelligence*, pages 7895–7903. AAAI Press, 2022.

Bertrand Charpentier, Oliver Borchert, Daniel Zügner, Simon Geisler, and Stephan Günnemann. Natural posterior network: Deep bayesian predictive uncertainty for exponential family distributions. In *9th International Conference on Learning Representations (ICLR)*,

online, 2021. OpenReview.net.

Andrey Malinin, Sergey Chervontsev, Ivan Provilkov, and Mark Gales. Regression prior networks. In *arXiv*, 2020. URL https://arxiv.org/abs/2006.11590.

Tilmann Gneiting and Adrian E. Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

Viktor Bengs, Eyke Hüllermeier, and Willem Waegeman. On second-order scoring rules for epistemic uncertainty quantification. In *arXiv*, 2023. URL https://arxiv.org/abs/2301.12736.

Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307–327, 1986. ISSN 0304-4076. doi: https://doi.org/10.1016/0304-4076(86)90063-1.

Rachael Carroll and Colm Kearney. *GARCH Modeling of Stock Market Volatility*, pages 71–90. Chapman & Hall/CRC finance series. CRC Press, New York, NY, USA, 1st edition, 2009.

John Fry and Eng-Tuck Cheah. Negative bubbles and shocks in cryptocurrency markets. *International Review of Financial Analysis*, 47:343–352, 2016. ISSN 1057-5219.

Christian M. Hafner. Testing for Bubbles in Cryptocurrencies with Time-Varying Volatility. *Journal of Financial Econometrics*, 18(2):233–249, 10 2018. ISSN 1479-8409. doi: 10.1093/jjfinec/nby023.

Cathy Yi-Hsuan Chen and Christian M. Hafner. Sentiment-induced bubbles in the cryptocurrency market. *Journal of Risk and Financial Management*, 12(2):1–12, 2019. ISSN 1911-8074.

José Antonio Núñez, Mario I. Contreras-Valdez, and Carlos A. Franco-Ruiz. Statistical analysis of bitcoin during explosive behavior periods. *PLOS ONE*, 14(3):1–22, 03 2019. doi: 10.1371/journal.pone.0213919.

Alla Petukhina, Simon Trimborn, Wolfgang Karl Härdle, and Hermann Elendner. Investing with cryptocurrencies – evaluating their potential for portfolio allocation strategies. *Quantitative Finance*, 21(11):1825–1853, 2021.

José Miguel Hernández-Lobato and Ryan P. Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In Bach and Blei (2015), pages 1861–1869.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, volume 48, pages 1050–1059. JMLR.org, 2016.

Michael Jordan. The exponential family: Conjugate priors, 2009.

David F. Andrews and Colin L. Mallows. Scale mixtures of normal distributions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 36(1):99–102, 1974.

S.T. Boris Choy and Jennifer S.K. Chan. Scale mixtures distributions in statistical modelling. *Australian & New Zealand Journal of Statistics*, 50(2):135–146, 2008. ISSN 1369-1473.

Christian Brownlees, Robert Engle, and Bryan Kelly. A practical guide to volatility forecasting through calm and storm. *Journal of Risk*, 14(2):3–22, 2011.

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996. ISSN 1573-0565.

Adrian E. Raftery, Miroslav Kárný, and Pavel Ettler. Online prediction under model uncertainty via dynamic model averaging: Application to a cold rolling mill. *Technometrics*, 52 (1):52–66, 2010. doi: 10.1198/TECH.2009.08104. PMID: 20607102.

Robert F. Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, 50(4):987–1007, 1982.

Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. A comparison of arima and lstm in forecasting time series. In M. Arif Wani, editor, *Proceedings of the 17th IEEE International Conference on Machine Learning and Applications*, pages 1394–1401, Orlando, FL, USA, 2018. IEEE.

Alexander Lipton. Cryptocurrencies change everything. *Quantitative Finance*, 21(8):1257–1262, 2021. doi: 10.1080/14697688.2021.1944490.

Wenbo Ge, Pooia Lalbakhsh, Leigh Isai, Artem Lenskiy, and Hanna Suominen. Neural network–based financial volatility forecasting: A systematic review. *ACM Computing Surveys*, 55(1), 2022.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In Bengio et al. (2018).

Robert A. Wood, Thomas H. McInish, and J. Keith Ord. An investigation of transactions data for nyse stocks. *Journal of Finance*, 40(3):723–739, 1985.

Prem C. Jain and Gun-Ho Joh. The dependence between hourly prices and trading volume. *Journal of Financial and Quantitative Analysis*, 23(3):269–283, 1988.

Thomas H. McInish and Robert A. Wood. An analysis of intraday patterns in bid/ask spreads for nyse stocks. *Journal of Finance*, 47(2):753–764, 1992.

James Eaves and Jeffrey Williams. Are intraday volume and volatility u-shaped after accounting for public information? *American Journal of Agricultural Economics*, 92(1):212–227, 2010.

Larry J. Lockwood and Scott C. Linn. An examination of stock market return volatility during overnight and intraday periods, 1964-1989. *Journal of Finance*, 45(2):591–601, 1990. ISSN 00221082, 15406261. URL http://www.jstor.org/stable/2328672.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

Frederick C. Webber. *Multi-Objective Reinforcement Learning with Concept Drift*. PhD thesis, Air Force Institute of Technology, 2017.

Aswath Damodaran. *Damodaran on Valuation: Security Analysis for Investment and Corporate Finance*. Wiley, 2 edition, 2006.

Dogu Araci. Finbert: Financial sentiment analysis with pre-trained language models. In *arXiv*, 2019. URL https://arxiv.org/abs/1908.10063.

Tom M. Mitchell. *Machine Learning*. McGraw-Hill Education, 1 edition, 1997.

Sebastian Ruder, Matthew E. Peters, Swabha Swayamdipta, and Thomas Wolf. Transfer learning in natural language processing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*, pages 15–18, Minneapolis, MN, USA, 2019. Association for Computational Linguistics.

Wenjuan Han, Bo Pang, and Ying Nian Wu. Robust transfer learning with pretrained language models through adapters. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 854–861, Online, 2021. Association for Computational Linguistics.

Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors. *Advances in Neural Information Processing*

Systems 30, Long Beach, CA, USA, 2017. Curran Associates, Inc.

Hugo Larochelle, Marc'Aurelio Ranzato, Raia, Hadsell, Maria-Florina Balcan, and Hui Lin, editors. *Advances in Neural Information Processing Systems 33*, NIPS 2020, Vancouver, BC, Canada, 2020. Curran Associates, Inc.

Yoshua Bengio and Yann LeCun, editors. *3rd International Conference on Learning Representations*, ICLR 2015, San Diego, CA, USA, 2015.

Kristen Grauman, Erik Learned-Miller, Antonio Torralba, and Andrew Zisserman, editors. *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR, 2015. IEEE.

Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors. *Advances in Neural Information Processing Systems 27*, NIPS, 2014. Curran Associates, Inc.

Frank J. Fabozzi and Harry M. Markowitz, editors. *The Theory and Practice of Investment Management*, volume 1. John Wiley & Sons, Ltd, 2011. ISBN 9781118267028.

Doina Precup and Yee Whye Teh, editors. *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70, Sydney, NSW, Australia, 2017. JMLR.org.

Sanjoy Dasgupta and David McAllester, editors. *Proceedings of the 30th International Conference on Machine Learning*, ICML'13, 2013. JMLR.org.

Francis R. Bach and David M. Blei, editors. *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 37, 2015. JMLR.org.

Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors. *Advances in Neural Information Processing Systems 31*, NIPS'18, Montréal, QC, Canada, 2018. Curran Associates, Inc.

Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors. *Advances in Neural Information Processing Systems 32*, Vancouver, BC, Canada, 2019. Curran Associates, Inc.

