



<input type="checkbox"/>	Bachelor's thesis
<input checked="" type="checkbox"/>	Master's thesis
<input type="checkbox"/>	Licentiate's thesis
<input type="checkbox"/>	Doctoral dissertation

Subject	Economics	Date	23.6.2020
Author	Markus Mäkelä	Number of pages	103+appendices
Title	Predicting U.S. business cycles with recurrent neural networks. An extensive multivariate time-series analysis for comparing LSTM and GRU networks		
Supervisor	Prof. Jouko Vilmunen		

Abstract

This study examines how 22 different long short-term memory (LSTM) and gated recurrent unit (GRU) network architectures suit predicting U.S. business cycles. The networks create 91-day forecasts for the dependent variable by using multivariate time-series data comprising 26 leading indicators' values for the previous 400 days. The proposed models are evaluated by using a train-test split, where the proposed models are trained with data from 1980 to 2005, and the out-of-sample set consists of data between 2005 and 2015. The performance is evaluated by using mean squared error (MSE) and mean absolute error (MAE), and early warning signs are also considered beneficial.

The training algorithm consists of typical deep learning methods. MSE and L1 regularization are used for determining the cost, and minibatches of 32 examples are applied together with Nesterov accelerated momentum (NAG) learning algorithm. Early stopping is introduced to halt the training process when strong signs of overfitting are detected. Each proposed recurrent neural network (RNN) architecture is trained three times, and these three networks' averaged predictions are examined when comparing the architectures.

Performance-wise, a few LSTM networks stand out from the other proposed networks. Although the performance results favor the proposed LSTM networks slightly over their GRU equivalents, the difference is not substantial and, in turn, the proposed GRU networks offer less deviation in MSE and MAE between each architecture. However, these steadier performance results do not generate less volatile forecasts. Instead, the best performing networks and architectures differentiate by offering less volatile predictions that also vary less from the real values.

Most of the models generate a considerable amount of early warning signs before the 2007 recession, which indicates their suitability for detecting turning points in business cycles. Moreover, a wide range of the proposed LSTM and GRU network architectures learn the general pattern, also the smaller architectures comprising only one hidden layer and less than 500 optimizable parameters. This suggests that these methods offer noteworthy solutions for business cycle forecasting and, more widely, supports applying nonlinear machine learning methods with multivariate data for macroeconomic forecasting tasks where prevalent methods have been found unable to deliver adequate accuracy.

Key words	Business cycles, forecasting, machine learning, recurrent neural networks
-----------	---





<input type="checkbox"/>	Kandidaatintutkielma
<input checked="" type="checkbox"/>	Pro gradu -tutkielma
<input type="checkbox"/>	Lisensiaatintutkielma
<input type="checkbox"/>	Väitöskirja

Oppiaine	Taloustiede	Päivämäärä	23.6.2020
Tekijä	Markus Mäkelä	Sivumäärä	103+liitteet
Otsikko	Vertailu erikokoisten LSTM- ja GRU-neuroverkkojen soveltumisesta Yhdysvaltojen taloussykliennustamiseen		
Ohjaaja	Prof. Jouko Vilmunen		

Tiivistelmä

Tässä tutkielmassa vertaillaan 22 eri LSTM- ja GRU-neuroverkon soveltuvuutta Yhdysvaltojen taloussykliennustamiseen. Valittujen neuroverkkojen tehtävä on luoda 91 päivän ennusteita valitulle selitettävälle muuttujalle käyttämällä 400:n aikaisemman päivän havaintoarvoja 26:sta indikaattorista. Valittujen mallien optimoimiseen käytetään havaintoja ajanjaksolta 1980-2005 ja niiden arviointiin ajanjaksoa 2005-2015. Suorituskyvyn arvioimisessa sovelletaan keskineliövirhettä ja keskiabsoluuttistavirhettä. Tämän lisäksi aikaiset signaalit syklin kääntymisestä nähdään suotuisina.

Neuroverkkojen parametrien optimoimiseen käytetty algoritmi sisältää tyypillisiä syväoppimisen menetelmiä. Kustannus määritetään käyttämällä keskineliövirhettä ja L1-termiä. NAG-algoritmia käytetään parametrien päivittämiseen, jolle harjoitus instanssit syötetään 32 kappaleen erissä. Optimoiminen keskeytetään ennen takarajaa, mikäli saadaan merkittäviä viitteitä optimoitavan mallin ylisovittumisesta. Jokainen valittu neuroverkkoarkkitehtuuri treenataan kolme kertaa ja näiden kolmen neuroverkon tuottamien ennusteiden keskiarvoja käytetään pohjana eri arkkitehtuurien vertailussa.

Suorituskykyä tarkasteltaessa, muutama LSTM-neuroverkko pystyy saavuttamaan muita vaihtoehtoja paremman tarkkuuden. Vaikka suorituskyvystä kertovat tulokset suosivat valittuja LSTM-arkkitehtuureita, erot LSTM- ja GRU-neuroverkkojen suorituskyvyssä ovat keskimäärin pieniä. Toisaalta, GRU-menetelmät pystyvät tarjoamaan vähemmän vaihtelua arkkitehtuurien keskinäisten neuroverkkojen suorituskyvyssä, mutta tämä ei kuitenkaan johda vakaampiin ennusteisiin. Sen sijaan, parhaat suorituskyvyt antavat LSTM-neuroverkot erottautuvat muista tarjoamalla muita vakaampia ennusteita, jotka myös eroavat todellisista arvoista muita vähemmän. Suurin osa tutkituista malleista tuottaa huomattavan määrän signaaleita syklin vaihtumisesta ennen vuonna 2007 alkanutta lamaa. Sekä pienet että suuret neuroverkot selviävät syklin ennustamisesta pääpiirteissään hyvin, minkä takia LSTM- ja GRU-neuroverkkoja voidaan pitää varteenotettavina vaihtoehtoina taloussykliennustamisessa. Tämän lisäksi, tulokset kannustavat soveltamaan epälineaarisia koneoppimismenetelmiä yhdessä usean muuttujan aikasarja-aineistojen kanssa sellaisiin makrotalouden ennusteongelmiin, joihin ei aikaisemmin ole löydetty tarpeellista tarkkuutta saavuttavaa ratkaisua.

Avainsanat	Taloussykli, ennustaminen, koneoppiminen, neuroverkot
------------	---





**UNIVERSITY
OF TURKU**

Turku School of
Economics

PREDICTING U.S. BUSINESS CYCLES WITH RECURRENT NEURAL NETWORKS

**An Extensive Multivariate Time-series Analysis for Comparing LSTM
and GRU Networks**

Master's Thesis
in Economics

Author:
Markus Mäkelä

Supervisor:
Prof. Jouko Vilmunen

23.6.2020
Turku

The originality of this thesis has been checked in accordance with the University of Turku quality assurance system using the Turnitin OriginalityCheck service.

TABLE OF CONTENTS

1	INTRODUCTION.....	7
2	THEORY REVIEW.....	9
2.1	Time-series.....	9
2.2	U.S. business cycles	11
2.3	Business cycles as a dynamical system.....	12
2.4	Supervised machine learning.....	13
2.5	Bias-variance trade-off	15
2.6	Preprocessing.....	17
2.6.1	Linear interpolation.....	17
2.6.2	Detrending	19
2.6.3	Min-max scaling	19
2.6.4	Train-test split	20
2.7	Deep learning.....	22
2.7.1	Recurrent neural networks	24
2.7.2	Long short-term memory cell	26
2.7.3	Gated recurrent unit	28
2.7.4	Deep neural network architectures	30
2.7.5	Training epoch	30
2.7.6	Forward pass and cost function	32
2.7.7	Backpropagation through time.....	34
2.7.8	Nesterov accelerated gradient.....	36
2.7.9	Learning rate	39
2.7.10	Parameter initialization	40
2.7.11	Training process and early stopping	41
2.7.12	Deep learning hyperparameters	43
3	LITERATURE REVIEW.....	44
3.1	Deep learning for time-series analysis.....	44

3.2	The novel RNN methods' performance in the previous literature.....	45
3.3	Review of the previous related research in finance and economics	48
3.3.1	Related research in finance	48
3.3.2	Related research in economics.....	50
3.4	Conclusions of the previous literature	53
4	METHODS	55
4.1	Data	56
4.2	Preprocessing	64
4.2.1	Detrending and scaling	64
4.2.2	Interpolation and reshaping	65
4.2.3	Train-test split.....	67
4.3	Recurrent neural network selection	68
5	RESULTS	73
5.1	Architectures and the training process.....	73
5.2	Model capacity and performance.....	76
5.3	The performance of the small architectures.....	81
5.4	The third GRU-4 iteration	83
5.5	The large architectures and overfitting	85
5.6	The performance of the midsize architectures.....	89
5.7	The best architecture	91
6	CONCLUSIONS	93
	REFERENCES.....	95
	APPENDICES	104
	Appendix 1. LSTM results	104
	Appendix 2. GRU results.....	110

LIST OF FIGURES

Figure 1 – Dynamical system.....	12
Figure 2 – Related dynamical systems.....	13
Figure 3 – The bias-variance trade-off (Goodfellow et al. 2016, 118, 128)	16
Figure 4 – A depiction of linear interpolation.....	18
Figure 5 – A depiction of a typical relationship between model capacity, training error and testing error	21
Figure 6 – A multilayer perceptron.....	22
Figure 7 – A depiction of a classical recurrent unit	24
Figure 8 – A depiction of a recurrent neural network.....	25
Figure 9 – An unfolded computational graph	25
Figure 10 – A long short-term memory cell.....	27
Figure 11 – A gated recurrent unit	29
Figure 12 – A training loop.....	31
Figure 13 – An error curve comparison between MSE and MAE.....	33
Figure 14 – A gradient descent step for one parameter	36
Figure 15 – A three-dimensional illustration of how a cliff affects a training process (Goodfellow et al. 2016, 285)	37
Figure 16 – A comparison of gradient descent updates with and without momentum (Goodfellow et al. 2016, 293)	38
Figure 17 – The learning rate curve	40
Figure 18 – Error curves for the training process	41
Figure 19 – Forecasting performance (sMAPE) of the ML and statistical methods included in the study (Makridakis et al. 2018, 15).....	45
Figure 20 – Forecasting performance (sMAPE) versus computational complexity (Makridakis et al. 2018, 18).....	46
Figure 21 – Preprocessing phases	64
Figure 22 – A depiction of X and Y.....	65
Figure 23 – A depiction of X and Y examples.....	66
Figure 24 – Combined interpolation and reshaping function.....	66
Figure 25 – The train-test split.....	67
Figure 26 – Recurrent GRU or LSTM neural network architecture	69
Figure 27 – The architectures and the number of parameters.....	71
Figure 28 – Architectures and epochs.....	74

Figure 29 – Errors for single iterations and ensembles.....	76
Figure 30 – Variance per architecture.....	79
Figure 31 – The average estimates for the three smallest LSTM and GRU architectures 82	
Figure 32 – The third GRU-4 iteration	84
Figure 33 – Model capacity and fit.....	85
Figure 34 – Test set MSE over the training epochs the three largest LSTM and GRU architectures	86
Figure 35 – The average estimates for the three largest LSTM and GRU architectures	88
Figure 36 – The average estimates for three midsize LSTM and GRU architectures	90
Figure 37 – LSTM-32-16 predictions	92

LIST OF TABLES

Table 1 – U.S. recessions between 1980 and 2015 (NBER, 2019b).....	11
Table 2 – Abbreviations for the data tables	57
Table 3 – Economic variables.....	57
Table 4 – Financial variables	60
Table 5 – Behavioral variables.....	62
Table 6 – The selected recurrent neural network architectures	70
Table 7 – Architectures and epochs	74
Table 8 – Architectures and MSE for ensembles.....	78
Table 9 – Architectures and MAE for ensembles	78
Table 10 – LSTM-32-16 performance	91

1 INTRODUCTION

Accurate predictions of business cycles could give valuable information for decision making, ranging from individuals' consumption plans to countercyclical macroeconomic policies. However, predicting these fluctuations, especially the timing of recessions, has been proven to be a difficult task (Rudebusch & Williams 2008, 2; Zarnowitz & Braun 1992, 21), making it an interesting area to try to develop more accurate methods.

According to universal approximation theorem by Hornik et al. (1989, 363-364), a feedforward network with at least one hidden layer and nonlinear activation functions, can theoretically represent any function, linear or nonlinear. This unprecedented ability to represent functions that have been difficult with other current methods has made artificial neural networks (ANNs) stand out in problems that demand a very high modeling capacity from the applied model. However, finding a suitable ANN and optimizing its parameters has been found challenging, but lately, together with more enhanced methods, suitable data and computational resources, applying these methods has become more convenient. (Géron 2017, 258; Goodfellow et al. 2016, 280-290.)

Because of these advances related to finding suitable ANNs and optimizing them, together with a large amount of existing economic data, these methods can be recognized as noteworthy tools for economists and especially for making predictions (Varian 2014, 1, 6, 20-21). The predictions and insights that are made by using these sophisticated machine learning methods can be particularly helpful for policymakers in new ways that were not common, or even possible, with standard econometric methods (Basuchoudhary et al. 2017, 1). However, using ANNs in the domain of economics is relatively new, but interesting, because of the linear models' incapability to model many real-world processes that appear to include nonlinearity (Binner et al. 2004, 2).

This study compares different size recurrent neural networks (RNNs) in predicting business cycles for the U.S. The proposed RNNs consist of either long short-term memory cells (LSTMs) or gated recurrent units (GRUs) and they are trained with multivariate data that comprises 27 time-series of several economic, financial and behavioral indicators. The RNNs are configured so that, per example, each model used 26 explanatory variables' observations for the previous 400 days to generate a 91-day forecast of the U.S. business cycle. The results for the comparison are done by using a train-test split, where the models are trained with data from the time period between 1980 and 2005, and evaluated by using the data from the following ten years. Each proposed RNN architecture is

trained three times to counter the randomness and stochasticity that the training algorithm introduces to the parameter optimization process.

The research is conducted so that it answers the following three research questions. Firstly, does the proposed methods suit predicting business cycles. Secondly, how large RNN architecture appears suitable for the task at hand. Thirdly, which method, LSTM or GRU, suits better the prediction problem.

This study examines methods that are relatively little studied in the domain of economics and, simultaneously, they appear to offer a vast number of potential applications. For this reason, this study tries to offer a reference point for the following studies and, hence, it concentrates on applying only common and approved methods. The main focus is on examining the performance-related effects of using different RNN models. The evaluation is done by using typical performance metrics, such as mean squared error (MSE) and mean absolute error (MAE). However, to gain a deeper understanding, the plots of the predictions for the testing set are examined. Appropriate early warning signals are considered beneficial, but evaluating them is recognized somewhat subjective. Thus, the evaluation favors more objective indicators.

The research is organized into five separate sections. The theoretical background is split into two separate chapters. The first chapter concentrates on presenting the necessary theoretical foundations related to the business cycles, time-series analysis and the chosen methods. The theory review should offer the necessary knowledge for understanding the results and the motivations behind the chosen methods. Therefore, in order to serve common economics practitioners, the selected deep learning (DL) methods are introduced carefully. The second chapter introduces the related literature that comprises the previous studies related to time-series analysis, finance and economics. The previous studies indicate the proposed methods' weaknesses and strengths, along with suggesting suitable applications. The literature review also presents a timeline that helps to connect this study to the existing work and further motivates using these novel RNN methods for complex macroeconomic prediction problems. After the related theory and literature are reviewed, the chosen methods are examined in more detail in the fourth chapter. This chapter covers the data examination, the selected preprocessing methods for the data, the training algorithm and the proposed RNN architectures that are used for generating the results. The results are examined in the fifth chapter, that is followed by the conclusions.