John Lintner. The valuation of risk assets and the selection of risky investments in stock portfolios and capital budgets. *Review of Economics & Statistics*, 47(1):13–37, 1965. ISSN 00346535.

Fischer Black. Capital market equilibrium with restricted borrowing. *Journal of Business*, 45 (3):444–455, 1972. ISSN 00219398, 15375374.

Malcolm Baker, Brendan Bradley, and Jeffrey Wurgler. Benchmarks as limits to arbitrage: Understanding the low-volatility anomaly. *Financial Analysts Journal*, 67(1):40–54, 2011.

Richard Roll. A critique of the asset pricing theory's tests part i: On past and potential testability of the theory. *Journal of Financial Economics*, 4(2):129–176, 1977. ISSN 0304-405X. doi: https://doi.org/10.1016/0304-405X(77)90009-5.

Eugene F. Fama and Kenneth R. French. The capital asset pricing model: Theory and evidence. *Journal of Economic Perspectives*, 18(3):25–46, September 2004. doi: 10.1257/0895330042162430. URL http://www.aeaweb.org/articles?id=10.1257/0895330042162430.

John H. Cochrane. *Asset Pricing*. Princeton University Press, 2005. ISBN 0691121370.

Richard G. Sloan. Do stock prices fully reflect information in accruals and cash flows about future earnings? *Accounting Review*, 71(3):289–315, 1996.

Michael J. Cooper, Huseyin Gulen, and J. Schill Michael. Asset growth and the cross-section of stock returns. *Journal of Finance*, 63(4):1609–1651, 2008.

Andrew Ang, Robert J. Hodrick, Yuhang Xing, and Xiaoyan Zhang. The cross-section of volatility and expected returns. *Journal of Finance*, 61(1):259–299, 2006. doi: 10.1111/j.1540-6261.2006.00836.x.

Robert Novy-Marx. The other side of value: The gross profitability premium. *Journal of Financial Economics*, 108(1):1–28, 2013. ISSN 0304-405X. doi: https://doi.org/10.1016/j.jfineco.2013.01.003.

Stephen A. Ross. The arbitrage theory of capital asset pricing. *Journal of Economic Theory*, 13 (3):341–360, 1976. ISSN 0022-0531. doi: https://doi.org/10.1016/0022-0531(76)90046-6.

Frank J. Fabozzi, Raman Vardharaj, , and Frank J. Jones. *Multifactor Equity Risk Models*, chapter 13, pages 327–343. Volume 1 of Fabozzi and Markowitz (2011), 2011c. ISBN 9781118267028.

CFA Institute. A practitioner's guide to factor models. Technical report, The Research Foundation of The Institute of Chartered Financial Analysts, Charlottesville, VA, USA, 1994. URL https://www.cfainstitute.org/-/media/documents/book/

rf-publication/1994/rf-v1994-n4-4445-pdf.ashx.

Victor Dheur and Souhaib Ben Taieb. A large-scale study of probabilistic calibration in neural network regression. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202, pages 7813–7836. JMLR.org, 2023.

APPENDIX A

Appendix

A1 Supplementary review of asset pricing

A supplementary review of the finance theory relevant for predictions in financial markets is provided in this section. Readers are encouraged to this read this section as preface to Section 1.3. However, this is not a prerequisite to understanding the rest of this thesis.

Consider the following scenario, an investor is making an investment decision between two assets: 1) a government bond that pays a guaranteed coupon; and 2) a highly risky stock. Assuming that the investor is risk averse, it is clear that the stock must offer a higher return than the government bond to compensate for the higher risk. Given an asset that earns an uncertain return, how much should an investor expect in return for holding this asset? In financial theory, this problem is known as *asset pricing*. The seminal work by Markowitz (1952) led to the development of the MPT, proposing to model the first two moments of stock returns. Suppose asset i has *expected return* $\mathrm{E}[r_i]$ and *risk* measured as variance of expected return σ_i^2. Investors are assumed to be risk averse and seek to select portfolios that maximise expected portfolio return,

$$\mathrm{E}[r_p] = \sum_i \mathrm{w}_i \, \mathrm{E}[r_i], \tag{A.1}$$

where r_p is portfolio return, w_i is portfolio holdings of asset i (for simplicity, it is assumed that $\sum_i \mathrm{w}_i = 1$), and minimise portfolio return variance,

$$\sigma_p^2 = \sum_i \mathrm{w}_i^2 \, \sigma_i^2 + \sum_i \sum_{j \neq i} \mathrm{w}_i \, \mathrm{w}_j \, \sigma_i \sigma_j \rho_{ij}, \tag{A.2}$$

where σ_p^2 is portfolio return variance, ρ_{ij} is correlation between expected return of asset i and j. This led to the development of the first asset pricing model, called the CAPM (Sharpe, 1964; Lintner, 1965; Black, 1972). The CAPM offered the first stock return prediction model with a theoretical underpinning,

$$\mathrm{E}[r_i] = r^* + \beta_i(\mathrm{E}[r'] - r^*), \tag{A.3}$$

where r^* is the risk-free rate (e.g., return on government bonds which are assumed to be risk-free), $\mathrm{E}[r']$ is the expected return of the market[1], and $\beta_i = \frac{\mathrm{Cov}(r_i, r')}{\mathrm{Var}(r')}$ is the sensitivity of expected return of asset i to the market. Exclusive to this section, we denote statistics related to the risk free rate with asterisk $*$ and the market with dash $'$. In CAPM, $\mathrm{E}[r']$ can be interpreted as factor m in Equation 1.3. However, it is typically assumed to be the long-term average market return. The intuition for CAPM is as follows. Suppose there are two stocks, both have expected return of $8\,\%$ and return standard deviation of $10\,\%$. Correlation between the two stocks is $50\,\%$. Then, from Equation (A.1) and (A.2), a portfolio with $50\,\%$ weight in each stock has portfolio expected return of $8\,\%$ and standard deviation of $7.9\,\%$. Thus, the diversified portfolio offers a better return/risk trade-off than either of the two stocks on its own. Generalising, there exists a well-diversified portfolio that offers the same level of expected return with equal or less level of risk (due to imperfect correlation between assets), as illustrated on the left of Figure A.1. The red curve on the left of Figure A.1 illustrates the *Efficient Frontier*, which represents the portfolio with the lowest possible risk (x-axis) for each level of expected return (y-axis). Note that all individual stocks (e.g., *C* and *D*) must lie on or within the interior of the efficient frontier. MPT assumes that there exists a well-diversified portfolio which offers the optimal mean-variance trade-off. This portfolio is called the *market portfolio*. The tangent to the red curve which crosses the y-axis at the risk-free rate is called the Security Market Line (SML). Note that SML intersects the Efficient Frontier at the market portfolio and is above the Efficient Frontier for all other levels of risk. Portfolios on this line represent allocations between the risk-free rate and the market portfolio. For example, if the desired level of risk is point **A** in Figure A.1, the level of risk is lower than the market portfolio

[1]The *market portfolio* is the portfolio of all assets weighted by their respective market value. Market value (also called market capitalisation) is the price of each stock multiplied by the number of shares on issue. Excess return of the market $\mathrm{E}[r'] - r^*$ is known as *market risk premium*.

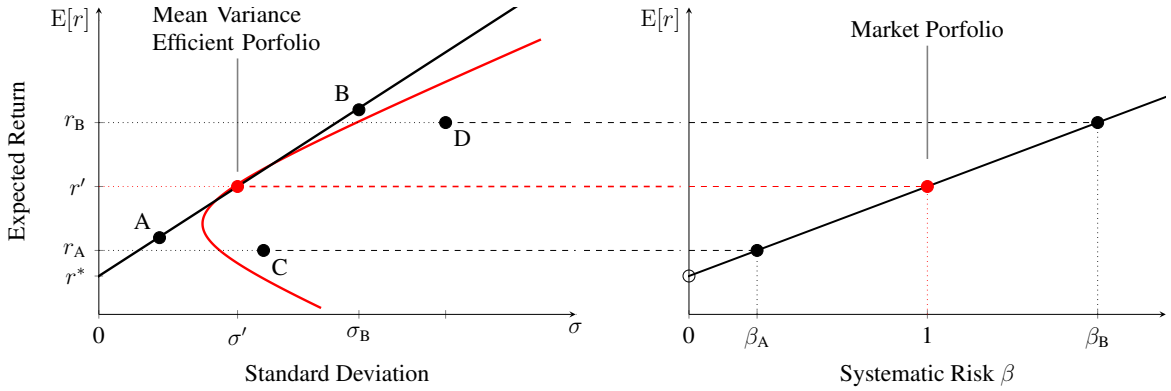


FIGURE A.1: Illustration of the Capital Asset Pricing Model. **Left**: Expected return of assets relative to standard deviation of returns. The red curve is known as the *Efficient Frontier*, where every point on the curve represents a possible portfolio combination. The tangent that passes through r^* (risk free rate) is the *Security Market Line*. **Right**: Expected return relative to systematic risk β. The market portfolio has $\beta' = 1$. A stock that has $\beta < 1$ (point **C**) is expected to earn a lower return than the market, while a stock with $\beta > 1$ (point **D**) is expected to earn a higher return than the market.

(or any risky portfolios on the Efficient Frontier). Point **A** represents a partial allocation to the risk-free rate (i.e., invest in government bonds) with the balance invested in the market portfolio. Conversely, if the desired level of risk is point **B**, the investor borrows at risk-free rate and invests the entire amount in the market portfolio (in MPT, borrowing at the risk-free rate is assumed to be accessible by all investors). In either case, the investor can achieve a better return/risk trade-off by allocating between the risk-free rate and the market portfolio than any of the portfolios on the Efficient Frontier or individual assets. Thus, a corollary of CAPM is that all investors hold the market portfolio and vary their allocation to the risk-free rate to arrive at their risk preference.

Due to the diversification benefits of the market portfolio, CAPM stipulates that the expected return of an individual asset is determined by its correlation to the market portfolio, known as systematic risk[2], and not by the total risk (i.e., variance) of the asset. In other words, only risk that arises due to correlation with the market portfolio is compensated (in the form of higher expected return), as this cannot be further diversified away by holding the market

[2]Denoted as β_i or *beta* in finance literature. Systematic risk is assumed to be non-diversifiable (Sharpe, 1964).

portfolio. Idiosyncratic risk that is specific to the asset is not compensated as there exists a well-diversified portfolio that offers the same level of expected return at a lower risk. Expected return as determined by correlation to the market is illustrated on the right of Figure A.1. In here, the market portfolio has $\beta' = 1$, while stock A (B) with $\beta_A < 1$ ($\beta_B > 1$) is expected to earn return $\mathrm{E}[r_A] < E[r']$ ($\mathrm{E}[r_B] > E[r']$), respectively. Underpinning both MPT and CAPM is the *efficient market hypothesis* (Fama, 1970). In the weak form, the hypothesis postulates that investors cannot outperform the market using publicly known information.

Therefore, one may arrive at the conclusions: 1) predicting stock returns is just a matter of estimating β; and 2) portfolio selection is of little value given that the market portfolio offers the optimal return/risk trade-off. However, empirical evidence proved otherwise. Jensen (1968) was the first to note that CAPM implied a time-series regression,

$$r_{i,t} - r^* = \alpha_i + \beta_i(r'_t - r^*) + \epsilon_{i,t}, \tag{A.4}$$

where $r_{i,t}$ is return of stock i at time t (at a given frequency, e.g. monthly), r'_t is market return at t and $\epsilon_{i,t}$ is stochastic noise. The empirical evidence suggested that α was statistically significantly above zero, which contradicted the predictions of CAPM. A positive α indicates that low β stocks earn a return that is higher than fair compensation for bearing market risk, an anomaly known as the *low beta effect* (Baker et al., 2011). Roll (1977) provided a renown critique of CAPM, noting the that a test of the validity of CAPM is a joint test together with the validity of the market portfolio proxy, and that the true market portfolio is unobservable (as it emcompasses all assets in the world, including private assets). A more recent discussion on CAPM was provided by Fama and French (2004). The authors reviewed published empirical tests of CAPM and concluded that "CAPM's empirical problems probably invalidate its use in applications."

Since the publication of CAPM, numerous other firm features (also known as risk factors or asset pricing *anomalies*[3]) were found to predict cross-sectional stock returns. These include

[3]A variable that is predictive of stock returns is called an *anomaly* as its occurrence contradicted the prediction of CAPM (Cochrane, 2005).

the size effect[4] (Banz, 1981), the value effect[5] (Stattman, 1980; Rosenberg et al., 1985)), accruals[6] (Sloan, 1996), asset growth[7] (Cooper et al., 2008), momentum[8] (Jegadeesh and Titman, 1993), low volatility effect[9] (Ang et al., 2006) and gross profitability[10] (Novy-Marx, 2013). Hundreds of firm characteristics are said to contain information on future stock returns — a survey by Harvey et al. (2016) contained 313 published asset pricing anomalies. The true DGP or stock returns is likely to be significantly more complex than originally suggested by CAPM.

Alternative asset pricing theories have been proposed. One alternative is the APT (Ross, 1976), formally,

$$\mathrm{E}[r_{i,t+1}] = r^* + s_{i,1}\xi_{1,t} + s_{i,2}\xi_{2,t} + ... + s_{i,M}\xi_{M,t}, \tag{A.5}$$

where $s_{i,m}$ is sensitivity (also called exposures in finance literature, e.g., Grinold and Kahn, 1999) of stock i to returns of *systematic risk factor*[11] ξ_m (also called factor returns). APT is a generalisation of CAPM, where systematic risk to the market in CAPM is replaced with any number of factors that determine stock returns. The intuition of the model lies in the assumption that given M non-diversifiable risk factors, one can replicate risk exposures of any stock (i.e., sensitivities of the stock to M predictors) with a portfolio of stocks that offer the same return. If the two do not offer the same return, an investor can buy the asset with a higher return and short[12] the asset with a lower expected return and generate a risk-free profit. Such activity is called *arbitrage* (hence the name Arbitrage Pricing Theory) and leads to the *no arbitrage condition* under efficient markets. This model forms the basis of quantitative investing methods used by practitioners, as described in Grinold and Kahn (1999). However,

[4]Stocks with low market capitalisation tend to outperform stocks with high market capitalisation.

[5]Stocks with high book value to market value ratio (known as value stocks) tend to outperform stocks with low book-to-market ratio (known as growth stocks).

[6]Firms with low operating accruals tend to outperform firms with high accruals. Operating accrual refers to the difference between accrued earnings and cash earnings.

[7]Firms with low year-on-year change in total assets tend to outperform firms with high growth.

[8]Past winners tend to continue to outperform past losers.

[9]Stocks with low idiosyncratic volatility tend to outperform stocks with high idiosyncratic volatility.

[10]Stocks with high gross profit to total assets tend to outperform stocks with low gross profit to total assets.

[11]Systematic risk factors are factors that contribute to expected return. An example is the market portfolio in CAPM. For the purposes of this thesis, this can be interpreted as firm features (e.g., firm profitability, social media sentiment), to the extent that such features are predictive of stock returns.

[12]Shorting refers to the borrowing and selling of an asset that the investor do not own. If the price of the asset falls, the investor can buy it back at a later date and generate a profit.