2 THEORY REVIEW

2.1 Time-series

In essence, time-series data is a collection of points arranged in chronological order. A time-series can be defined as

$$x_{1,T} = \{x_t | t = 1, \dots, T\},$$

where $x_{1,T}$ is a time-series for a time interval $[1, T]$, x_t is either a scalar or a vector that includes the realizations for time step t and T represents the number of captured realizations. If the realizations x_t for the time interval are scalars, the time-series is one-dimensional, but if they are vectors, the time-series is two-dimensional and can be denoted as a matrix $X_{1,T}$. Continuous processes can be tracked by collecting records of the process over time, creating a discrete time-signal. In time-series analysis, the examined processes are often stochastic, meaning that these collected realizations x_t are generated by some random process that draws these values from some set of all the possible values according to some distribution function. (Lütkepohl 2005, 1-4.)

According to Långkvist et al. (2014, 3-4), time-series data has several unique properties that distinguishes it from other types of data. The following description of these properties follows the structure and content that they used in their study.

Firstly, time-series data usually contains noise that can make it harder to find valuable information from the data. For example, in the area of economics, the daily movements are not usually important when trying to extract information about the macro trends that occur more slowly and are not affected significantly by small frequent short-term movement.

Secondly, the underlying process might be very complex and, therefore, understanding and modeling it tolerably, even by analyzing all the available time-series data, might still be too challenging or even impossible with the best possible techniques available.

Thirdly, time-series data can have the same value at different time steps, meaning either the same or something else depending on, for example, some previous values. Modeling this time-dependency correctly is challenging for many reasons, but also because the length of a sequence for capturing this relationship could be unknown. This, in turn, could make the selection of a suitable model and the related methods difficult.

Lastly, there is stationarity. Since a realization x_t can be captured only once at any given time unit t , it is not possible to identify the mean or variance of all the possible

realizations of x_t per time unit t . For example, it is impossible to observe two or more different realizations of the S&P 500 index at any given time unit. However, the mean and covariance can be calculated for some time-series $x_{1,T}$ by using the realizations over some time interval $[1, T]$. For example, in order to use several popular autoregressive models and their extensions, the time-series data should be weakly or strongly stationary. A time-series is weakly stationary if

$$E(x_t) = \mu \text{ and } Cov(x_t, x_{t+h}) = \gamma_{t,t+h} = \gamma_h,$$

meaning that the expected value μ for a realization x_t is a constant, and the covariance between values x_t and x_{t+h} depends only on their distance between each other, denoted here as h . Strong stationarity is achieved when time-series values' distribution is time-invariant. For example, time-series $x_{1,T}$ is strictly stationary if the joint distribution function is identical for any two different subsets of $x_{1,T}$ that share the same length, such as, $x_{1,5}$ and $x_{10,15}$. (Hamilton 1994, 45-46; Tsay & Chen 2019, 2.)

Time-series analysis is a method for extracting knowledge from time-series data. By performing a time-series analysis, one can better understand the past, but also, use the extracted information to make predictions about the future. (Nielsen 2019, 1.) To make predicting plausible, data from important variables should be available, and it should contain useful information related to the future developments of the chosen variable or variables. With this data, some function $f(\cdot)$ can be found that could be used to make predictions for one time step $t + 1$ or several time steps $[t + 1, t + h]$ ahead. The latter type of method is called sequence-to-sequence predicting, and it can be demonstrated for variable y by using multivariate data X as follows

$$\hat{y}_{t+1,t+h} = f(X_{t-k,t}, \theta),$$

where $\hat{y}_{t+1,t+h}$ denotes a sequence of predictions for an interval $[t + 1, t + h]$, $X_{t-k,t}$ the input matrix containing several variables' sequential data for time steps $[t - k, t]$ and θ the function f parameter values. (Lütkepohl 2005, 1.)

Various different models can be used for modeling the relationships between $X_{t-k,t}$ and $\hat{y}_{t+1,t+h}$, but some of them suit the problem better than the others. Finding a suitable model can be recognized as a model selection problem and, thus, it is related to the area of machine learning (ML). This study applies the typical ML approaches for finding a suitable model for the task at hand, that is modeling the relationships between the past values of the chosen 26 indicators and the future values of the U.S. business cycle.

2.2 U.S. business cycles

Mitchell (1927, 468) describes business cycles as fluctuations in the economic activity that affect the major portion of an organized community, and are not seasonal but rather occasional in their nature. Each cycle includes one upward and downward motion, also called respectively as expansion and recession. Various definitions exist for these concepts and, therefore, different actors might have different views on the economic situation of different economies. For the U.S., the National Bureau of Economic Research's Business Cycle Dating Committee determines the dates when its economy has a recession or expansion. According to NBER (2019), this decision is made by analyzing economic activity in the U.S. broadly and

“A recession is a period between a peak and a trough, and an expansion is a period between a trough and a peak.”

The historical record of the recessions in the U.S. can be found from NBER's website¹, but also from the Federal Reserve Economic Data (FRED) databank². This study comprises the time period between 1980 and 2015. During that time, the U.S. has experienced five recessions, shown in table 1.

Table 1 – U.S. recessions between 1980 and 2015 (NBER, 2019b)

Peak	Trough	Length (months)
January 1980	July 1980	6
July 1981	November 1982	16
July 1990	March 1991	8
March 2001	November 2001	8
December 2007	June 2009	18

Between the years 1980 and 2015, the U.S. has spent most of the time in economic expansion. The economy has spent a total of 56 months in recession and 364 months in expansion. Recessions' share of the time period is approximately 13.334%, making them significantly rarer than the expansions, but not rare enough to be recognized as outliers. Though their scarcity makes the process of learning to predict them difficult, there are no appropriate methods to overcome these disbenefits.

¹ <http://www.nber.org/cycles/cyclesmain.html>

² <https://fred.stlouisfed.org/series/USREC>

2.3 Business cycles as a dynamical system

In economics, a sequence is a subset of real numbers for some time interval I (Giordano et al. 2013, 7). In this domain, the interval under consideration is typically limited to

$$I = [0, +\infty),$$

(Barnett et al. 2015, 1751), where 0 denotes the first realization available. A dynamical system is a process that generates realizations over time that can be stored in a sequence to form a time-series. By using the known realizations, it is possible to describe the change from one time step to the next by estimating some function f (Giordano et al. 2013, 5-7).

The economy can be seen as a dynamical system that evolves in time, generating realizations whose fluctuations can be defined as business cycles. The classical dynamical system can be described as follows

$$s_t^{(1)} = f(s_{t-1}^{(1)}, \theta^{(1)}),$$

where $s_t^{(1)}$ resembles the state of the system at discrete time step t , and it is defined by some function f , previous state $s_{t-1}^{(1)}$ and some set of parameters $\theta^{(1)}$. One important aspect of this process is that it is recurrent, meaning that the state $s_t^{(1)}$ is dependent on its previous states, as depicted in figure 1. (Goodfellow et al. 2016, 369.) In addition, the formula can be decomposed to show the recurrence:

$$s_t^{(1)} = f(f(f(s_{t-3}^{(1)}, \theta^{(1)}), \theta^{(1)}), \theta^{(1)}).$$

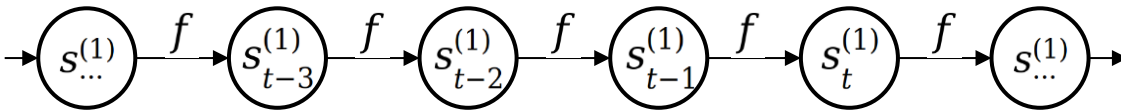


Figure 1 – Dynamical system

This type of dynamical system's change over time can also be affected by some external dynamical system $s_t^{(2)}$, that is unique, but has similar properties to the first dynamical system $s_t^{(1)}$. The function for the first dynamical system can be now written as

$$s_t^{(1)} = f(s_{t-1}^{(1)}, s_t^{(2)}, \theta^{(1)}),$$

where the first dynamical system's state at time step t depends on its previous state $s_{t-1}^{(1)}$, some external system's current state $s_t^{(2)}$ and some set of parameters $\theta^{(1)}$, as depicted in figure 2. In this type of situation, one should also understand what is the second dynamical system's effect on the first dynamical system.

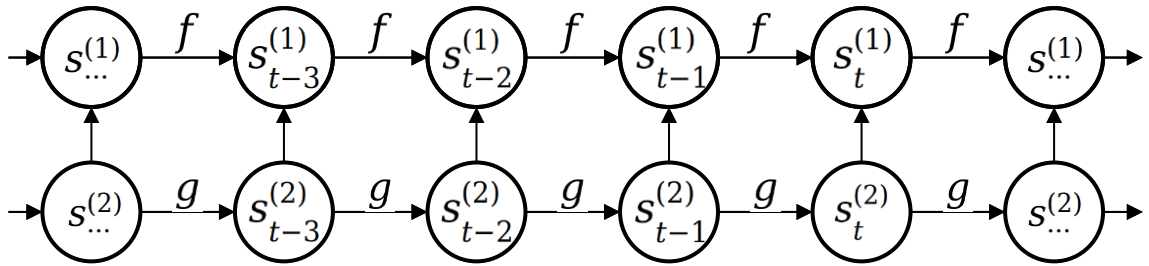


Figure 2 – Related dynamical systems

As mentioned earlier, these types of systems can be tracked by collecting realizations over time. The resulting time-series allow creating models that try to replicate the rules that define how the dynamical system evolves in time. As mentioned previously, the economy can be recognized as a dynamical process that affects and is affected by several different other dynamical systems. Because of these attributes, it is justified to use multivariate data that includes some set of so-called leading indicators, that comprises information of several dynamical systems, for predicting business cycles (Qi 2001, 383-384).

2.4 Supervised machine learning

Machine learning (ML) is defined by Samuel (1959, 1) as a way to program a digital computer so it can be thought to be able to learn, or by Goodfellow et al. (2016, 96) as a study of designing algorithms that can learn from data.

ML has a close relationship with other common data analyzing and modeling tools since they share several methods and tools. According to Varian (2014, 5-6), in statistics and econometrics, the data analysis can be broken into four categories: 1) prediction, 2) summarization, 3) estimation and 4) hypothesis testing. He suggests that the major difference between ML, common statistics and econometrics is that where statisticians and econometricians focus primarily on insights and relationships that can be found from the data, ML concentrates mostly on making accurate predictions. Thus, ML should be considered when the focus is on predicting.

Machine learning can be divided into two main classes, that are a predictive and descriptive approach (Murphy 2012, 2). The predictive ML algorithms can be used for predicting some missing information by using some known information, and the descriptive ML algorithms are typically used for finding and describing patterns in data. The algorithms for the predictive tasks are commonly trained by using supervised learning methods and the latter by using unsupervised learning methods. It is also good to acknowledge that there are many other ways to classify different types of machine learning, but they

are out of the scope of this study, which only applies supervised learning algorithms for business cycle forecasting.

In supervised learning, algorithms are provided with input-output pairs. (Hastie et al. 2009, 29; Goodfellow et al. 2016, 103.) Inputs can be defined, for example, as a matrix $X \in \mathbb{R}^{i \times j}$ where i represents the number of examples or rows in the data and j represents the number of explanatory variables or features. In turn, similarly to the inputs, also the outputs can be defined by using some common data object, such as a matrix $Y \in \mathbb{R}^{i \times k}$ or a vector $y \in \mathbb{R}^{i \times 1}$, depending on the number of dependent variables k .

A supervised machine learning prediction problem can be described with the following equation:

$$f(x^{(i)}, \theta) = \hat{y}^{(i)}.$$

Here the vector $x^{(i)}$ represents the i^{th} example, that is on row i in the data matrix X . $f(\cdot)$ represents some function, θ the function's parameters and $\hat{y}^{(i)}$ the outputs for the i^{th} inputs. In supervised ML, the task is typically to find some function with some set of parameters that is able to achieve satisfying accuracy at mapping the known input values into estimated output values that are as close as possible to the real output values. (Varian 2014, 6; Goodfellow et al. 2016, 103-105.)

In machine learning, it is recognized that there are plenty of different types of models that suit different types of tasks and data. Therefore, much of the ML theories and methods concern finding a suitable model for a given problem and data. For example, classification, regression and clustering tasks have their own typical models and other data related techniques, but behind it all, there lies a fundamental theory that motivates questioning the current common practices and testing new methods. According to Wolpert (1996) and Wolpert and Macready (1997) papers' mathematical demonstrations, if absolutely no assumption about the data or the task is made, no model or algorithm is better than all the other possible alternatives in every different task. Therefore, in this situation, one should evaluate all the possible options, also called the hypothesis space, in order to find the solution that fits the given problem the best. These theorems are called the No Free Lunch theorems (NFL-theorems), and they are recognized as one of the most important theories in the field of ML.

However, Géron (2017, 31) notes that, in practice, evaluating all the different possible models is impractical, or even impossible with the scarce resources and, hence, some assumptions about the data and the task should be made in order to narrow down the set

of algorithms to evaluate. Therefore, even though there lies a significant motivation for trying out all the known models and methods when creating systems that can learn, the previous work in the field has a significant role in the human and computing resource allocation.

2.5 Bias-variance trade-off

Bias, variance and their trade-off are fundamental at understanding the essential concepts of machine learning, such as under- and overfitting, generalization error and model capacity, which build the framework for finding a suitable model for some prediction task. Typically, error in an estimator can be divided into two components: bias and variance. Bias addresses the amount of error that occurs from the prediction's expected deviation from the real value, and variance tells how much error is generated from the deviation of the expected prediction from the real prediction. (Goodfellow et al. 2016, 120-128; Ge-man et al. 1992, 2.)

In this study, the mean squared error (MSE) is the most important error measure because it is used in parameter optimization and model evaluation. By following Murphy's (2012, 202) notations, we can derive the expected bias and variance for the expected MSE as follows

$$\begin{aligned}
 MSE &= \mathbb{E}[(\hat{y} - y^*)^2], \\
 MSE &= \mathbb{E}\left[\left((\hat{y} - \bar{y}) + (\bar{y} - y^*)\right)^2\right], \\
 MSE &= \mathbb{E}[(\hat{y} - \bar{y})^2] + 2(\bar{y} - y^*)\mathbb{E}[\hat{y} - \bar{y}] + (\bar{y} - y^*)^2, \\
 MSE &= \mathbb{E}[(\hat{y} - \bar{y})^2] + (\bar{y} - y^*)^2, \\
 MSE &= \text{var}(\hat{y}) + \text{bias}^2(\hat{y}), \\
 MSE &= \text{variance} + \text{bias}^2.
 \end{aligned}$$

Here, \hat{y} stands for the prediction, y^* for the real value, and \bar{y} for the expected prediction for a given input. The expected MSE and the expected prediction \bar{y} can be discovered by testing the model repeatedly by using a large number of training data and then averaging the result (James et al. 2013, 34).

From the decomposition, one can see that, in order to minimize MSE, both variance and bias should be decreased. However, they are connected to model capacity, as depicted in figure 3.