APT does not stipulate what the risk factors are. The risk factors used are up to the user to define. As concluding remarks on asset pricing theory, the debate on the correct model is one between a highly prescriptive model that specifies a theory-derived return predictor (CAPM β), and a loosely defined model where there could be any number of predictors (APT). The empirical failings of CAPM and the vast number of "anomalies" discovered is providing evidence to the latter.

A2 Supplementary review of forecasting models

A supplementary review of the quantitative investment process is provided in this section. Readers are encouraged to review this section for a better understanding of the quantitative investment process. However, this is not a prerequisite to understanding the rest of this thesis.

A2.1 Forecasting returns

Return forecasting (as illustrated in Figure A.2) requires converting raw data into features. Raw data can comprise of stock price history, company financials, social media sentiment,

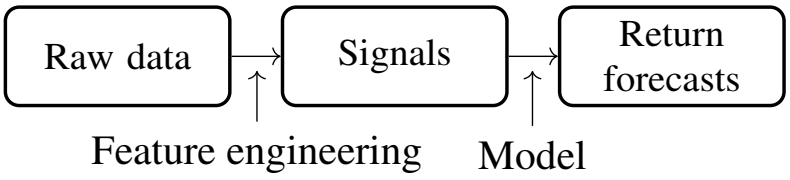


FIGURE A.2: Illustrative steps in converting raw data into forecasts. Raw data (e.g., share price and company financials) are first converted into features, typically crafted using domain knowledge. A forecasting model (e.g., OLS or machine learning models) then combines features into an expected return.

media reports, commodity prices, currency exchange rates or any other information that is predictive of returns. Through feature engineering, these raw data are transformed into signals. For example, the Earnings/Price ratio is calculated as a firm's earnings per share divided by the prevailing share price (Alford et al., 2011). This signal constitutes one of the many features that are used as inputs to the forecasting model. A linear regression, such as the

Fama-MacBeth regression, is a popular choice of F in Equation (1.2) (Zhou and Fabozzi, 2011)). However, as shown by Gu et al. (2020) and through this thesis, machine learning models can improve forecast accuracy and offer alternative approaches to forecasting.

A2.2 Forecasting risk

In the pioneering work of Markowitz (1952), a portfolio is viewed in terms of its first two moments of portfolio returns (mean and variance). This is the industry accepted general measure of risk (for example, see Grinold and Kahn, 1999; and Fabozzi and Markowitz, 2011). However, estimating a full variance-covariance matrix of a large universe of stocks could be onerous. For instance, there are more than 5,000 listed stocks in the U.S. alone. A variance-covariance matrix for the U.S. market could have $5000 \times 5000 = 25$million entries. Therefore, the variance-covariance matrix is typically estimated using a *factor risk model* (Fabozzi et al., 2011c). A factor risk model takes the same form as the return forecasting model (Equation (1.3)) and may include factors that are assumed to explain riskiness of a stock, such as its company size and debt burden. Suppose $M = 10$, the variance-covariance matrix only has $10 \times 10 = 100$ entries — a significantly simpler computation than operating over 25 million entries. The factor risk model is estimated using the Fama-MacBeth procedure as described in the preceding section. At time t, we perform $t-1$ cross-sectional multivariate regressions for every time period. This generates m time-series of factor returns $\hat{\boldsymbol{\xi}}_{t-1,m} = \{\hat{\xi}_{1,m}, \hat{\xi}_{2,m}, \ldots, \hat{\xi}_{t-1,m}\}$, where each value $\hat{\xi}_{t,m}$ in the series is the coefficient of factor[13] m in the multivariate cross-sectional regression at t. The factor covariance matrix is simply the sample variance-covariance matrix of the time-series of factor returns,

$$\hat{\boldsymbol{V}}_t^{(f)} = \begin{bmatrix} \mathrm{Cov}(\hat{\boldsymbol{\xi}}_{t-1,1}, \hat{\boldsymbol{\xi}}_{t-1,1}) & \mathrm{Cov}(\hat{\boldsymbol{\xi}}_{t-1,1}, \hat{\boldsymbol{\xi}}_{t-1,2}) & \cdots & \mathrm{Cov}(\hat{\boldsymbol{\xi}}_{t-1,1}, \hat{\boldsymbol{\xi}}_{t-1,M}) \\ \mathrm{Cov}(\hat{\boldsymbol{\xi}}_{t-1,2}, \hat{\boldsymbol{\xi}}_{t-1,1}) & \mathrm{Cov}(\hat{\boldsymbol{\xi}}_{t-1,2}, \hat{\boldsymbol{\xi}}_{t-1,2}) & \cdots & \mathrm{Cov}(\hat{\boldsymbol{\xi}}_{t-1,2}, \hat{\boldsymbol{\xi}}_{t-1,M}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cov}(\hat{\boldsymbol{\xi}}_{t-1,M}, \hat{\boldsymbol{\xi}}_{t-1,1}) & \mathrm{Cov}(\hat{\boldsymbol{\xi}}_{t-1,M}, \hat{\boldsymbol{\xi}}_{t-1,2}) & \cdots & \mathrm{Cov}(\hat{\boldsymbol{\xi}}_{t-1,M}, \hat{\boldsymbol{\xi}}_{t-1,M}) \end{bmatrix}, \quad \text{(A.6)}$$

[13] As a reminder to readers, signals, factors and firm features are used interchangeably in this thesis.

where $\mathrm{Cov}(\hat{\boldsymbol{\xi}}_{t-1,i}, \hat{\boldsymbol{\xi}}_{t-1,j})$ is the covariance function applied to returns of factor i and j over $t = \{1, \ldots, t-1\}$. Covariance matrix of stocks is then estimated as the sum of *factor risk* and *specific risk* (Fabozzi et al., 2011c),

$$\boldsymbol{V}_t' = \boldsymbol{X}_t \boldsymbol{V}_t^{(f)} \boldsymbol{X}_t^{\mathsf{T}} + \boldsymbol{d}_t \circ \boldsymbol{I}_N, \tag{A.7}$$

where $\boldsymbol{d}_t \circ \boldsymbol{I}_N$ is the Hadamard product of specific variance and the identity matrix of size N. $\boldsymbol{d}_t$ is the element-wise variance of the time-series of regression residuals $\boldsymbol{\epsilon}_t$ in Equation (1.3) (CFA Institute, 1994). This allows newly listed stocks to have reasonable risk forecasts, provided that they have the relevant inputs to the risk model.

A2.3 Forecasting transaction costs

As noted in Section 1.2, the stock market employs a *limit order book* system. In the example illustrated in Table 1.1, the last traded price is \$1.00/share. Suppose the buyer would like to transact 10,000 shares immediately, pushing the last traded price to \$1.02 (second row of the ask queue). The buyer's average price is $\frac{5{,}000 \times 1.01 + 5{,}000 \times 1.02}{10{,}000} = 1.015$. In this scenario, the buyer has incurred a market impact cost of $1.5\,\%$. Market impact comprises of the *bid-ask spread*[14] and a cost for pushing the traded price higher than the last traded price before the trade. Market impact depends on the size of the trade and tends to be higher for low-price and small capitalisation stocks (Zhou and Fabozzi, 2011). Larger trades will consume more of the available volume (termed liquidity) and have a higher price impact. Forecasting market impact is difficult, as market participants only observe completed trades and not intended orders. For example, in the example illustrated in Table 1.1, suppose a buyer wants to trade 100,000 shares at \$1.01. Seeing the limited shares on offer and concerned about potential price impact, the buyer decides to not place the order. However, there could be other potential sellers wanting to sell large parcels but face the same problem. Thus, liquidity can potentially be more than actual traded volumes. Transaction cost forecasting is also an important topic as it determines the economics of a trade. For instance, suppose the expected return of a stock over a week is $2\,\%$ and this opportunity is available every week. This results in an attractive

[14] *Crossing the spread* is when a buyer pays the ask price or a seller takes the bid price for a trade. Bid-ask spread is typically measured relative to the mid price and is calculated as $\frac{2 \times (1.01 - 1)}{1.01 + 1} = 1\,\%$.

compounded pre-cost return of $180\,\%$ in one year. However, suppose transaction cost is $1\,\%$ each way (i.e., $1\,\%$ is paid on acquisition and on disposal). Then, this seemingly profitable strategy is no longer profitable. Additionally, each trade incurs a commission to the broker (facilitator of the trade, usually a brokerage firm).

A3 Hyperparameters used in Chapter 3

In this section, we provide a list of hyperparameter search ranges and mean hyperparameters used to train the neural networks in Section 3.5. Hyperparameter search was performed on all combinations of hyperparameters. Table A.2 records mean hyperparameters over 10 network trainings.

TABLE A.1: Disclosed model parameters in Gu et al. (2020) and in our replication. We fill missing values with the cross-sectional median or zero if median is unavailable. 'H' is hidden layer activation. 'O' is output layer activation. ADAM is the optimiser proposed by Kingma and Ba (2015).

Parameter	Gu et al. (2020)	Chapter 3
Preprocessing	Rank [-1, 1]; Fill median	Rank [-1, 1]; Fill median/0
Hidden layers	32-16-8	32-16-8
Activation	H: ReLU / O: Linear	H: ReLU / O: Linear
Batch size	10,000	DNN 10,000 / OES 1,000
Batch normalisation	Yes	Yes
L_1 penalty	$[10^{-5}, 10^{-3}]$	$\{10^{-5}, 10^{-4}, 10^{-3}\}$
Early stopping	Patience 5	Patience 5 / Tolerance 0.001
Learning rate η	$[0.001, 0.01]$	$\{0.001, 0.01\}$
Optimiser	ADAM	ADAM
Loss function	MSE	MSE
Ensemble	Average over 10	Average over 10

TABLE A.2: Mean hyperparameters are calculated over the ensemble of 10 networks and across all periods.

	With Interactions		W/O Interactions	
%	DNN	OES	DNN	OES
Mean L_1 penalty	0.0012	0.0154	0.0024	0.0028
Mean η	0.77	0.10	0.67	0.10

A4 Hyperparameters used in Chapter 4

In this section, we provide a list of hyperparameter search ranges and mean hyperparameters used to train the neural networks in Section 4.3.2. Hyperparameter search was performed on all combinations of hyperparameters. Table A.3 records common hyperparameters used in all models. Table A.4, A.5, A.6 and A.7 record hyperparameter search range and average hyperparameters of 10 networks for STAE & TCN, N-BEATS, LSTM and transformer, respectively.

TABLE A.3: Common hyperparameters for all networks used in Section 4.3.2. These are fixed values and are not subject to hyperparameter tuning. ADAM is the optimiser proposed by Kingma and Ba (2015).

Parameter	Common hyperparameters
Hidden layers	8
Activation	ReLU
Batch size	5,000
Batch normalisation	Yes
Early stopping	Patience 5 / Tolerance 0.0001
Learning rate η	0.01
Optimiser	ADAM

TABLE A.4: STAE and TCN specific hyperparameter ranges used in Section 4.3.2. Values enclosed by $\{\cdot\}$ are choices within the set. Values enclosed by $[\cdot]$ are a single list where each value indicates a layer. Mean hyperparameters are average chosen hyperparameters in an ensemble of 10.

Parameter	STAE	TCN
Search range		
Convolutional filters	$\{8, 16, 32\}$	$\{8, 16, 32\}$
Encoder kernel size	$\{2, 5, 10\}$	$\{2, 5, 10\}$
Decoder kernel size	$[2, 5, 5, 5]$	
Spatial dropout	$\{0.2, 0.4\}$	$\{0.2, 0.4\}$
Auxiliary loss ω	$\{0.2, 0.4\}$	
Mean hyperparameters		
Convolutional filters	27.2	18.4
Encoder kernel size	2.9	6.9
Spatial dropout	0.28	0.34
Auxiliary loss ω	0.34	

TABLE A.5: N-BEATS specific hyperparameter ranges used in Section 4.3.2. Values enclosed by $\{\cdot\}$ are choices within the set. Values enclosed by $[\cdot]$ are a single list where each value indicates a layer. Mean hyperparameters are average chosen hyperparameters in an ensemble of 10.

Parameter	N-BEATS
Search range	
Stacks	[Trend, Generic]
Blocks per stack	$\{2, 3, 4\}$
θ dimenions	$\{[4, 8], [8, 16], [16, 32]\}$
Mean hyperparameters	
Blocks per stack	2.6
Mean θ dimenions	13.8

TABLE A.6: LSTM specific hyperparameter ranges used in Section 4.3.2. Values enclosed by $\{\cdot\}$ are choices within the set. Values enclosed by $[\cdot]$ are a single list where each value indicates a layer. Mean hyperparameters are average chosen hyperparameters in an ensemble of 10.

Parameter	LSTM
Search range	
LSTM layers	$\{[8], [16], [32], [16, 8], [32, 16], [32, 16, 8]\}$
Mean hyperparameters	
Mean total LSTM units	32.8
Mean no. LSTM layers	1.7

TABLE A.7: Transformer specific hyperparameter ranges used in Section 4.3.2. Values enclosed by $\{\cdot\}$ are choices within the set. Values enclosed by $[\cdot]$ are a single list where each value indicates a layer. Mean hyperparameters are average chosen hyperparameters in an ensemble of 10.

Parameter	Transformer
Search range	
Key size	$\{4, 8\}$
No. heads	$\{1, 2\}$
No. transformer blocks	1
Dropout rate	$\{0.2, 0.4\}$
Mean hyperparameters	
Key size	$\{4, 8\}$
No. heads	$\{1, 2\}$
No. transformer blocks	1
Dropout rate	$\{0.2, 0.4\}$

A5 Hyperparameters used in Chapter 5

In this section, we provide a list of hyperparameter search ranges and mean hyperparameters used to train the neural networks in Section 5.4.1 and 5.4.2. Hyperparameter search was performed on all combinations of hyperparameters. Both cryptocurrency and U.S. equities datasets share the same hyperparameter ranges but with hyperparameter search performed separately. Table A.9 records mean hyperparameters over 10 network trainings for each network architecture.

TABLE A.8: Hyperparameter ranges used in Section 5.4.1 and 5.4.2. The 'LSTM layers' hyperparameter is a list, with the length of the list indicating how many LSTM layers were used and each element of the list indicating the number of units of each LSTM layer. Similarly, 'Hidden layers' indicate the number of fully connected hidden layers. Each element of the list indicate the dimension of that hidden layer. ADAM is the optimiser proposed by Kingma and Ba (2015).

Parameter	Search range
LSTM layers	$\{[16, 8], [32, 16, 8], [32, 16], [64, 32, 16]\}$
Hidden layers	$\{[8], [16, 8]\}$
Dropout rate	$\{0.2, 0.3, 0.4\}$
Activation	ReLU
Batch size	1,000
Batch normalisation	Yes
Early stopping	Patience 5 / Tolerance 0.0001
Learning rate η	0.01
Optimiser	ADAM

TABLE A.9: Mean hyperparameters are calculated over the ensemble of 10 networks for Ensemble, Evidential and Combined. Mean LSTM and hidden units is the average number of LSTM or hidden units in the network. 'no.' indicate number of layers.