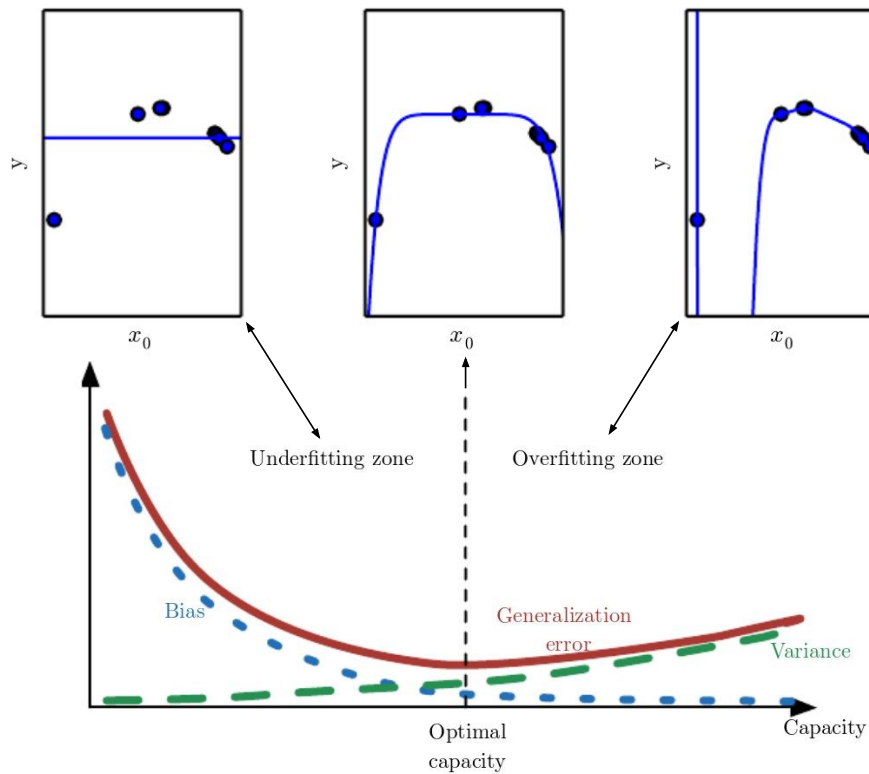


Figure 3 – The bias-variance trade-off (Goodfellow et al. 2016, 118, 128)

Generally, the models with too low modeling capacity for modeling some data tend to have high bias and low variance, and in turn, the models that possess too much modeling capacity tend to have small bias but high variance. This happens because the more capable models are able to model the random noise in the data and become too sensitive to small changes in the input data and, thus, gain more variance when tested with unseen data. This phenomenon is also called overfitting. In turn, underfitting occurs when using insufficient models that are too constrained to find a suitable fit for the data. These two phenomena are interlinked by the modeling capacity, and they form a theory called the bias-variance trade-off. (Goodfellow et al. 2016, 127-128; James et al. 2013, 33-36.)

An estimate of a model's true performance, also called the generalization error, can be obtained by testing the model with examples that are not used for tuning its parameters. This estimation is usually done by dividing the available data into two or more folds that are then used either to train or to test the model. (Goodfellow et al. 2016, 108-111.) In this study, the generalization error is estimated by dividing the whole data into two folds where one is used for model training and the other for estimating the generalization error. The methods used in this study for dividing the data into training and testing sets are examined in more detail in section 2.6.4.

2.6 Preprocessing

A dataset is seldom instantaneously ready to use for ML tasks. Instead, it is usually pre-processed so that it suits better the chosen models and parameter optimization methods. In fact, finding good quality data and preprocessing it accordingly is found as one of the most important areas in ML. (Nielsen 2019, 17; Géron 2017, 26.)

In this study, the motivation for the chosen preprocessing methods comes from three sources. Firstly, deep learning solutions prefer particular types of properties from the data. Secondly, time-series data has its own ways when it comes to preprocessing. Thirdly, the preprocessing should be done in a way that is also possible in the production environment. Essentially, this means that the preprocessing methods should not use information from the time steps that are not known at some particular moment for some particular instance. In this study, for convenience, only the future information is cropped when preprocessing. It does not take into account the lag between the date when a realization occurs and the date when it is actually released.

The time-series data used in this study had several imperfections, such as missing values, incoherent value scales and trends. These defects have been handled by using common preprocessing methods that are introduced in the following sections.

2.6.1 Linear interpolation

The data used in this study suffers from missing values that mostly originate from low-frequency time-series. In order to obtain more data for the data-hungry deep learning methods, daily data is used. Hence, all the time-series with a lower frequency demand upsampling.

Interpolation is a common method for handling missing values and upsampling. It approximates missing values in a time-series by using their neighboring data points. In this study, a linear interpolation is used, whose equation is

$$y_{(t)} = y_{(t-1)} + \frac{(x_{(t)} - x_{(t-1)})(y_{(t+1)} - y_{(t-1)})}{x_{(t+1)} - x_{(t-1)}},$$

where $(x_{(t-1)}, y_{(t-1)})$ and $(x_{(t+1)}, y_{(t+1)})$ are the known data points that are used for finding the missing coordinate $x_{(t)}$ or $y_{(t)}$. Linear interpolation can also be used to fill a chain of missing values, for example, when upsampling quarterly data into daily data. Figuratively, it draws a line between two known realizations and the missing values in that range are filled by using this particular line, as depicted in figure 4.

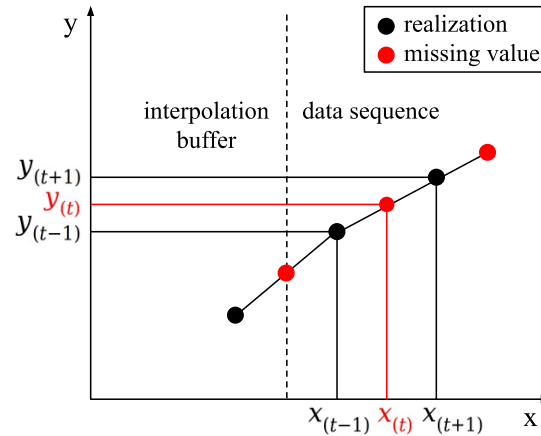


Figure 4 – A depiction of linear interpolation

If there are no realizations before or after the missing value, the adjacent slope can be used. In this study, the previous slope is used for the last missing values that would not otherwise receive a value. As depicted in figure 4, the missing value for time step $t + 2$ would be set by using the slope determined by the realizations for time steps $t - 1$ and $t + 1$. In turn, when dealing with the first missing values, this same slope could be used. However, in this study, the previous realizations are used for determining these missing values in order to get more accurate time-series. Thus, for example, the realizations for time steps $t - 3$ and $t - 1$ are used for determining the value for the missing realization on time step $t - 2$. Here, the $t - 3$ realization is drawn from outside of the data sequence under consideration, an area called interpolation buffer. This method should not be used for the last missing values if one wants to simulate a situation where there is no information about future values and prevent the aforementioned lookahead effect.

Linear interpolation has some defects. Firstly, the realizations of a dynamical system seldom follow the aforementioned equation. Hence, a time-series that is impaired of missing values cannot be fully repaired by using linear interpolation and, therefore, time-series with fewer missing values should be preferred. Secondly, the adjacent data points' ability to provide valuable information about the missing value or many values between them might be weak, for example, when the signal is very similar to a random walk or is otherwise very volatile. Thirdly, if used incorrectly, it might also provide lookahead information to the data. (Nielsen 2019, 48-50.)

Even though of these defects, interpolation is a useful method for handling the missing values since it enables the use of some valuable time-series. Moreover, the trends in macroeconomic data tend to be medium and long-term movements, making the short-term variations less significant and, therefore, allowing more short-term imprecision from

the data that, in turn, supports using fairly simple methods for handling the missing values in this study.

2.6.2 Detrending

A trend in the context of time-series data is typically a systematic change that is not seasonal. A trend in a time-series can be detected, for example, by examining its plot or by testing its stationarity. In economic time-series, trends are typical because several economic processes' realizations depend on their previous values. Zhang and Qi (2002; 2005) showed that if a time-series contains trends and seasonal variations, both should be pre-processed before feeding it to an ANN model. Furthermore, Qi and Zhang (2008) demonstrated that differencing should be used for detrending a time-series.

Because ANN models can deal with nonstationary data, stationarity tests are not implemented in this study, and differencing is used for handling only the time-series with notable trends that can be detected by exploring their plots. The differencing is done by using the previous time step value. No deseasonalization is implemented to the time-series, but several already deseasonalized time-series are used.

2.6.3 Min-max scaling

When using deep learning, the data is typically scaled between 0 and 1 (Bengio 2012, 447). Because many scaling or normalization methods use minimum and maximum values, they are vulnerable to trends and outliers. If a trend continues after the parameters for the scaling function are set, it is likely that the future data can cross the boundaries if the scaling parameters are not adjusted. The notable trends are handled in this study and, therefore, the common min-max scaling equation:

$$x_{scaled} = (x - x_{min}) / (x_{max} - x_{min}),$$

should be feasible for scaling the time-series data. In min-max scaling, some value from a time-series x , or the whole time-series, is scaled by using its the smallest x_{min} and highest x_{max} value. Each time-series is scaled separately.

Although this method is popular, it is vulnerable to outliers because it utilizes the minimum and maximum values. The data used in this study is relatively smooth, and it does not seem to include significant changes that could not be explained. Moreover, for this particular application, some rare values typically have a reasonable explanation and, therefore, might be necessary when predicting business cycles.

2.6.4 Train-test split

To estimate the generalization error of some model, which is the expected value of some chosen error measure, such as MSE, when averaged over the future data, one can use a common method called a train-test split. It is applied by dividing a data set once into two separate sets: the training set, that is used for optimizing the models' parameters, and the testing set, that, in turn, is used for testing the models' performance with unseen data examples, hence yielding an estimate of the models' generalization error. (Goodfellow et al. 2016, 108-111; Murphy 2012, 23.) One crucial aspect of this method is to define the training and testing sets in a way that simultaneously maximizes the models' performance and the accuracy of the generalization error estimate. Furthermore, when defining the split, a trade-off should be acknowledged. Since, generally, the more data is used for optimizing the model, the better the model performs, and in turn, the more data is used for evaluating the model, the better estimate of the generalization error is achieved. Hence, when using train-test split, it tends to give pessimistic estimates of the generalization error, compared to what the models could achieve by using the whole available data. Moreover, the sets should be somewhat similar to each other. The training set should contain the information, the so-called general pattern, to be learned, and the testing set should test how well the models learned to model this pattern and, hence, it needs to contain a sufficient number of examples about the general pattern. (Goodfellow et al. 2016, 108-111.)

When dealing with time-series data, both, the training and testing set, should contain a continuous sequence of consecutive examples that form a continuous time window. Furthermore, when trying to predict the future by using past information, the sequence used for testing should contain the newer data points, and the training set the older data points, in order to simulate the real-life use case. (Nielsen 2019, 343.)

As introduced in the NFL-theorems, there is no universally superior model for every possible prediction problem. Instead, any model from the vast hypothesis space should be considered as a possible option if no assumptions are made concerning the task at hand. A model suitable is commonly found based on the bias-variance trade-off framework, together with some performance measures that guide the decisions concerning the model capacity. Generally, a complex problem demands a model that is able to model complex relationships and vice versa (Goodfellow et al. 2016, 112-113).

As is introduced earlier, the evaluation of the generalization error is fundamental for the model selection process. Now the generalization error can be estimated to some degree by using a train-test split, as shown in figure 5. If the sets for training and testing are ideal, the error obtained by testing the model with the testing set's data will represent the generalization error. Hence, when the model underfits or overfits the data, the testing error should increase due to the bias-variance trade-off. A model with less than the optimal modeling capacity has trouble to capture the general pattern adequately. Though these models cannot offer the best performance, they might be useful for providing information about the task at hand, possibly about the most important relationships. When the model capacity is increased over the optimal value, the model can adjust itself better to the training set specific noise and other irregularities that weakens its ability to perform well with unseen examples in the training set. (Goodfellow et al. 2016, 108-114, 127-128; James et al. 2013, 33-36.)

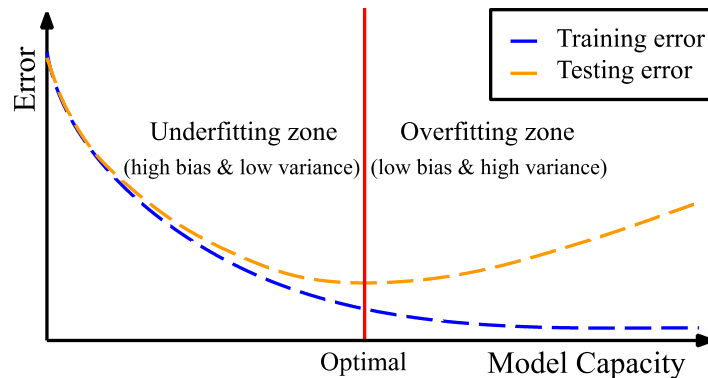


Figure 5 – A depiction of a typical relationship between model capacity, training error and testing error

The train-test split is done in this study by dividing the data into two separate time windows where the latest data is used for testing and the rest for training a selection of artificial neural network models. In order to make the testing set adequate, it should include the general pattern, here the business cycle. Hence, the whole time-series data is divided so that the testing set includes the latest U.S. business cycle between 2005 and 2015. The chosen train-test split together with all the selected time-series are introduced in more detail in sections 4.1. and 4.2.3.

2.7 Deep learning

An artificial neural network (ANN) with more than one layer between the input and output layers is considered deep. These types of ANNs are called deep neural networks (DNNs) and all the methods concerning finding a suitable DNN and its parameter values more broadly as deep learning (DL) (Goodfellow et al. 2016, 8-18). Hornik et al. (1989) proved that a classical ANN model, a multilayer perceptron (MLP) that possesses a sufficient number of units with nonlinear activations function can approximate any continuous function at an arbitrary level of accuracy. For this reason, ANNs are regarded as universal function approximators. Since their very high capacity to model complicated patterns in data, DL algorithms have enjoyed success in extremely complex domains, such as, image and speech recognition that were not conquered by any other models. (Goodfellow et al. 2016, 152, 225.)

In deep learning, the model consists of numerous individual arithmetic operations that together form a bigger entity, a network of small functions that is optimized, for example, to map inputs to outputs if the task is a supervised learning problem. Before introducing the more advanced ANNs that are used in this study, a multilayer perceptron (MLP) is explained, which serves as a background. A common way to examine ANNs is by studying their architecture from a computational graph or some other depiction. In this way, the computations might be easier to be perceived and the network's main traits to be evaluated (Goodfellow et al. 2016, 201).