%	Ensemble	Evidential	Combined
Mean dropout rate	33	26	25
Mean total LSTM units	56	41.6	61.6
Mean no. LSTM layers	2.5	2.4	2.5
Mean total hidden units	19.2	11.2	12.8
Mean no. hidden layers	1.7	1.2	1.3

A6 Marginal distribution of a Scale Mixture

From Equation (5.10), we have $\mathrm{N}(y|\gamma, \frac{\sigma^2}{\lambda})\,\mathrm{Gam}(\lambda|\alpha,\beta)$. Marginalising over λ produces the data likelihood,

$$\begin{aligned}
\mathrm{p}(y|\gamma,\sigma^2,\alpha,\beta) &= \int_\lambda^\infty \mathrm{p_N}(y|\gamma,\sigma^2\lambda^{-1})\mathrm{p_G}(\lambda|\alpha,\beta)\,\mathrm{d}\lambda \\
&= \int_\lambda^\infty \left[\sqrt{\frac{\lambda}{2\pi\sigma^2}}\exp\left\{-\frac{\lambda(y-\gamma)^2}{2\sigma^2}\right\}\right]\left[\frac{\beta^\alpha}{\Gamma(\alpha)}\lambda^{\alpha-1}\exp^{-\beta\lambda}\right]\mathrm{d}\lambda \\
&= \frac{\beta^\alpha}{\Gamma(\alpha)\sqrt{2\pi\sigma^2}}\int_{\lambda=0}^\infty \lambda^{\alpha-\frac{1}{2}}\exp\left\{-\frac{\lambda(y-\gamma)^2}{2\sigma^2}-\beta\lambda\right\}\mathrm{d}\lambda \\
&= \frac{\beta^\alpha}{\Gamma(\alpha)\sqrt{2\pi\sigma^2}}\left[\frac{(y-\gamma)^2}{2\sigma^2}+\beta\right]^{-(\alpha+\frac{1}{2})}\int_{\lambda=0}^\infty\left\{\lambda\left[\frac{(y-\gamma)^2}{2\sigma^2}+\beta\right]\right\}^{\alpha-\frac{1}{2}} \\
&\quad \exp\left\{-\lambda\left[\frac{(y-\gamma)^2}{2\sigma^2}+\beta\right]\right\}\mathrm{d}\left\{\lambda\left[\frac{(y-\gamma)^2}{2\sigma^2}+\beta\right]\right\},
\end{aligned}$$

since $\int_0^\infty x^{\alpha-1}\exp(-x)\,\mathrm{d}x = \Gamma(\alpha)$,

$$= \frac{\beta^\alpha}{\sqrt{2\pi\sigma^2}}\frac{\Gamma(\alpha+\frac{1}{2})}{\Gamma(\alpha)}\left[\frac{(y-\gamma)^2}{2\sigma^2}+\beta\right]^{-(\alpha+\frac{1}{2})}$$

and re-arranging $\beta^\alpha = (\frac{1}{\beta})^{-\alpha} = (\frac{1}{\beta})^{-(\alpha+\frac{1}{2})+\frac{1}{2}}$,

$$\begin{aligned}
&= \frac{\Gamma(\alpha+\frac{1}{2})}{\Gamma(\alpha)}\frac{1}{\sqrt{2\pi\sigma^2\beta}}\left[\frac{(y-\gamma)^2}{2\sigma^2\beta}+1\right]^{-(\alpha+\frac{1}{2})} \\
\mathrm{p}(y|\gamma,\sigma^2,\alpha,\beta) &= \mathrm{St}\left(y;\gamma,\frac{\sigma^2\beta}{\alpha},2\alpha\right). \qquad \text{(A.8)}
\end{aligned}$$

To show that the last step of Equation (A.8) is true, we start with the probability density function of the t-distribution parameterised in terms of precision $\mathrm{St}(y|\gamma,b^{-1},a)$ (Bishop, 2006),

$$\mathrm{St}(y|\gamma,b^{-1},a) = \frac{\Gamma(\frac{a+1}{2})}{\Gamma(\frac{a}{2})}\left[\frac{b}{\pi a}\right]^{\frac{1}{2}}\left[1+\frac{b(y-\gamma)^2}{a}\right]^{-(\frac{a+1}{2})},$$

where γ is location, b is inverse of scale and a is shape[15]. Substituting in $b^{-1} = \frac{\sigma^2\beta}{\alpha}$ and $a = 2\alpha$,

$$\begin{aligned}\mathrm{St}\left(y|\gamma, \frac{\sigma^2\beta}{\alpha}, 2\alpha\right) &= \frac{\Gamma(\alpha+\frac{1}{2})}{\Gamma(\alpha)}\left[\frac{(\frac{\alpha}{\sigma^2\beta})}{2\pi\alpha}\right]^{\frac{1}{2}}\left[1+\frac{\alpha(y-\gamma)^2}{2\sigma^2\alpha\beta}\right]^{-(\alpha+\frac{1}{2})} \\ &= \frac{\Gamma(\alpha+\frac{1}{2})}{\Gamma(\alpha)}\frac{1}{\sqrt{2\pi\sigma^2\beta}}\left[\frac{(y-\gamma)^2}{2\sigma^2\beta}+1\right]^{-(\alpha+\frac{1}{2})}.\end{aligned}$$

A7 Negative log-likelihood of marginal distribution of a Scale Mixture

From Equation (A.8), the NLL of the marginal t-distribution is,

$$\mathrm{p}(y|\gamma,\sigma^2,\alpha,\beta) = \frac{\Gamma(\alpha+\frac{1}{2})}{\Gamma(\alpha)}\frac{1}{\sqrt{2\pi\sigma^2\beta}}\left[\frac{(y-\gamma)^2}{2\sigma^2\beta}+1\right]^{-(\alpha+\frac{1}{2})}$$

$$-\log[\mathrm{p}(y|\gamma,\sigma^2,\alpha,\beta)] = \log\left[\frac{\Gamma(\alpha)}{\Gamma(\alpha+\frac{1}{2})}\right] + \tfrac{1}{2}\log[2\pi\sigma^2\beta] + (\alpha+\tfrac{1}{2})\log\left[\frac{(y-\gamma)^2}{2\sigma^2\beta}+1\right].$$

A7.1 Benchmarking on UCI dataset

In this section, we compare our method to Ensemble and Evidential using the UCI benchmark dataset. This is intended to provide readers with a direct comparison to Lakshminarayanan et al. (2017) and Amini et al. (2020) on the same dataset used in both works. The collection consists of nine real world regression problems, each with 10–20 features and hundreds to tens of thousands of observations. We note the *Wine* dataset within UCI contains discrete values (ratings of wine characteristics, such as color and taste) which may render the assumption of a continuous, symmetrical data distribution less appropriate if these values are skewed. More recently, larger uncertainty quantification datasets have been published in Dheur and Ben Taieb (2023), which may be useful in assessing the state-of-the-art in non-time-series uncertainty quantification methods. We follow Lakshminarayanan et al. (2017) and Amini

[15]Note that the definition of scale b and shape a is used exclusively in this section. Not to be confused with network bias b and activation vector $\boldsymbol{a}$ used in the rest of this thesis.

TABLE A.10: Comparing Ensemble (Lakshminarayanan et al., 2017), Evidential (Amini et al., 2020) and Combined (this work) on RMSE and NLL using the UCI benchmark datasets. Average result and standard deviation over 5 trials for each method. The best method for each dataset and metric are highlighted in **bold**.