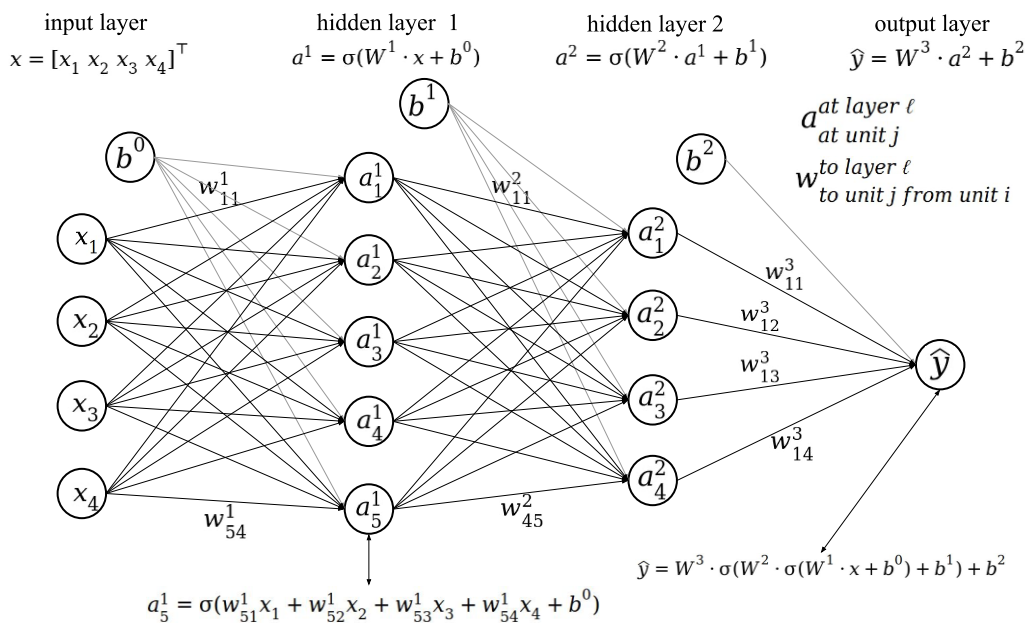


Figure 6 – A multilayer perceptron

Figure 6 depicts an MLP with four separate layers. The model receives an input vector x that is then transformed into an estimate \hat{y} . The two hidden layers in the model will ultimately create two different representations of the original inputs by doing several simple arithmetic operations that they feed forward to the next layer along with the related weights. For example, an activation for unit a_5^1 is calculated by taking a weighted sum of the previous layer's activations and their relative weights for the examined unit that is then added to the bias term b^0 , following some activation function to introduce nonlinearity. Several different functions can be used as an activation function, such as sigmoid σ or hyperbolic tangent τ . The activation a_5^1 can now be written in a vector notation by using dot product as follows

$$a_5^1 = \sigma((w_5^1)^\top x + b^0).$$

As depicted in figure 6, the activations for layer L are calculated with the previous activations a^{L-1} or inputs x , the matrix W^L that consists of the relative transposed weight vectors w_j^L for each separate unit j in the layer L , the bias term b^{L-1} and some activation function as follows

$$a^L = \sigma(W^L a^{L-1} + b^{L-1}).$$

The examined MLP in figure 6 consists of one input layer, two hidden layers and one output layer. These layer operations formulate a chain of functions that yields the output \hat{y} in the following way

$$f(x, \theta) = W^3 \sigma(W^2 \sigma(W^1 x + b^0) + b^1) + b^2 = \hat{y},$$

where θ consists of all the parameters in the model, which are the weights W^L and the bias terms b^L . In this example, the output layer does not apply an activation function. (Goodfellow et al. 2016, 164-172; Graves 2012, 12-16.)

To control the model capacity in DL, one has to adjust the architecture and possible regularization terms. Generally, the more a model has parameters and less regulation, the higher is its ability to model complex patterns and the more prone it is to overfit to the provided data. The depth of the examined MLP model is three. Essentially, all layers other than the input layer are counted. The width of a layer is determined by the number of units it has. Several studies have found that typically larger architectures tend to be the most successful if they are regularized properly, especially the networks with higher depth. (Bengio 2012, 450-451; Goodfellow et al. 2016, 165, 193-197.) Although the universal approximation theorem shows that an ANN with just one hidden layer and a sufficient number of units can represent any continuous function, deeper models are favored.

Typically, depth gives more modeling capacity with fewer parameters and, thus, requires less computational resources, albeit exposing the training process to more faults, such as exploding and vanishing gradients. (Goodfellow et al. 2016, 193-197, 285-287.)

This MLP model is also classified as a deep feedforward network that comes from the nature of how the activations are propagated through the network, which is forward layer-by-layer. In addition to forward propagating networks, there are also networks with connections that allow them to feed activations back into the network, called recurrent neural networks. (Goodfellow et al. 2016, 164.)

2.7.1 Recurrent neural networks

Recurrent neural networks (RNNs) are a class of ANN models that include recurrent connections in their architecture. Feeding information back into the network builds a memory-like function to RNN models, which makes them more suitable for some sequential tasks than the typical feedforward networks. (Goodfellow et al. 2016, 367; Graves 2012, 18-19.)

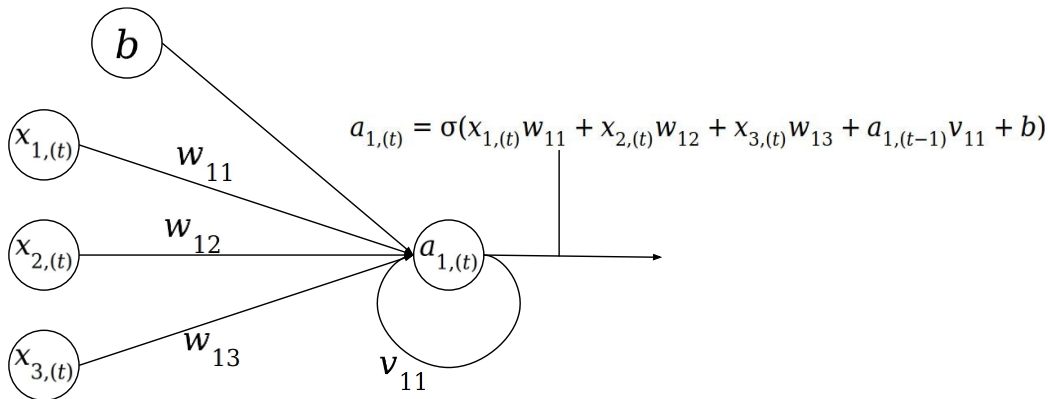


Figure 7 – A depiction of a classical recurrent unit

Similarly to the normal hidden units that were introduced in the MLP example, a recurrent unit a_1 receives a bias term b and inputs x at time step t that are multiplied with their relative weights, as depicted in figure 7. The addition is the recurrent connection that multiplies some activation at the previous time step, that is here its own previous activation $a_{1,(t-1)}$ with its relative weight v_{11} . After these arithmetic operations, an activation function σ is introduced to add nonlinearity to the recurrent unit's activation. (Goodfellow et al. 2016, 369-376; Graves 2012, 18-21.) The function for this classical recurrent unit at time step t can be written by using a vector notation as follows

$$a_{1,(t)} = \sigma(w^\top x_{(t)} + v^\top a_{(t-1)} + b).$$

There are a vast number of different options on how the network could be designed by defining the number of layers (depth), choosing the number of units per layer (width) and the connections between them. The following simple example of an RNN model follows Goodfellow et al. (2016, 372-376) outlines.

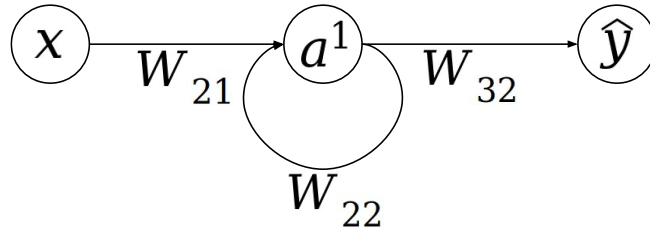


Figure 8 – A depiction of a recurrent neural network

An RNN with one hidden layer comprises three weight matrices W_{21} , W_{22} and W_{32} , as depicted in figure 8. The bias terms can be regarded as input nodes that feed forward a value of one that is multiplied by a weight parameter and, thus, can be included in the weight matrices to simplify the notations. During a forward pass, the RNN generates a prediction \hat{y} for time step t according to the following equation:

$$\hat{y}_{(t)} = W_{32}\sigma(W_{21}x_{(t)} + W_{22}a_{(t-1)}^1),$$

which resembles the MLP model apart from the recurrent connections $W_{22}a_{(t-1)}^1$.

To illustrate how an RNN operates with sequential data, the computational graph can be unfolded with respect to time steps so that the recurrence stands out better, as depicted in figure 9. The model is given the sequential input data one time step at a time, and with every input, the model is able to produce an output for that particular time step.

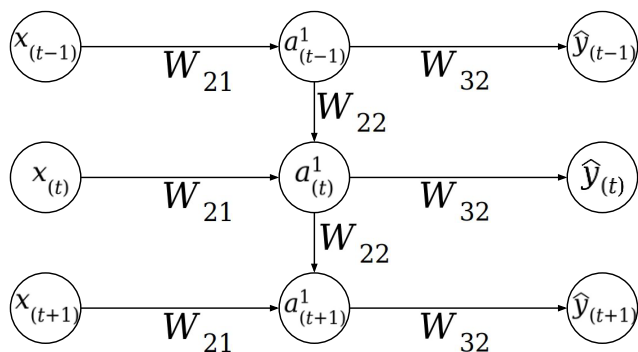


Figure 9 – An unfolded computational graph

RNNs are able to use past information to make predictions, but RNNs that consist of only the classical recurrent units perform poorly when modeling long-term dependencies in sequential data (Bengio et al. 1994). This is due to the common problems in deep learning,

called vanishing and exploding gradients that might occur when backpropagating the cost to the model parameters (Goodfellow et al. 2016, 390). This defect makes it oftentimes unideal to use this type of RNN for tasks that demand good handling of long-term dependencies, and for this reason, the classical recurrent units were introduced only as an introduction to the RNNs and are not used otherwise in this study. Instead, more sophisticated methods are chosen that are able to handle better sequences with long-term dependencies called long short-term memory cells (LSTMs) and gated recurrent units (GRUs). These gated RNN methods exploit the so-called paths through time that make their training process more robust against the aforementioned issues (Goodfellow et al. 2016, 404). According to the results by Greff et al. (2017) and Jozefowicz et al. (2015) studies, these two aforementioned gated RNN methods seem to perform quite evenly along with few other similar designs in different types of tasks that use sequential data.

2.7.2 Long short-term memory cell

The first version of LSTM was introduced by Hochreiter and Schmidhuber (1997) to create a method that could overcome the previous RNNs' problems related to modeling long-term dependency. According to Olah (2015), the later versions of the LSTM and its other modifications are widely used in different problems that demand good performance in handling data with long-term dependencies. The following example of an LSTM cell follows the outlines made by Goodfellow et al. (2016, 404-406), Graves (2012, 31-38) and Olah (2015).

An LSTM cell, depicted in figure 10, receives some inputs and produces some outputs for the next layer and for itself for the next time step, similarly to the previously examined classical recurrent unit. The difference is the architecture of arithmetic operations inside the LSTM cell, along with two different outputs, the activation a and the cell state c . LSTM cell's core idea is to update the cell state every time step, and then use it to create a cell activation in accordance with some inputs and parameter values. It is this cell state, also called the cell memory, that creates the path through time, and makes the LSTM more robust against the aforementioned unideal backpropagation behavior.

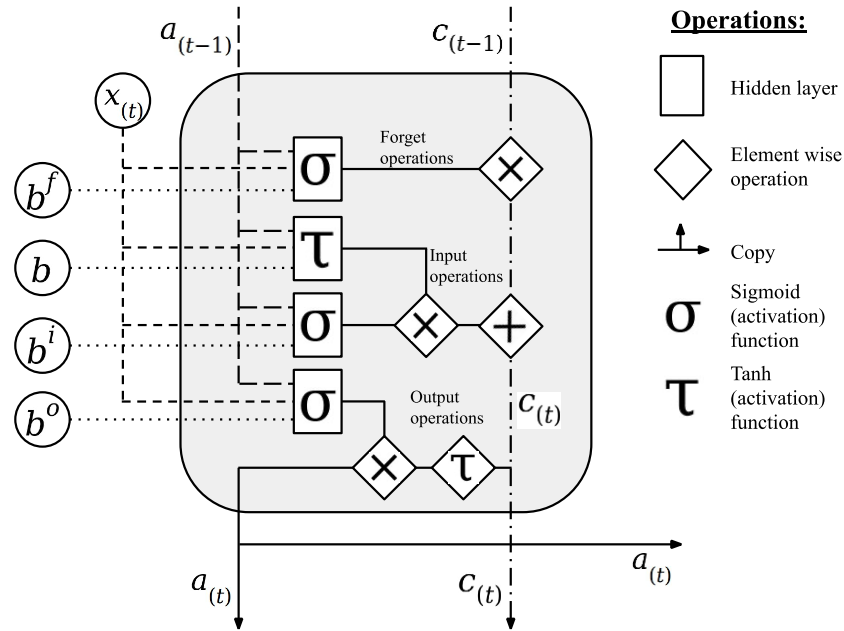


Figure 10 – A long short-term memory cell

The LSTM cell's arithmetic operations can be seen to construct three gates that control the flow of information in the cell and also out of it. Two of the three gates are dedicated to forget and add information into the cell state as follows

$$c_{(t)} = g_{(t)}^f c_{(t-1)} + g_{(t)}^i i_{(t)},$$

where $c_{(t)}$ stands for the cell state at time step t , $g_{(t)}^f$ for the forget gate, $c_{(t-1)}$ for the previous cell state at time step $t - 1$, $g_{(t)}^i$ for the input gate and $i_{(t)}$ for the input, that is now a combination of the previous layer's current activations, here denoted as $x_{(t)}$, and the current layer's previous activations $a_{(t-1)}$. $g_{(t)}^f$, $g_{(t)}^i$ and $i_{(t)}$ are calculated with operations similar to the classical recurrent unit:

$$g_{(t)}^f = \sigma(W^f x_{(t)} + V^f a_{(t-1)} + b^f),$$

$$g_{(t)}^i = \sigma(W^i x_{(t)} + V^i a_{(t-1)} + b^i),$$

$$i_{(t)} = \tau(W x_{(t)} + V a_{(t-1)} + b).$$

The operation that handles “the forgetting” in the cell is the multiplication between the forget gate $g_{(t)}^f$, that receives a value between 0 and 1 due to the sigmoid activation function σ , and the cell state on the previous time step $c_{(t-1)}$. The forget gate should learn when it needs to decrease the cell state and by how much, by finding the right parameter values W^f , V^f and b^f . Similarly, the input gate $g_{(t)}^i$ and the inputs $i_{(t)}$ should learn when and what they should add to the cell state, by finding the optimal parameters W^i , W , V^i , V , b^i and b .

In addition to these operations that concentrate on evolving the cell state, there are operations that determine the actual activation that the LSTM cell outputs at time step t . These operations are hyperbolic tangent function τ and the output gate $g_{(t)}^o$. The activation is drawn from the cell state $c_{(t)}$ as follows

$$g_{(t)}^o = \sigma(W^o x_{(t)} + V^o a_{(t-1)} + b^o),$$

$$a_{(t)} = g_{(t)}^o \tau(c_{(t)}).$$

Similarly to the previous gates, the output gate $g_{(t)}^o$ should learn how much it should decrease the mapped cell state at a given time step. However, before applying the output gate, the hyperbolic tangent function maps the cell state between -1 and 1.

An RNN that uses only LSTM cells is called an LSTM network. When compared to the depiction of the RNN in figures 8 and 9, the LSTM cells can be thought to replace the classical recurrent units while maintaining the same layer-wise connections. As can be derived from the LSTM cell's arithmetic operations, an LSTM network introduces lots of parameters and arithmetic operations. Since computational resources and digital memory are scarce, there has been an incentive to create lighter versions of the traditional LSTM cell, and several modifications have been introduced, one being a gated recurrent unit (GRU).