	RMSE			NLL		
Dataset	**Ensemble**	**Evidential**	**Combined**	**Ensemble**	**Evidential**	**Combined**
Boston	$\mathbf{2.66 \pm 0.20}$	2.95 ± 0.29	2.89 ± 0.31	2.28 ± 0.05	2.30 ± 0.05	$\mathbf{2.23 \pm 0.05}$
Concrete	5.79 ± 0.16	5.98 ± 0.23	$\mathbf{5.40 \pm 0.18}$	3.07 ± 0.02	3.11 ± 0.04	$\mathbf{2.98 \pm 0.03}$
Energy	1.86 ± 0.04	1.84 ± 0.06	$\mathbf{1.71 \pm 0.20}$	1.36 ± 0.02	1.41 ± 0.04	$\mathbf{1.35 \pm 0.05}$
Kin8nm	$\mathbf{0.06 \pm 0.00}$	0.06 ± 0.00	0.06 ± 0.00	$\mathbf{-1.39 \pm 0.02}$	-1.28 ± 0.03	-1.35 ± 0.02
Naval	$\mathbf{0.00} \pm 0.00$	0.00 ± 0.00	0.00 ± 0.00	$\mathbf{-6.10 \pm 0.05}$	-5.99 ± 0.09	-5.89 ± 0.35
Power	3.02 ± 0.09	3.02 ± 0.08	$\mathbf{2.95 \pm 0.08}$	2.57 ± 0.01	2.56 ± 0.03	$\mathbf{2.53 \pm 0.02}$
Protein	3.71 ± 0.10	4.28 ± 0.23	$\mathbf{3.67 \pm 0.13}$	$\mathbf{2.61 \pm 0.03}$	2.73 ± 0.08	2.70 ± 0.05
Wine	0.60 ± 0.03	$\mathbf{0.56 \pm 0.02}$	0.59 ± 0.03	0.94 ± 0.04	$\mathbf{0.92 \pm 0.04}$	1.00 ± 0.03
Yacht	$\mathbf{1.22 \pm 0.22}$	1.48 ± 0.47	3.97 ± 1.06	1.06 ± 0.08	$\mathbf{0.96 \pm 0.19}$	1.17 ± 0.11

et al. (2020) in evaluating our method on root mean squared error (RMSE, which assesses forecast accuracy) and NLL (which assesses overall distributional fit), and compare against Ensemble and Evidential. While we do not explicitly compare inference speed, as our Combined method also uses ensembling, inference speed is expected to be comparable to Ensemble while being slower than Evidential. We use the source code provided by Amini et al. (2020), with the default topology of a single hidden layer with 50 units for both Ensemble and Evidential[16]. For Combined, as individual modelling of distribution parameters (Section 5.3.2) requires a network with two or more hidden layers, we have used a single hidden layer with 24 units, followed by 4 separate stacks of a single hidden layer with 6 units each. Thus, the total number of non-linear units is 48 (compared to 50 for Ensemble and Evidential). Note that even though the total number of units are similar across the three models, learning capacity may differ due to different topologies.

Table A.10 records experiment results on the UCI dataset. On RMSE, we find that both Ensemble and Combined have performed well, having the best RMSE in four datasets each. In two of the sets (*Kin8nm* and *Naval*), all three methods produced highly accurate results that are not separable to two decimal points. Turning to NLL, we observe a trend towards Combined having lower NLL than the other two methods for four sets, followed

[16]Source code for Amini et al. (2020) is available on Github: https://github.com/aamini/evidential-deep-learning

by Ensemble with three sets. Comparing Combined to Evidential, we find that Combined generally has lower RMSE (7 of 9 sets) and NLL (6 of 9 sets). Although our method is designed for uncertainty quantification of complex time-series and all 9 datasets are pooled (non-time-series) datasets, we still observe some improvements in both RMSE and NLL.

Next, we present further ablation studies on the UCI dataset. Table A.11 records results of *Alternative*, which utilizes ensembling and SMD parameterisation but not separate modelling of hyperparameters. Alternative has the same network topology as Ensemble and Evidential (a single hidden layer with 50 units), as opposed to Combined which has two hidden layers with a total of 48 units. We observe that Ensemble has the lowest RMSE in 5 (of 9) datasets, followed by Alternative (3 of 9), while Alternative has the best NLL in 6 (of 9) datasets and Ensemble has 3 (of 9). On both metrics, Evidential has the least favourable performance. Comparing Combined in Table A.10 and Alternative in Table A.11, Combined has lower RMSE and NLL in 5 of 9 datasets. Thus, we conclude that separate modelling of hyperparameters provided an incremental benefit on the UCI datasets.

TABLE A.11: Comparing Ensemble, Evidential and Alternative (without separate modelling of the four parameters of SMD) on RMSE and NLL using the UCI benchmark datasets. Average result and standard deviation over 5 trials for each method. The best method for each dataset and metric is highlighted in **bold**.

	RMSE			NLL		
Dataset	**Ensemble**	**Evidential**	**Alternative**	**Ensemble**	**Evidential**	**Alternative**
Boston	$\mathbf{2.66 \pm 0.20}$	2.95 ± 0.29	2.87 ± 0.18	$\mathbf{2.28 \pm 0.05}$	2.30 ± 0.05	2.29 ± 0.04
Concrete	5.79 ± 0.16	5.98 ± 0.23	$\mathbf{5.72 \pm 0.15}$	3.07 ± 0.02	3.11 ± 0.04	$\mathbf{3.03 \pm 0.02}$
Energy	1.86 ± 0.04	$\mathbf{1.84 \pm 0.06}$	1.88 ± 0.04	1.36 ± 0.02	1.41 ± 0.04	$\mathbf{1.35 \pm 0.03}$
Kin8nm	$\mathbf{0.06 \pm 0.00}$	0.06 ± 0.00	0.06 ± 0.00	$\mathbf{-1.39 \pm 0.02}$	-1.28 ± 0.03	-1.38 ± 0.02
Naval	$\mathbf{0.00 \pm 0.00}$	0.00 ± 0.00	0.00 ± 0.00	-6.10 ± 0.05	-5.99 ± 0.09	$\mathbf{-6.12 \pm 0.06}$
Power	3.02 ± 0.09	3.02 ± 0.08	$\mathbf{2.97 \pm 0.10}$	2.57 ± 0.01	2.56 ± 0.03	$\mathbf{2.54 \pm 0.02}$
Protein	$\mathbf{3.71 \pm 0.10}$	4.28 ± 0.23	3.75 ± 0.11	$\mathbf{2.61 \pm 0.03}$	2.73 ± 0.08	2.72 ± 0.02
Wine	0.60 ± 0.03	0.56 ± 0.02	$\mathbf{0.55 \pm 0.02}$	0.94 ± 0.04	0.92 ± 0.04	$\mathbf{0.92 \pm 0.02}$
Yacht	$\mathbf{1.22 \pm 0.22}$	1.48 ± 0.47	1.45 ± 0.33	1.06 ± 0.08	0.96 ± 0.19	$\mathbf{0.93 \pm 0.09}$

In Table A.12, we further remove model averaging. The network used is identical to Evidential but trained using the SMD parameterisation (i.e., we simply change the loss function in Evidential to Equation (5.12)). We observe that the network trained using the SMD parameterisation has lower RMSE in 6 of 9 and lower NLL in 8 out 9 datasets. We argue that the improved performance of the SMD parameterisation is due to its simplicity.

TABLE A.12: Comparing Normal-Inverse-Gamma and Normal-Gamma on RMSE and NLL using the UCI benchmark datasets. Average result and standard deviation over 5 trials for each method. The best method for each dataset and loss function is highlighted in **bold**.