2.7.3 Gated recurrent unit

Cho et al. (2014) created an LSTM variant called a gated recurrent unit (GRU) that requires fewer computations but is able to perform similarly in tasks that require modeling long-term dependencies in data. In comparison to an LSTM cell's architecture, a GRU has only two gates for controlling the information flow inside the unit, whereas an LSTM cell has three. Therefore, it also requires fewer parameters, since it has only three embedded hidden layers. Also, a GRU does not have a separate cell state, as shown in figure 11. Instead, it uses the activations from the previous time step $a_{(t-1)}$, and the previous layer's activations at the current time step $x_{(t)}$ to produce a new activation $a_{(t)}$, according to its arithmetic architecture and parameter values.

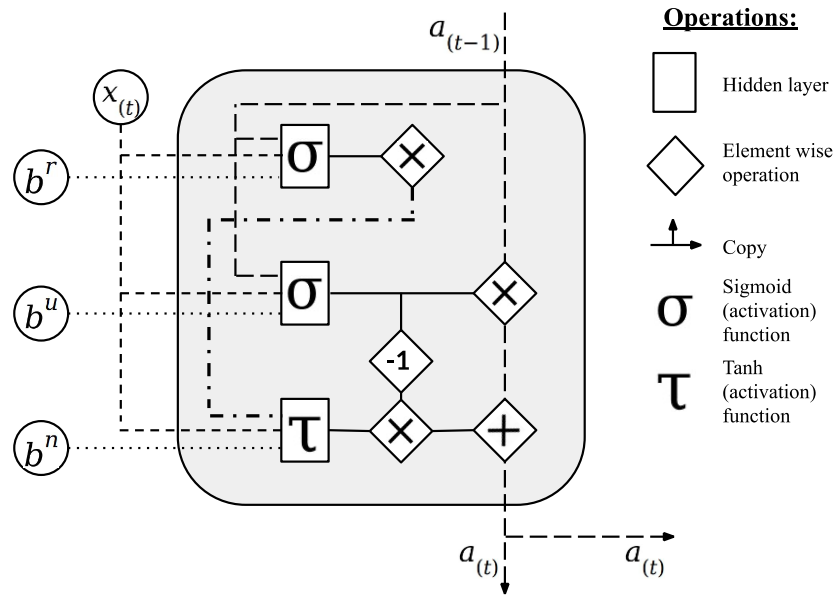


Figure 11 – A gated recurrent unit

The activation at time step t for a GRU is calculated by the following equation:

$$a_{(t)} = g_{(t)}^u a_{(t-1)} + (1 - g_{(t)}^u) a_{(t)}^n,$$

where the new activation $a_{(t)}^n$ is obtained as follows

$$a_{(t)}^n = \tau(W^n x_{(t)} + V^n (g_{(t)}^r a_{(t-1)}) + b^n).$$

The functions for the reset and update gate are the following:

$$g_{(t)}^r = \sigma(W^r x_{(t)} + V^r a_{(t-1)} + b^r),$$

$$g_{(t)}^u = \sigma(W^u x_{(t)} + V^u a_{(t-1)} + b^u).$$

Similarly to an LSTM cell's gates, the reset gate $g_{(t)}^r$ and the update gate $g_{(t)}^u$ should learn when and by how much they decrease the information flow in the unit by discovering suitable values for their weights and biases. The new activation $a_{(t)}^n$ is obtained by gating the previous activations $a_{(t-1)}$ through the reset gate $g_{(t)}^r$ and then using the result as an input alongside with activations from the previous layer, denoted here as $x_{(t)}$. Together with the parameters W^n , V^n and b^n , they form an affine function whose output is then mapped between -1 and 1 by using the hyperbolic tangent τ as a nonlinear activation function. The activation for the GRU is combined from the previous activation $a_{(t-1)}$ and the new activation $a_{(t)}^n$ in a ratio that is determined by the update gate $g_{(t)}^u$. (Cho et al. 2014, 3; Goodfellow et al. 2016, 407-408.)

2.7.4 Deep neural network architectures

The more an ANN has learnable parameters, the more it tends to offer modeling capacity, but also its architecture has a significant effect on its ability to express patterns in data. Arguably, the most significant effect comes from multiple hidden layers that are connected to each other. (Goodfellow et al. 2016, 165, 193-197.) In theory, the layers closer to the input layer concentrate on finding lower-level details from the data, and the following layers are able to use these concepts and build higher-level concepts that are relevant for solving the task and decreasing the cost (Goodfellow et al. 2016, 195-197). Although Hornik et al. (1989) have shown that an ANN with only one hidden layer can approximate any continuous function, it is seldom used when applied to complex tasks, because ANNs with more than one hidden layer typically offer better performance with fewer parameters. (Delalleau & Bengio 2011; LeCun et al. 2015.)

Several studies, such as Graves et al. (2013), found RNNs with multiple hidden layers offering significant performance gains compared to other established methods at the time. Cho et al. (2014, 3) argue that an LSTM or GRU network with only one hidden layer is able to learn various dependencies over different time scales in the data because of their unique gates. However, Pascanu et al. (2014, 1) suggest that this effect is further strengthened via additional hidden layers that are able to create higher-level concepts from the previous layers' activations.

Based on these findings, testing deep recurrent neural networks for the task at hand is reasonable, although they introduce a higher risk for exploding and vanishing gradients. As these risks are acknowledged, the training algorithm for optimizing the selected LSTM and GRU networks includes methods that should make the training process more stable, such as Nesterov accelerated momentum.

2.7.5 Training epoch

In the context of ML, the parameter optimization is responsible for the phenomenon called learning. Basically, the goal for the parameter optimization in supervised deep learning is to find a set of parameters θ that minimizes the value determined by some cost function $J(\cdot)$. Optimizing an ANN's parameters is a nonconvex optimization problem since the model itself is nonconvex. Therefore, it does not possess many of the helpful attributes for optimization compared to convex models. Arguably, the most significant difference is that an ANN usually has several local minima. For this reason, it is possible,

and quite likely, that the parameter optimization does not find the global minimum on the cost function. However, this has not necessarily been a deal-breaker in various applications because stochastic gradient descent (SGD) and its modifications tend to find parameter values that yield a low enough value with a reasonable number of computations. (Goodfellow et al. 2016, 149-150, 271-272, 279-285.)

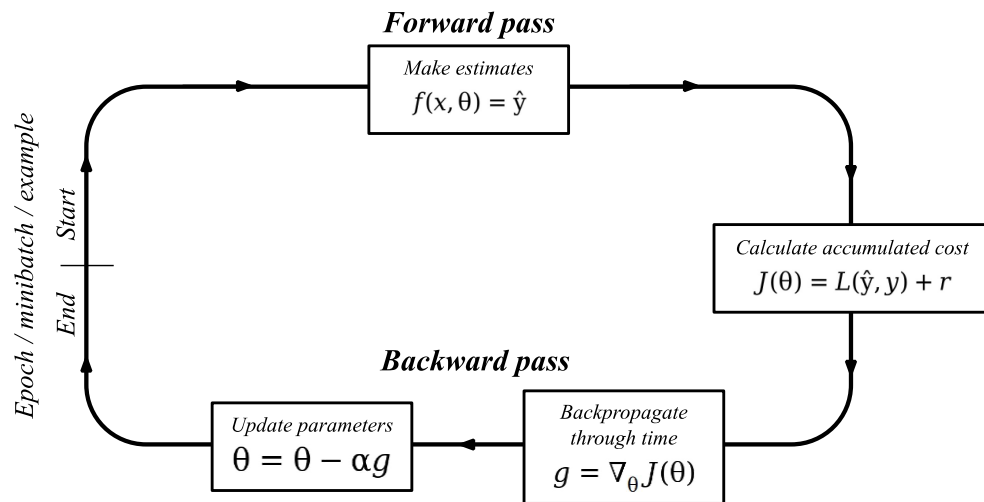


Figure 12 – A training loop

Finding a satisfying set of parameters for an ANN model is usually an iterative process that consists of separate steps, all of which are designed to minimize the cost or provide information about how the optimization process evolves. The steps that are used in this study are depicted in figure 12. During the whole training process, the training set, and typically also the testing set, are fed into the model several times. These datasets can be fed through the training loop all at once, in smaller batches, also commonly called minibatches, or one example at a time. Typically, only the training set is divided. In the optimal scenario, the training set is fed one example at a time. However, it demands lots of computations, especially if the training set consists of a vast number of examples and, hence, minibatches are often used instead. When the whole training set has been fed through the training loop in one way or another, one epoch of training has been carried out. (Bengio 2012, 442-444; Goodfellow et al. 2016, 149-150, 274-276.) This study applies the minibatch training method, and the following sections introduce the steps in the proposed minibatch training loop in more detail.

2.7.6 Forward pass and cost function

The phase where the model receives some inputs and calculates some outputs is called the forward pass and it is generally denoted as

$$f(x, \theta) = \hat{y},$$

where the function f is an ANN that transforms some input values x into estimates \hat{y} . These estimates are then used in the next step to determine the cost according to some cost function $J(\cdot)$. The cost function typically consists of some loss function and it might also include regularization terms. (Goodfellow et al. 2016, 172, 272.) In this study, mean squared error (MSE) is used for calculating the loss $L(\hat{y}, y)$ and an L1 regularization term l_1 to drive weight decay during the training process. The cost function can be written as follows

$$J(\theta) = L(\hat{y}, y) + l_1.$$

The cost function can be seen to lead the learning process and, hence, it should be examined carefully to better understand the behavior the models learn from the training set. By selecting what type of errors and model attributes generate how much cost, the model can be guided to focus on certain types of patterns in the data, and to produce a certain type of outputs. The following sections provide more details concerning the chosen cost function, along with justifying the selected methods.

2.7.6.1 Loss function

Mean squared error (MSE) and mean absolute error (MAE) are both common functions for calculating loss in regression problems. For some minibatch b , MSE and MAE can be calculated by using equations:

$$MSE_b = \frac{1}{n} \sum_{i=1}^n (\hat{y}^{(i)} - y^{(i)})^2,$$

$$MAE_b = \frac{1}{n} \sum_{i=1}^n |\hat{y}^{(i)} - y^{(i)}|,$$

where n represents the number of examples in minibatch b , and i the index of an example in the minibatch. When applied in a cost function, the loss function is used for calculating the average loss for a minibatch after example-specific losses have been obtained.

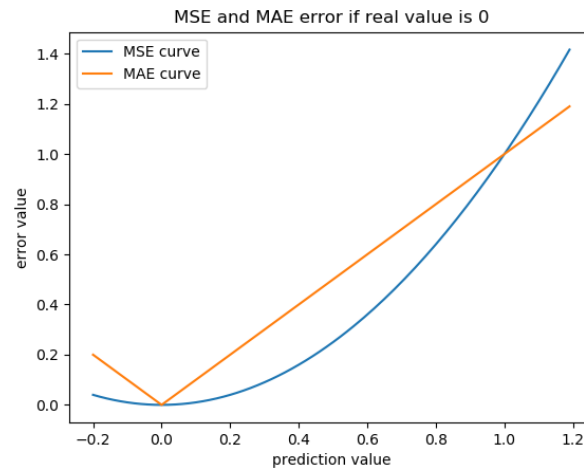


Figure 13 – An error curve comparison between MSE and MAE

In this study, the regression task resembles a binary classification since the business cycle time-series has only two possible values, 1 and 0. However, it is important to acknowledge that the predictions do not exactly represent real probabilities. Because the model treats the task as a pure regression, there are fundamentally no boundaries for the prediction values, but during the training process, the model should learn to target the predictions between 0 and 1. When comparing the equations and the error graphs for the aforementioned interval in figure 13, one can find that MSE generates less or an equal amount of error compared to MAE if the model predicts values between -1 and 1. Whereas MAE can be considered more neutral when it comes to determining how much loss is generated from an inaccurate prediction, MSE would encourage a model to give more significant signals if it finds a risk of a peak or trough happening during the multi-step prediction, since it produces less loss than MAE. This attribute is considered valuable for an application that could be used as an early warning system, although it further averts the predictions from pure probability values. While it is reasonable to penalize less from giving a hint that a peak or trough might occur during the forecasting period, choosing MSE can also be justified by examining the ratio between expansions and recessions. As mentioned in section 2.2, recessions' share of the whole time period between 1980 and 2015 is only approximately 13.334%. Therefore, it is probable that the model recognizes recessions quite rare and, hence, favors predicting values close to zero. Therefore, in order to encourage a model to output some other values, MSE could be selected over MAE. Furthermore, predicting values over one is unnecessary and, therefore, generating more error for such values is reasonable and should not affect the learning negatively. For these reasons, MSE has been selected as the loss function for the model training process, but both, MSE and MAE, are used for model evaluation in this study.

2.7.6.2 L1 regularization

The L1 regularization consists of one coefficient hyperparameter λ and the sum of all the parameters' absolute values:

$$l_1 = \lambda \sum_{p=0}^P |\theta_p|,$$

where P denotes the number of parameters in the model. Its core idea is to drive parameters that model weak, probably unnecessary, relationships or noise close to zero. This method, therefore, helps to reduce the model's sensitivity to small changes in input values and, hence, decreases the amount of variance the model introduces to its predictions. In addition to the ability to make the training process more robust against overfitting, it can also be seen to further enhance the ANN models' feature selection properties, where it diminishes the less important relationships' effect on the output. (Bengio 2012, 451-452.)

L1 regularization is used over some other regularization methods because it offers desirable theoretical foundations for more effective feature selection when using noisy multivariate data. Because the multivariate data used in this study contains time-series from several economic indicators, it is probable that some of their own movements alone or their combined representations that are made by the networks themselves are irrelevant when trying to model only the general relationships between the inputs and outputs. In addition to the L1 regularization, there is also a popular L2 regularization method, but according to Bengio (2012, 451), it should not be used together with an early stopping method and, therefore, L1 is preferred instead.

2.7.7 Backpropagation through time

A backpropagation algorithm allows calculating the parameter specific gradients g , that are then used for updating these parameter values by using some learning algorithm, such as stochastic gradient descent. When using RNNs, the common algorithm for calculating these gradients is a backpropagation through time (BPTT) algorithm. If predictions from multiple time steps are used, not only from the last time step, the loss accumulates from several output values $\hat{y}_t^{(i)}$ at different time steps t , as depicted in figure 9. Thus, the accumulated loss for example i would be

$$L(\hat{y}^{(i)}, y^{(i)}) = \sum_{t=1}^T L(\hat{y}_t^{(i)}, y_t^{(i)}),$$

where T denotes the number of time steps whose output values are used for calculating the loss. At each time step, the output values are typically calculated by using different

input values, but in turn, the parameter values stay the same. Therefore, the same parameters account for several output values' errors in accordance with their relative inputs. (Goodfellow et al. 2016, 373, 378-380.) Now the equations for determining the cost for example i can be written as follows

$$J(\theta)_t = L(\hat{y}_t^{(i)}, y_t^{(i)}) + l_1$$

$$J(\theta) = \sum_{t=1}^T J(\theta)_t.$$

When calculating the gradients, BPTT takes into account the possible multiple outputs at different time steps and the cumulative effect of the parameters that are used in recurrent connections to the outputs. This method works in the opposite direction of the forward propagation by utilizing the chain rule for partial derivatives for determining the parameter specific gradient values and then summing them together. (Graves 2012, 19-20; Goodfellow et al. 2016, 378-380.) The common equation for the BPTT algorithm is the following

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{t=1}^T \frac{\partial J(\theta)_t}{\partial \theta},$$

whose actual calculations depend significantly on the network, similar to the forward propagation. Several more accurate presentations exist for the common RNN architectures.^{3,4} This study uses only the output values $\hat{y}_T^{(i)}$ that are received on the last time step T . In the following sections, the BPTT algorithm is denoted ∇_{θ} and all the gradients for the parameters θ are stored in g .

Because BPTT applies an extensive number of multiplications, it is sensitive to the values that it multiplies. High and low input and activation values might cause gradient values to increase or diminish substantially and these problems are called the exploding and vanishing gradients. (Goodfellow et al. 2016, 282-285.) When considering the cost plane, this introduces sharp edges and plateaus that could oscillate the training process if it happens to traverse through this type of region. Therefore, training an RNN typically oscillates more than the other type of networks, such as an MLP. However, LSTM and GRU should be more robust against these types of problems, but in addition, a more stable learning algorithm is also used in this study.

³ For a classical RNN unit: Graves (2012, 19-20) and Goodfellow et al. (2016, 378-380)

⁴ For an LSTM cell: Graves (2012, 38)

2.7.8 Nesterov accelerated gradient

A stochastic gradient descent (SGD) and its different variants are considered as the most popular learning algorithms in DL. An SGD uses the average gradient values that are calculated over a minibatch of n examples. To summarize the previous steps in the training loop, the equation for calculating gradients at epoch e for minibatch b by using some backpropagation algorithm ∇_{θ} is

$$g_{(e,b)} = \nabla_{\theta} \left(\left(\frac{1}{n} \sum_{i=1}^n L(f(x^{(i)}, \theta), y^{(i)}) \right) + r \right).$$

First, the training set is divided into minibatches $\mathbb{B}(b)$ that contain n number of examples, hence $\mathbb{B}(b) = \{x^{(1)}, \dots, x^{(n)}\}$. The examples in minibatch b are then provided to a forward pass to generate estimates for a loss function. Next, the sum of the example wise loss is divided by the number of examples in the minibatch. Then, some regularization terms could be added, that follows some backpropagation algorithm ∇_{θ} , such as BPTT, that obtains parameter specific average gradients for the minibatch that are stored in $g_{(e,b)}$. (Goodfellow et al. 2016, 149-152, 276, 290-292.)

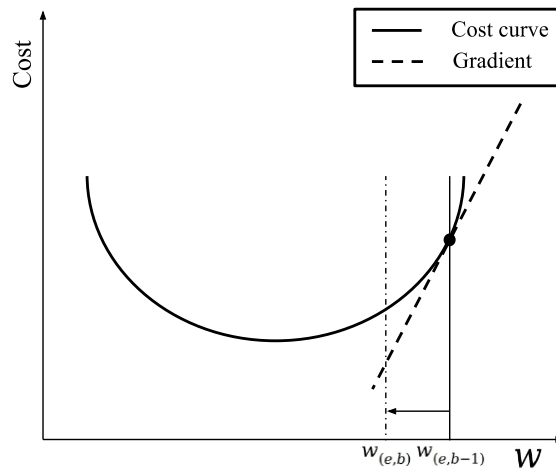


Figure 14 – A gradient descent step for one parameter

After the average gradients have been calculated for minibatch b , for example, an SGD can be applied to handle the actual learning by updating the parameter values by using the following equation:

$$\theta_{(e,b)} = \theta_{(e,b-1)} - \alpha_{(e)} g_{(e,b)}.$$

The fundamental idea of a gradient descent algorithm is to minimize the cost gradually step-by-step. An example is depicted in figure 14 for some parameter w . This parameter is updated in the opposite direction to its gradient, by a magnitude defined by the gradient

and an adjustable learning rate $\alpha_{(e)}$. The learning rate is typically decreased during the training process to allow larger updates at the beginning of the training process to speed up learning, and later, the smaller updates enable it to settle to some minimum on the cost plane. (Goodfellow et al. 2016, 149-150, 290-292.) The learning rate is examined in more detail in section 2.7.9.

The parameter specific cost curve, depicted in figure 14, is a two-dimensional slice of the multidimensional cost plane that is determined by the cost function. For every example, the curvature of the cost plane is different, because it is determined by the factors that are used for calculating the cost $J(\cdot)$. Moreover, the longer multiplication chains the network architecture introduces and the higher input and activation values it receives and generates, the steeper curvatures exist on the cost plane. (Goodfellow et al. 2016, 285-286; Trask 2019, 86-87.)

When a parameter value is changed, the coordinate on the global cost plane changes. The global minimum can be found without updating all the parameters, but typically when training an ANN, all the parameters are updated at the same time because there are no sufficient methods that could determine what parameters to update or not. (Trask 2019, 86-89.)

Vanishing and exploding gradients occur due to irregularities on the cost plane. Essentially, a gradient can receive a high value if the coordinate on the cost plane appears on a steep edge, as depicted in figure 15. If the gradient values are huge, they will move the coordinate on the global cost plane substantially, possibly in the wrong direction and, hence, damage the learning process. Also, vanishing gradients harm the learning process by making parameter updates close to zero and, thus, impeding the search for lower values on the global cost plane. (Goodfellow et al. 2016, 285.)

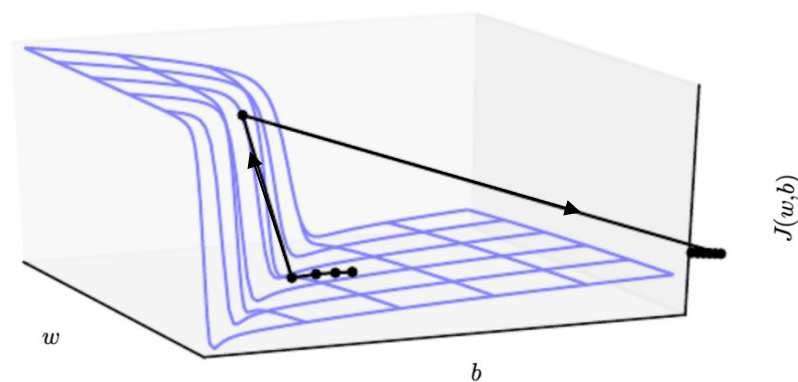


Figure 15 – A three-dimensional illustration of how a cliff affects a training process (Goodfellow et al. 2016, 285)

Several modifications of the SGD method exist that allow a more stable and efficient training process. The momentum method adds a vector to the normal gradient descent algorithm that gives it a similar attribute as mass does to objects in physics. It typically helps the learning algorithm to navigate more consistently on the cost plane and allows it to converge faster. (Géron 2017, 299-300; Goodfellow et al. 2016, 292-293.) Figure 16 compares parameter updates made by an SGD with and without momentum. In this illustrative example, the red line represents the learning path when the momentum is used, and the black arrows depict what the parameter updates would have been without the momentum. When using the momentum, the parameter updates appear less radical and, hence, it also converges to the minimum faster.

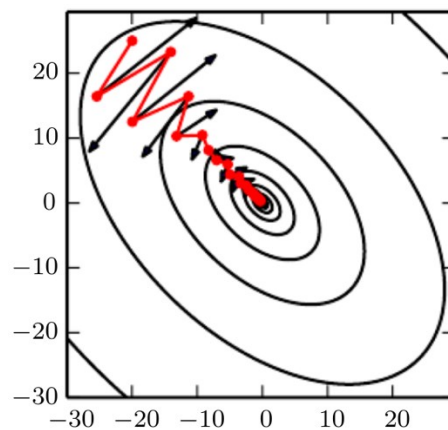


Figure 16 – A comparison of gradient descent updates with and without momentum (Goodfellow et al. 2016, 293)

The equations for an SGD algorithm with momentum are

$$m_{(e,b)} = \beta m_{(e,b-1)} - a_{(e)} g_{(e,b)},$$

$$\theta_{(e,b)} = \theta_{(e,b-1)} - m_{(e,b)},$$

where $m_{(e,b)}$ is a momentum vector that consists of all the momentum values, one for each parameter, at the current epoch e and batch b . It is calculated by using the previous momentum $m_{(e,b-1)}$, the hyperparameter β , that determines the momentum effect's strength, which is typically set to 0.9, the current gradient $g_{(e,b)}$ and learning rate $a_{(e)}$. (Géron 2017, 299-300.)

Nesterov accelerated gradient (NAG) is a modification to the SGD with a momentum that, according to Géron (2017, 300-301) and Goodfellow et al. (2016, 296), typically further speeds up the learning process and, hence, requires less computing to converge. The fundamental idea in NAG is to take the previous momentum values into account

when calculating the current gradients. By adding the previous NAG momentum effect $\beta m_{(e,b-1)}^{NAG}$ to the parameter values, the gradients can be calculated from a further point on the cost plane and, thus, correct and speed up the parameter updates. In this study, NAG is used for parameter optimization in order to use scarce computing resources more efficiently. Now the parameter updates are given by the following equations:

$$\begin{aligned} g_{(e,b)}^{NAG} &= \nabla_{\theta} \left(\left(\frac{1}{n} \sum_{i=1}^n L(f(x^{(i)}, \theta + \beta m_{(e,b-1)}^{NAG}), y^{(i)}) \right) + r \right), \\ m_{(e,b)}^{NAG} &= \beta m_{(e,b-1)}^{NAG} - \alpha_{(e)} g_{(e,b)}^{NAG}, \\ \theta_{(e,b)} &= \theta_{(e,b-1)} - m_{(e,b)}^{NAG}. \end{aligned}$$

When using an SGD or some of its modifications along with time-series data, it is prone to parameter changes over time because the model is trained with data that is in chronological order. For example, the last gradients will be calculated by using the last minibatch of data, including the latest events that will now have the last say in the training process. If this last minibatch is very different compared to all the other minibatches, it could play a bigger role in parameter learning than is necessary. (Nielsen 2019, 347.) One way to prevent problems with too diverse minibatches is to increase the batch size to include more examples from a longer time span, therefore potentially making the separate minibatches' distributions of different types of events more alike and, hence, giving less importance on extreme events when calculating the gradients. However, using very small minibatches typically offers the best generalization and also adds regularization (Goodfellow et al. 2016, 276). The batch size is typically set between one and a few hundred, but when using graphical processing units (GPUs), as in this study, it is recommended to use power of two batch sizes since they offer a better runtime. (Bengio 2012, 448; Goodfellow et al. 2016, 276.) In order to arrive at some compromise, the minibatch size was set to 32, where each example represents one day.

2.7.9 Learning rate

The learning rate and its initial value at the beginning of a training process are argued to be among the most important hyperparameters in DL when an SGD or some of its modification is used. The typical values for a learning rate are between 1 and 10^{-6} if the inputs are scaled between 0 and 1. As mentioned previously, the learning rate is commonly decreased during the training process according to some schedule or function. No optimal learning rate method has yet been found. Instead, a suitable learning rate decay depends significantly on the task and other proposed methods. (Bengio 2012, 447-448.)

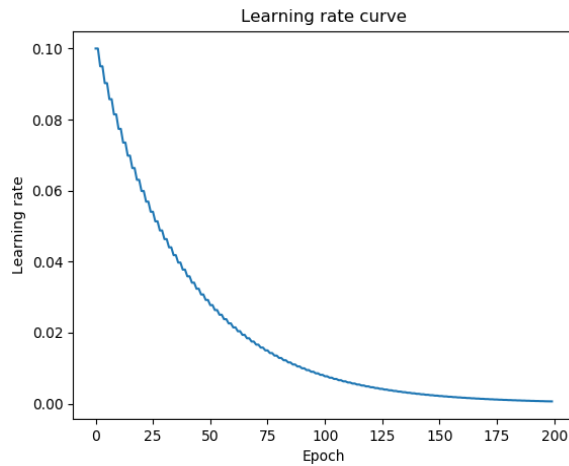


Figure 17 – The learning rate curve

Because the learning process is unique for every different problem, a handmade learning rate schedule is used, which is based on several test runs with the chosen setup. The learning rate schedule is determined by the following equation:

$$a_{(e)} = 0.95^{\lfloor e/\vartheta \rfloor} a_{(0)},$$

where $a_{(e)}$ denotes the learning rate for epoch e , $\lfloor e/\vartheta \rfloor$ means the highest integer that is not more or equal to e divided by ϑ . This gives the frequency of how often a drop to the learning rate should be applied. The initial learning rate $a_{(0)}$ is set to 0.1, and the drop is applied to every other epoch, meaning that the hyperparameter ϑ has a value of 2. At the beginning of the training process, the learning rate has relatively high values to speed up learning and to allow escaping some local minimum on the global cost plane. During the training process, the learning rate is gradually decreased, as shown in the learning rate curve in figure 17, in order to make it possible to settle down on some minima. This learning rate schedule may suit some ANNs better than the others. Therefore, studying the effect of different learning rate schedules and adaptive learning rate optimization algorithms, such as RMSprop and Adam, is motivated in future work.

2.7.10 Parameter initialization

The initial parameter values of an ANN model have a substantial effect on the whole training process. Typically, at the beginning of the training process, some small negative and positive values are set to the parameters by using some random function when using gradient descent learning algorithms together with RNNs (Graves 2012, 30; Goodfellow et al. 2016, 296-298). Furthermore, Goodfellow et al. (2016, 297-299) argue that random values should be used in order to break the symmetry between an ANN's units that would

otherwise make it possible for the learning algorithm to update them identically throughout the training process if some other unit specific stochasticity is not implemented. Also, when using RNNs that introduce long chains of computations, smaller initialization values should be preferred in order to avoid exploding gradients and unnecessarily high sensitivity to some input values.

Graves (2012) used in his RNN experiments a flat random distribution in the range between values -0.1 and 0.1 , along with a Gaussian distribution with a mean of 0 and a standard deviation of 0.1 . He found that these two methods performed similarly. In order to follow these guidelines, the same Gaussian distribution with a mean of 0 and a standard deviation of 0.1 is used for the parameter initialization in this study.

2.7.11 Training process and early stopping

The typical goal for the training process in DL is to find the parameter values θ that make the ANN model f as similar as possible to the function f^* , which has generated the data (Goodfellow et al. 2016, 164). As introduced in section 2.7.5, the process involves training epochs that drive the parameter values iteratively towards a low-cost area on the global cost plane. There is no strict limit to the number of epochs, but theoretical foundations suggest that there is a relationship between error and the number of training epochs.