	RMSE		NLL	
Dataset	**NIG**	**SMD**	**NIG**	**SMD**
Boston	$\mathbf{2.95 \pm 0.29}$	2.97 ± 0.20	$\mathbf{2.30 \pm 0.05}$	2.31 ± 0.05
Concrete	5.98 ± 0.23	$\mathbf{5.78 \pm 0.23}$	3.11 ± 0.04	$\mathbf{3.05 \pm 0.04}$
Energy	$\mathbf{1.84 \pm 0.06}$	1.87 ± 0.16	1.41 ± 0.04	$\mathbf{1.33 \pm 0.05}$
Kin8nm	0.06 ± 0.00	$\mathbf{0.06 \pm 0.00}$	-1.28 ± 0.03	$\mathbf{-1.37 \pm 0.01}$
Naval	0.00 ± 0.00	$\mathbf{0.00 \pm 0.00}$	-5.99 ± 0.09	$\mathbf{-6.27 \pm 0.09}$
Power	3.02 ± 0.08	$\mathbf{2.98 \pm 0.12}$	2.56 ± 0.03	$\mathbf{2.53 \pm 0.02}$
Protein	4.28 ± 0.23	$\mathbf{3.72 \pm 0.16}$	2.73 ± 0.08	$\mathbf{2.39 \pm 0.05}$
Wine	$\mathbf{0.56 \pm 0.02}$	0.56 ± 0.03	0.92 ± 0.04	$\mathbf{0.87 \pm 0.04}$
Yacht	1.48 ± 0.47	$\mathbf{1.44 \pm 0.49}$	0.96 ± 0.19	$\mathbf{0.91 \pm 0.18}$

A8 Further analysis of parameters in a Scale Mixture

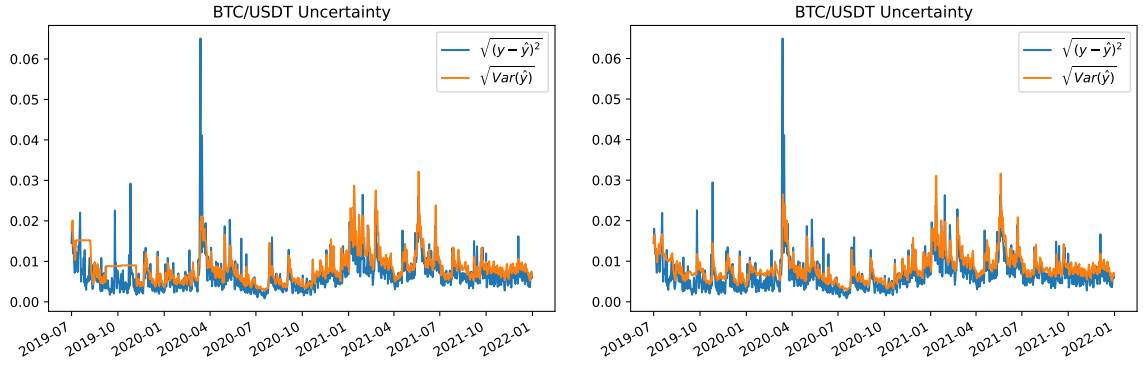
In the network architecture proposed in Section 5.3, output of the network is $\boldsymbol{\zeta} = (\gamma, \sigma^2, \alpha, \beta)$, which parameterises the SMD (Equation (5.10)). However, as noted in Section 5.3, we can set $\alpha = \beta$ and reduce the number of parameters to three (Equation (5.14)). Thus, an alternative specification of the network is to output $\boldsymbol{\zeta} = (\gamma, \sigma^2, \alpha)$ (i.e., three parameters instead of four and are computed through three subnetworks, instead of four in Figure 5.1). We label this network *A=B*. In Table A.13, we compare Combined (4 parameters) with S2B (3 parameters) using the UCI dataset (as introduced in Section A7.1). We observe that A=B is better than Combined on 8 (of 9) datasets on RMSE, while Combined is better than A=B on 1 (of 9). On NLL, A=B is better than Combined on 5 (of 9) datasets, while Combined is better than A=B on 4 (of 9). Even though A=B has a higher number of datasets with lower RMSE and NLL, we note that the differences are very small and are within margin of error (due to randomness in neural network training). Thus, we conclude that the two methods provide near identical results but note that A=B is simpler and more interpretable. However, we choose Combined with four subnetworks to conduct our analysis so that parameters can also be compared with those from Evidential and Extended Evidential in Appendix A9.

TABLE A.13: Comparing A=B (3 parameters) to Combined (4 parameters) on RMSE and NLL using the UCI benchmark datasets. Results are averaged over 5 trials and the best method for each dataset and metric are highlighted in **bold**.

	RMSE		NLL	
Dataset	A=B	Combined	A=B	Combined
Boston	2.91 ± 0.17	$\mathbf{2.89 \pm 0.31}$	2.27 ± 0.04	$\mathbf{2.23 \pm 0.05}$
Concrete	$\mathbf{5.39 \pm 0.19}$	5.40 ± 0.18	2.99 ± 0.03	$\mathbf{2.98 \pm 0.03}$
Energy	$\mathbf{1.56 \pm 0.16}$	1.71 ± 0.20	$\mathbf{1.30 \pm 0.05}$	1.35 ± 0.05
Kin8nm	$\mathbf{0.06 \pm 0.00}$	0.06 ± 0.00	$\mathbf{-1.36 \pm 0.02}$	-1.35 ± 0.02
Naval	$\mathbf{0.00 \pm 0.00}$	0.00 ± 0.00	-5.87 ± 0.12	$\mathbf{-5.89 \pm 0.35}$
Power	$\mathbf{2.93 \pm 0.08}$	2.95 ± 0.08	$\mathbf{2.53 \pm 0.02}$	2.53 ± 0.02
Protein	$\mathbf{3.60 \pm 0.10}$	3.67 ± 0.13	2.83 ± 0.04	$\mathbf{2.70 \pm 0.05}$
Wine	$\mathbf{0.57 \pm 0.02}$	0.59 ± 0.03	$\mathbf{0.96 \pm 0.03}$	1.00 ± 0.03
Yacht	$\mathbf{2.31 \pm 0.43}$	3.97 ± 1.06	$\mathbf{1.11 \pm 0.09}$	1.17 ± 0.11

A9 Further analysis of Evidential on uncertainty quantification in cryptocurencies

The Evidential method utilises NormalInverseGamma output layer (no separate subnetworks for distribution hyperparameters) and a t-distributed NLL derived from the NIG distribution (Equation (5.9)). In Section 5.4.1, we have observed that Evidential fails to provide uncertainty forecasts that track time-varying volatility. However, this was not observed in our proposed Combined method. In here, we test the effects of separate modelling of distribution hyperparameters in the output layer for Evidential. We label this as the *Extended Evidential* method. Square-root of average squared forecast error and square-root of forecast uncertainty for BTC/USDT for Extended Evidential and Combined are shown in Figure A.3. We find that separate modelling of distribution hyperparameters significantly improved accuracy of uncertainty forecasts of Extended Evidential. Forecast uncertainty of both Extended Evidential and Combined are generally similar, with the exception of still some block-like features for Extended Evidential in 2019. Comparing cryptocurrency experimental results of Extended Evidential to Combined, we find that Combined is better than Extended Evidential on both IC and NLL, and is better than Extended Evidential on RMSE at higher decimal places.



(a) Extended Evidential (b) Combined

FIGURE A.3: Actual prediction error and predicted uncertainty of Extended Evidential and Combined for BTC/USDT. Square root of the average forecast error and forecast uncertainty on each day shown.

TABLE A.14: Ablation study comparing Extended Evidential to Combined on average IC, RMSE and NLL for cryptocurrencies time-series forecasts. Average result and standard deviation over 10 trials for each method. Best method for each dataset is highlighted in **bold**.

Metric	Extended Evidential	Combined
IC (%)	6.39 ± 2.56	$\mathbf{9.87 \pm 3.17}$
RMSE (%)	0.867 ± 0.001	$\mathbf{0.867 \pm 0.001}$
NLL	-3.35 ± 0.01	$\mathbf{-4.14 \pm 0.01}$

Can't attach Re_ Thesis submission.eml, please review file online.