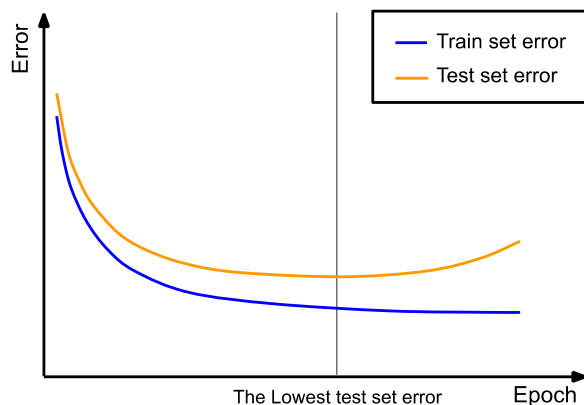


Figure 18 – Error curves for the training process

Figure 18 shows an example of a training graph where the metric on the vertical axis is some error measure and the horizontal axis shows the training epochs. Since the initial weights are usually small random numbers, the error curves tend to start from high values, but during the training process, the weights begin to settle on more suitable values as they begin to model meaningful relationships in the data, enabling a better fit. This, in turn,

decreases the error, but could also make the model eventually overfit the data if it has enough modeling capacity. (Murphy 2012, 572.) This behavior is strongly related to the bias-variance trade-off. At the beginning of the training process, the out-of-sample error is typically dominated by a high bias value, but if the model's modeling capacity is large enough, the bias decreases during the training process and the variance begins to increase, though these changes are rarely monotonic. (Prechelt 2012, 64-65.)

Early stopping is a practical method when training an ANN with sufficient capacity to start overfitting. It can be seen as a regularization method since its goal is to end the training process at the right time when the model performs best with unseen data. When the training process is running, the performance metrics can be used to detect overfitting, that is indicated by a rising test set error curve. The stopping criterion is typically set so that the training is halted when reasonably strong signs of overfitting have been received. A well working early stopping method stops the training process close to the point where the lowest out-of-sample error has been achieved. It also allows saving computing resources by halting the process before some of the unnecessary iterations have been executed. (Goodfellow et al. 2016, 243; Géron 2017, 136; Prechelt 2012, 54.)

Finding the optimal point to stop the training process is difficult for various reasons. Firstly, as shown in figure 16, the parameter updates might take missteps and, hence, the out-of-sample error can spike occasionally. Therefore, stopping the training process instantly when the out-of-sample error increases could be unreasonable. Secondly, the out-of-sample error itself is just an estimation of how well the model would perform in the future. Hence, stopping at the lowest out-of-sample set error does not guarantee the best possible model. Thirdly, the process depends heavily on the error measure. If the error measure is not well in line with the actual goal, early stopping itself would not yield optimal models even though it could stop the training process close to the lowest point. Lastly, as Prechelt (2012, 53-55, 64) points out in his study, the stopping criterion is often set by using some task-related knowledge instead of following some theoretical foundation. Arguably, because the training curves are also acknowledged to be task-dependent.

Because this study focuses on comparing different ANN architectures, not different optimization methods, only one early stopping method is used. Although Prechelt (2012) has suggested more advanced methods that have the potential to generalize well on different types of tasks, a lighter solution is used in this study. The chosen early stopping threshold is set so that the training process is halted when the test set cost has not decreased during the last 25 epochs. This method has been chosen based on a few test runs

to figure out how well a network with a mid-tier modeling capacity in this study appears to converge. Also, the test set cost curves tended to oscillate considerably, and the 25 epoch patience buffer appears to have enabled the training processes to reach the bottom of the test set curve most of the time.

When the training process is over, it is possible to examine the ANN with parameters that generated the lowest error or cost, but this could give a too optimistic picture of the ANN's general performance because of the notable oscillations in the out-of-sample error and, hence, the last parameter values are used instead.

2.7.12 Deep learning hyperparameters

Bengio (2012, 446) defines a hyperparameter for some ML algorithm A as

“a variable to be set prior to the actual application of A to the data, one that is not directly selected by the learning algorithm itself.”

Thus, hyperparameters, such as ANN architecture, learning rate, minibatch size and the maximum number of training epochs, can also be described as control knobs for the actual parameter search. They form their own search space and, hence, when applying DL, one has to deal with two search problems. Because the hyperparameters define the parameter search, they have a fundamental role in finding a well-fitting ANN model.

The hyperparameter search can be conducted by hand or by some search algorithm. Furthermore, the theoretical background does not offer clear foundations for finding an adequate hyperparameter setup and instead, the process usually involves trial and error loops when finding a suitable setup. Though the absence of a mathematically proven best practices for hyperparameter tuning, some guidelines exist. In this study, the current guidelines and common best practices have been followed together with some pretesting to tailor the setup more suitable for the task at hand. However, in the future, more hyperparameter setups should be reviewed.

3 LITERATURE REVIEW

3.1 Deep learning for time-series analysis

Deep learning is a relatively novel set of methods in the field of time-series forecasting, but it has already been recognized providing prominent tools for certain types of problems. Zhang (2012, 463) and Nielsen (2019, 289-290) propose that ANN models are data-driven nonparametric modeling methods, and they are less vulnerable to model misspecification when compared to the more traditional parametric methods in time-series forecasting. Furthermore, they do not require as many and as strict assumptions from the data generating process as the more traditional time-series methods. For example, no assumptions about the data distribution are required and stationarity is not necessary. Also, they appear to fare well with heterogeneous data. Because of these properties, ANNs enable using some time-series that would have been unideal with the more traditional methods. However, using DL often requires large datasets and, thus, might not be suitable for tasks that cannot offer time-series with a sufficient number of realizations (Nielsen 2019, 186).

In addition to their flexibility, they are recognized as universal approximators and, therefore, in theory, they are suitable for modeling very complex stochastic processes. Moreover, ANNs belong to a class of models that offer an ability to model nonlinear data due to their nonlinear activation functions. When compared to nonlinear methods, linear methods have been more popular in time-series analysis, albeit in many real-world applications, the time-series data is nonlinear (Tsay & Chen 2019, 1). The popular linear methods have offered good performance in a variety of modeling tasks in different domains, and they can be fit to data reliably and efficiently, but their modeling capacity is limited to linear functions (Goodfellow et al. 2016, 165). Therefore, fitting one to a nonlinear process causes underfitting and, hence, nonoptimal performance.

It is argued that some economic and financial time-series data contains nonlinearity. This makes the models that are able to model also these nonlinear relationships interesting in these particular domains because they can model nonlinear patterns and, thus, possibly gain a better performance when compared to the prominent linear models. The following sections offer a review of how ANNs have fared in previous related research and, thus, it serves as a background and motivation for the proposed methods.

3.2 The novel RNN methods' performance in the previous literature

According to the NFL-theorems, no model is universally better than all the other options in every situation where modeling is needed. Furthermore, according to the bias-variance trade-off theory, the model capacity should be suitable for the underlying problem. Hence the question, in the domain of time-series analysis, what type of problems the novel RNNs suit the best?

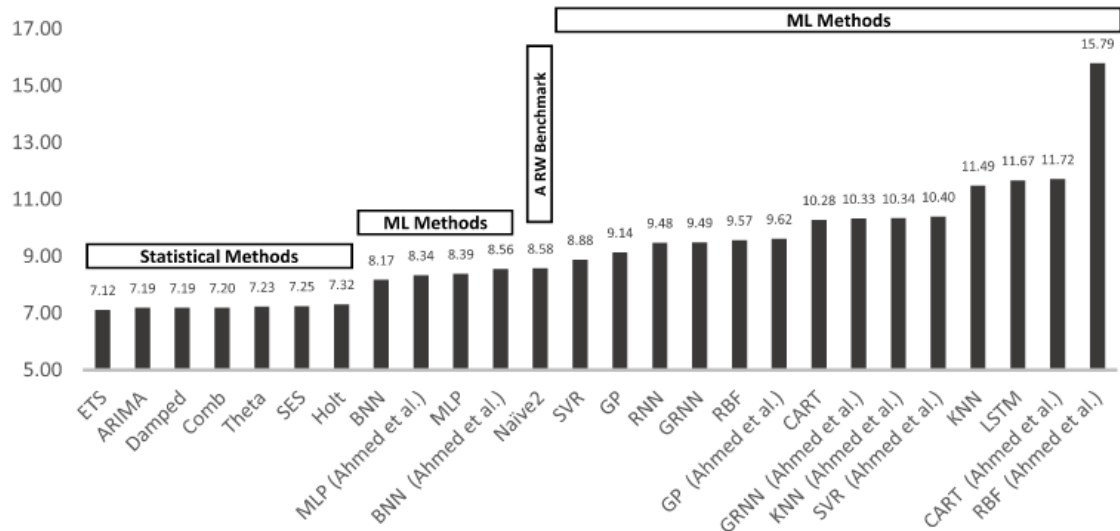


Figure 19 – Forecasting performance (sMAPE) of the ML and statistical methods included in the study (Makridakis et al. 2018, 15)

Recently, plenty of research has been conducted concerning how well the novel RNN methods suit time-series forecasting. An extensive study made by Makridakis et al. (2018) evaluated a wide range of different types of models by using more than 1000 univariate time-series from different areas, economics and finance included. The results of this study, shown in figure 19, suggest that the applied LSTM methods are among the worst-performing when univariate time-series are used. These findings are in line with several other previous studies. Gers et al. (2001) also point out LSTM methods' weak performance when applied to univariate time-series forecasting. Moreover, Makridakis and Hibon (2000, 458) found that

“Statistically sophisticated or complex methods do not necessarily produce more accurate forecasts than simpler ones,”

when applied to univariate data, which can also be verified in figure 20.

These findings imply that an LSTM network and its close variants should not be considered if univariate time-series is chosen. One possible explanation for these results is that these univariate datasets do not contain aspects that could be exploited by using these more sophisticated methods. For example, an MLP, the common feedforward ANN structure that is not developed particularly for analyzing sequential data, which is examined in section 2.7, is able to beat the more sophisticated LSTM network in these particular time-series tasks. The reason why the MLP was able to outperform the LSTM network might be due to its ability to concentrate better on simpler patterns in the data and model less noise than the LSTM network, but this has not been verified. Also, the ratio between nonlinear and linear datasets might have affected the results, here possibly to the benefit of the proposed linear models.

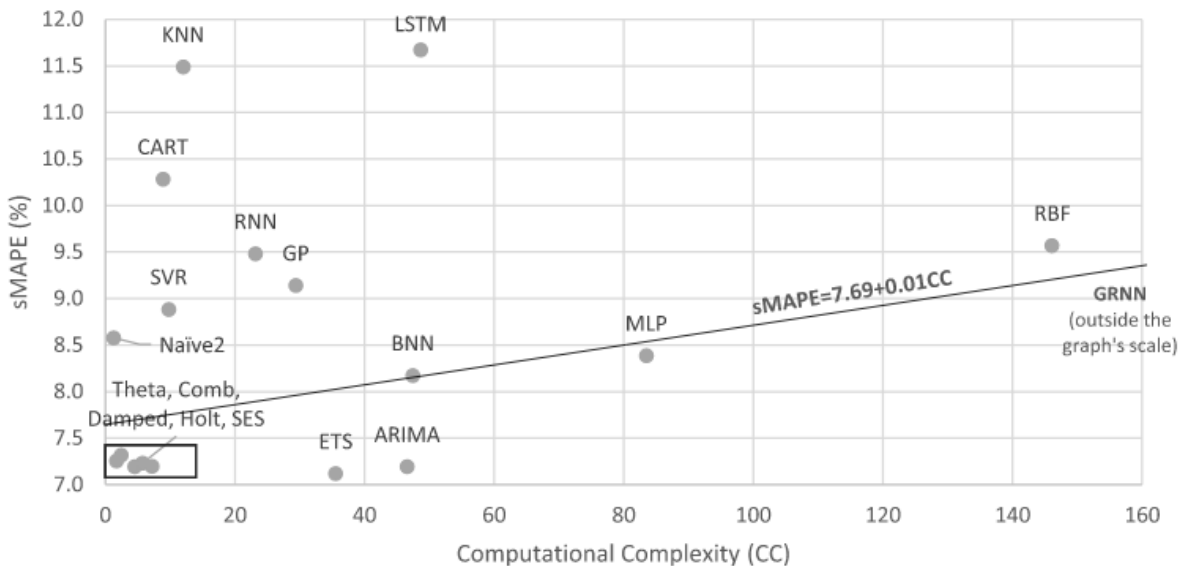


Figure 20 – Forecasting performance (sMAPE) versus computational complexity (Makridakis et al. 2018, 18)

However, unprecedented results have been achieved when LSTM networks and their variants have been applied to very difficult sequential tasks, such as analysis of audio (Marchi et al. 2014), video (Donahue et al. 2017) and machine translation (Luong et al. 2015). This indicates that there is a specific area of sequential tasks that are suitable for these more sophisticated methods. These tasks seem to have similar features as the other areas where ANNs have enjoyed greater success than any other model class; the relationships between input and output pairs are difficult to express by using human-made mathematical functions and a sufficiently large amount of high-quality data is available for an extensive training process.

These findings indicate that, although the performance with univariate time-series data appears appalling, they seem to fare well in very complex domains, which is in line with the presented theories. Therefore, these methods should be applied first and foremost to very complex tasks that are more suitable for their characteristics.

As the novel RNNs have been found to perform rather poorly with univariate time-series, they have also been criticized for lacking strong theoretical foundations concerning some fundamental decisions when creating, training and interpreting these models. For example, Brooks (2014, 420) finds that ANN models have four major weaknesses. Firstly, the parameter values in an ANN do not offer a meaningful interpretation and, hence, these models are commonly regarded as “black boxes.” Secondly, no diagnostic or specification tests exist for determining an ANN model’s adequacy properly. Thirdly, ANN models tend to overfit quickly to a training set and, therefore, they might provide poor performance with a testing set and later in a production environment. Lastly, when compared to the other possible options, optimizing the parameters of an ANN model demands typically significantly more computational resources than the other typical methods. In addition to these aforementioned weaknesses, there is a shortage of proper theoretical foundations for determining and adjusting hyperparameters in DL. Hence, scientifically proven best practices for hyperparameter search do not exist, but the area has developed some guidelines through continuous research.

The hyperparameter setups used in this study were motivated by other research in the field and, therefore, this study can be seen as a part of this larger research related to DL and suitable hyperparameter combinations. Also, to take into account the other weakness mentioned by Brooks (2014, 420), this study focuses on finding a suitable network architecture for the task at hand and, thus, makes interpreting the parameter values redundant. The evaluation of a set of different RNN architectures, which is ultimately a hyperparameter search problem, is done by examining their fit to unseen data by using common performance measures. The ANNs’ tendency to overfit data is taken into account by applying L1 regularization, and the test set performance measures are tracked throughout the training process in order to notice possible overfitting. Furthermore, an early stopping functionality is used to abort the training process before significant overfitting occurs. These methods should offer a relatively satisfying base for making decisions related to assessing these methods’ general adequacy to the task at hand.

3.3 Review of the previous related research in finance and economics

Overall, machine learning methods have not yet obtained a significant role in economics, although they have been found to offer great tools for forecasting (Varian 2014). However, even though only a limited amount of research has been conducted on ML in the domain of economics, many studies have been conducted concerning their applicability to finance, possibly because it is generally more focused on predicting than on understanding the underlying process. Although finance and economics are considered separate fields, their time-series data typically has similar qualities, such as nonlinearity, noise, seasonality and trends, together with an underlying data generating process that is considered extremely complex. Hence, the findings done in the domain of finance can give a clue how well these methods perform when applied to economic forecasting. Although several studies have been conducted related to applying ANNs to finance and economics, only a few are examined here that help to understand the bigger picture, the most prominent trends in financial and economic forecasting over time and the reasons behind them.

3.3.1 Related research in finance

Kumar (2009), Niaki and Hoseinzade (2013), Sheta et al. (2015) and Fischer and Krauss (2018) compared different models for predicting popular stock indexes, such as S&P 500 and Dow Jones Industrial Average. Kumar (2009) found that an MLP was able to outperform an ARIMA model when a univariate data of S&P 500 returns was used. In his study, the MLP could predict the direction of change (a classification problem) for the next day's S&P 500 value better than the proposed ARIMA model. When compared to results by Makridakis et al. (2018) where the ARIMA tended to outperform the MLP when applied to univariate data, this finding suggests that there can be exceptions when these models are applied to different time-series and different types of tasks, here a classification instead of pure regression.

Niaki and Hoseinzade (2013) and Sheta et al. (2015) compared MLPs to other types of models when using multiple input variables for predicting S&P 500 daily returns, which in turn was a regression task. The input variables included several common financial time-series, such as a relative change in gold and oil, a few dollar related exchange rates and the change in the market yield on U.S. Treasury securities for several maturities. They found that MLPs were able to outperform the proposed linear models. However, Sheta et al. (2015) discovered that a support vector machine, which is a common ML

model class that is also capable of modeling nonlinear relationships in data, could outperform the proposed MLP model. This is also in line with the Makridakis et al. (2018) results where the SVR model, which is a support vector machine model for a regression task, outperformed the MLP when applied to univariate data. Moreover, this suggests that more emphasis should be put on using different types of nonlinear machine learning models for modeling financial data, not just ANNs with nonlinear activation functions.

To summarize, Kumar (2009) studied the common MLP model and found that when using univariate data, the method might be useful when the prediction problem is transformed into a classification task. In turn, Niaki and Hoseinzade (2013) and Sheta et al. (2015) found that when using multivariate data, the nonlinear methods, including MLPs, are able to outperform the tradition linear models, possibly by exploiting the nonlinearity in the data. Both findings are in line with the Makridakis et al. (2018) findings, and further build up the understanding of which methods should be applied to what type of prediction problems. Moreover, many of these aforementioned researches put emphasis on short-term dependencies in the data and, thus, they leave the effect of the ability to model long-term relationships unknown.

As presented previously, in theory, the novel RNN methods should suit very complex modeling tasks that require modeling long-term dependencies. However, relatively few research exist where these methods have been applied to established research areas. As mentioned earlier, ANNs have several drawbacks, and the first RNNs were difficult to train and could not deliver good performance when modeling long-term dependencies was necessary. These reasons might have slowed down the ANN and RNN related research in finance and economics. However, more lately, possibly encouraged by the success in other areas, these methods have gained more attention in finance. Fischer and Krauss (2018) studied how well LSTM networks can predict the future share price for stocks that are included in the S&P 500 list. The study compared LSTM networks performance against MLP, random forest and logistic regression models. Their data comprised approximately 500 univariate time-series, one for each stock in the S&P 500 list during the examined time period from 1992 to 2015. Each model received only one stock's data, hence yielding approximately 500 separate models for each model class. The task was a binary classification, where a model predicts which side of the cross-sectional median revenue some particular stock would be at the next time step. After conducting several tests, they found that the chosen LSTM networks could outperform the other models, though random forest models, which is also a common nonlinear ML model class, could

achieve similar scores in some areas. This is in line with the previous studies since both, the support vector machine that outperformed the MLP in the Sheta et al. (2015) study and the random forest, are nonlinear ML models that possess high modeling capacity. What stands out is that when compared to Makridakis et al. (2018) results, an LSTM network could now outperform an MLP even though univariate data was used. This might be due to altering hyperparameter setups between the studies or because the task was transformed into a classification task in Fischer and Krauss (2018) study. Either way, finding a suitable ANN model and training it in the best possible manner might be harder when compared to the other proposed methods in these studies because of their vast hyperparameter space and nonconvexity (Krauss et al. 2017, 695). Therefore, more hyperparameter setups should be traversed before determining which method is best suited for predicting the S&P 500 and other stock indexes.

3.3.2 Related research in economics

After decades of research, sophisticated nonlinear ML models have started to gain more traction also in economics. This change is arguably driven by two main reasons. Firstly, macroeconomic forecasting is typically recognized as a very complex prediction problem. Secondly, the prevalent econometric and linear models have been found lacking the ability to forecast macroeconomic variables sufficiently (Moody 2012, 345). It appears that the econometric models that are created for evaluating and describing the economy are not suitable for making accurate forecasts, and the linear models' inability to model nonlinearity is arguably a defect when one wants to model complex macroeconomic processes that are considered to include nonlinearity. For example, business cycles, in particular, are thought to be a nonlinear process for a few reasons. Firstly, even though defining a recession and expansion is found somewhat difficult, expansions typically last longer than recessions (Mitchell 1927, 456-458). Secondly, the upward and downward motions seem to be asymmetrical. Generally, an expansion evolves gradually as the favorable events in the economy have a cumulative effect on each other until, at some point, the process starts to work in the opposite direction, but typically in a shorter time span and a more volatile manner (Keynes 1936, 313-314). Thirdly, Mitchell (1927, 458) finds the economy highly sensitive to rare, possibly unpredictable phenomena, such as wars, poor harvests and epidemics that typically escalate quickly, and because of the economic processes' interdependency, also globally. Moreover, these types of phenomena rarely expand the economy and, instead, they often account for a quick and volatile drop in

economic activity. This finding is also found true recently due to the economic consequences of the COVID-19 pandemic. Fourthly, Friedman (1964) suggested a “plucking model” when he observed that the amplitude of a recession is strongly related to the strength of the following expansion, while the expansion’s amplitude does not appear to have a significant effect on the following economic contradiction.

However, the early economic forecasting research found linear methods superior to nonlinear correspondents (Meese & Geweke 1984; Makridakis et al. 1982; Makridakis et al. 1993; Weigand & Gershenfeld 1994; Swanson & White 1995; Swanson & White 1997; Stock & Watson 1998). These studies have been conducted by using univariate time-series from various sources, generally finding exponential smoothing and autoregressive models the most suitable. As examined previously, similar results have been received from the more recent studies that have used univariate data. For example, Makridakis and Hibon (2000) and Makridakis et al. (2018) found that exponential smoothing (ETS) and autoregressive integrated moving average (ARIMA) seem to have led the pack performance-wise. In addition to the linear methods’ strong performance compared to nonlinear methods when using this particular type of data, Stock and Watson (1998) proposed that, in order to achieve the best possible performance, a combination of two different linear models’ forecasts should be used. This type of procedure is a common technique in ML, which is typically called ensemble learning, where two or more models are used to create predictions together by using, for example, some voting or weighting mechanism for crafting the final prediction values (Géron 2017, 182-187).

ANNs have received mixed results in economic forecasting research. Swanson and White (1997) conducted an extensive study where they compared ANNs to VAR models by using several different macroeconomic multivariate datasets and found little to no improvement in accuracy when using ANNs. However, after this study, the DL methods have evolved significantly and, as opposed to these findings, the later research found suitable modeling tasks for ANNs also in macroeconomics. Tkacz (2001), Chuku et al. (2017) and Jahn (2018) showed that ANNs were able to offer better accuracy than the common linear methods when forecasting GDP growth. In addition to the suitability of forecasting GDP growth, Binner et al. (2004) found two different types of nonlinear models: an RNN and regime-switching VAR, superior to a linear VAR when modeling U.K. inflation, again showing that more emphasis should be put on different types of nonlinear models in future research.

Moreover, ANNs have been found promising in predicting business cycles, which has been regarded as a very complex prediction problem in a range of studies. For example, Moody (2012, 344-364) argues that forecasting business cycles is challenging due to five main reasons. Firstly, macroeconomics is recognized as a non-experimental science because economies are not closed systems. Hence, controlled macroeconomic experimenting is difficult, or even impossible, to conduct to get a better understanding of how the economy functions. Secondly, this particular macroeconomic process appears extremely complex. Commonly, practitioners have applied two different types of models: econometric and linear time-series models, but neither method has produced a satisfying solution for the task, possibly due to their insufficient modeling capacity. Thirdly, it is argued that business cycles are nonlinear (Mitchell 1927, 456-458; Keynes 1936, 313-314; Friedman, 1964). Therefore, macroeconomic forecasting can possibly be improved by finding a suitable nonlinear model that could achieve better accuracy by exploiting the nonlinearities in the data that the common linear methods cannot model accurately. Fourthly, macroeconomic time-series typically contain a considerable amount of noise and, hence, the important patterns in the data might be difficult to observe and model. For this reason, the model should be somewhat robust against the noise in the input data, but still be able to provide accurate predictions. Lastly, many macroeconomic series are non-stationary. This feature comes from the combination of how the economy evolves and how it is tracked. Often macroeconomic time-series include strong trends that might require applying some technique for reducing the nonstationarity in the time-series or leaving them out completely, which can lead to a loss of some valuable information.

Qi (2001) studied how to predict U.S. recessions with an MLP together with multivariate data and found them ideal for this particular forecasting task because

“... business cycles are asymmetric and cannot be adequately accommodated by linear constant parameter single-index models. NNs are a class of flexible nonlinear models. Given enough data, they can approximate almost any functions arbitrarily close. Because there is little a priori knowledge about the true underlying function that relates financial, economic and composite indicators to the probability of future recessions, the NN models are an ideal choice for modeling these relationships.”

The focus of Qi's study was to find the best explanatory variables for predicting U.S. recessions and using MLP models as a guide. The study found that the interest rate spread

between the 10-year Treasury bond and the 3-month Treasury bill was the most important explanatory variable while some other useful explanatory variables being the Department of Commerce leading index, Stock and Watson (1998) index, real money supply and S&P 500 index, some of which are also used in this study. Furthermore, the proposed MLP models could predict well all the recessions in the proposed testing set and, thus, it ended up also concluding that ANN models are promising for forecasting business cycles.

3.4 Conclusions of the previous literature

The previous literature has offered somewhat clear conclusions that correspond to the assumptions drawn from the underlying ML related theory. From these studies' findings, one can remark that, although ANNs are regarded as universal approximators and, therefore, should be in theory suitable for any given prediction task, they tend to perform worse than the popular linear methods when the task caters only univariate time-series and it is a regression problem. Instead, more promising results have been obtained when these methods have been deployed to problems that introduce more complexity, nonlinearity, relevant multivariate data. These findings are in line with the NFL-theorems and the bias-variance trade-off theory since they show that ANNs are not superior to all the other models, albeit being universal approximators, and instead, they should be applied based on the attributes of the underlying task.

The performance of the applied ANN models varies notably between the examined studies and, hence, defining unambiguous criteria concerning the attributes that should be met when ANNs or the novel RNN methods are applied is challenging. It is important to acknowledge that DL and time-series forecasting offer numerous options, each of which can affect the results. Firstly, for example in economics, the timing of a prediction might have a significant effect on its accuracy. Especially during rare circumstances, as Kock and Teräsvirta (2014, 628) found when making predictions for the time period between 2007 and 2009 across different modeling techniques, including an ANN. Secondly, different types of ANN architectures, optimization algorithms and hyperparameter setups have their own effect on performance. Thirdly, random parameter initialization and the stochasticity in the training process introduce variation also to the predictions and, thus, also to the performance results. Fourthly, different data and preprocessing methods have a significant effect on various elements in the process. Lastly, the DL methods' quick progress has made some of the previous studies' findings unable to represent the current methods' performance.

In general, however, it appears that DL methods are suitable in complex nonlinear domains, where the previous theory does not offer a clear understanding of the data generating process, or it is even thought to be too complex to be written in a functional form. Due to the ANN's favorable properties, along with Qi (2001) and others' findings, DL methods should be considered as a relevant option when forecasting macroeconomic time-series. Arguably, several important areas have not yet been studied and, therefore, more ANN related research should be conducted in economics, concerning especially the modern DL methods, such as the more advanced RNN methods that, in theory, should suit well prediction problems that include sequential data with long-term dependencies. Qi (2001) found ANNs suitable for forecasting U.S. business cycles, but the study did not focus on which type of ANN models suit the problem best. Hence, the findings of this study should be significant in assessing DL methods' suitability for business cycle forecasting, but also to the progress of the macroeconomic forecasting.

4 METHODS

According to Lütkepohl (2005, 1), it is common that the values of economic variables are related to their own and some other variables' earlier realizations in time. Therefore, using a multivariate time-series data for modeling economic process, particularly the business cycles, is justified. In addition, as Baumohl (2012, 7) argues, the U.S. economy is a very complex process and to understand it, even partially, requires using information from various economic indicators. Though, understanding it, or furthermore predicting its future movements, appears to be a task so challenging that some professionals regard it as "a black box," a function so complicated at producing outputs from inputs that it is treated as an entity whose inner workings are not revealed to the examiner. Hence, trying ANN models that are also regarded as black boxes, though slightly unorthodox in the domain of economics, seem to be justifiable when based on the prevalent theories in ML, and the previous findings that suggest using a model with sufficient modeling capabilities compared to the attributes of the underlying process.

The focus of this study is to empirically compare how the popular and prominent recurrent neural network methods can handle predicting business cycles for the U.S. economy. The emphasis is on gaining insights about their general performance, since finding the best possible ANN solution for predicting the U.S. recession is recognized as a too advanced task with the currently available knowledge and resources. The experiments are conducted using the same setup, apart from the initial parameter values, for every chosen RNN architecture to provide a fair comparison. Every RNN architecture receives the same training and testing set, where their performance is evaluated by using the latter, which included the 2007 financial crisis, a recession that only a few could foresee. Arguably, finding the model that achieves the best test set performance scores in this setting does not justify to regard this particular RNN architecture as the absolute best solution with other methods or new data. However, it might show that the suggested DL methods are able to predict business cycles and, possibly, if several prospects perform well, it is justified to say that these methods deliver prominent instruments for economic forecasting. Also, by testing different types of RNN architectures, it is possible to receive some information about the necessary model capacity for the problem and the difference in the performance between LSTM and GRU networks.

All the data handling operations, model training and examination were executed in Python 3.6 (Python Software Foundation 2019). The following Python libraries were used

for data handling: NumPy by van der Walt et al. (2011), Pandas by McKinney (2010) and Scikit-learn by Pedregosa et al. (2011). The models were created and trained using Keras library by Chollet et al. (2015), which consumed TensorFlow library by Abadi et al. (2015). To speed up the training process, a Nvidia GeForce GTX 1080 Ti Founders Edition GPU was utilized. The various graphs in this study were created by using Excel by Microsoft Corporation (2018) and Matplotlib Python library by Hunter (2007).

In the following sections, the whole setup used for creating the results is examined in more detail. At first, the time-series data is displayed, followed by an introduction of the preprocessing algorithm and its methods along with their parameters. After the data related topics have been reviewed, the selected RNN architectures are presented together with a short re-examination of the training algorithm and its hyperparameter setup.

4.1 Data

The chosen indicators for this study are issued using numerical values, making the preprocessing more straightforward. The data is gathered from various sources. The plots for the selected time-series are classified into the following three categories: economic, financial and behavioral variables, and they are displayed in their own tables 4, 5 and 6, along with other time-series specific attributes, such as its source, frequency, minimum, maximum, mean and variance value over the period under investigation. Some abbreviations are used to make the large tables clearer, which are shown in table 2. The requirements for the time-series are shown in the following list:

- it has been tracked throughout the chosen period and the whole time-series data is available,
- it is accurate and reliable,
- it gives valuable information about the underlying process, based on some economic theory or expert knowledge and
- it should cover the most fundamental aspects of the economic process, especially related to the U.S. business cycles.

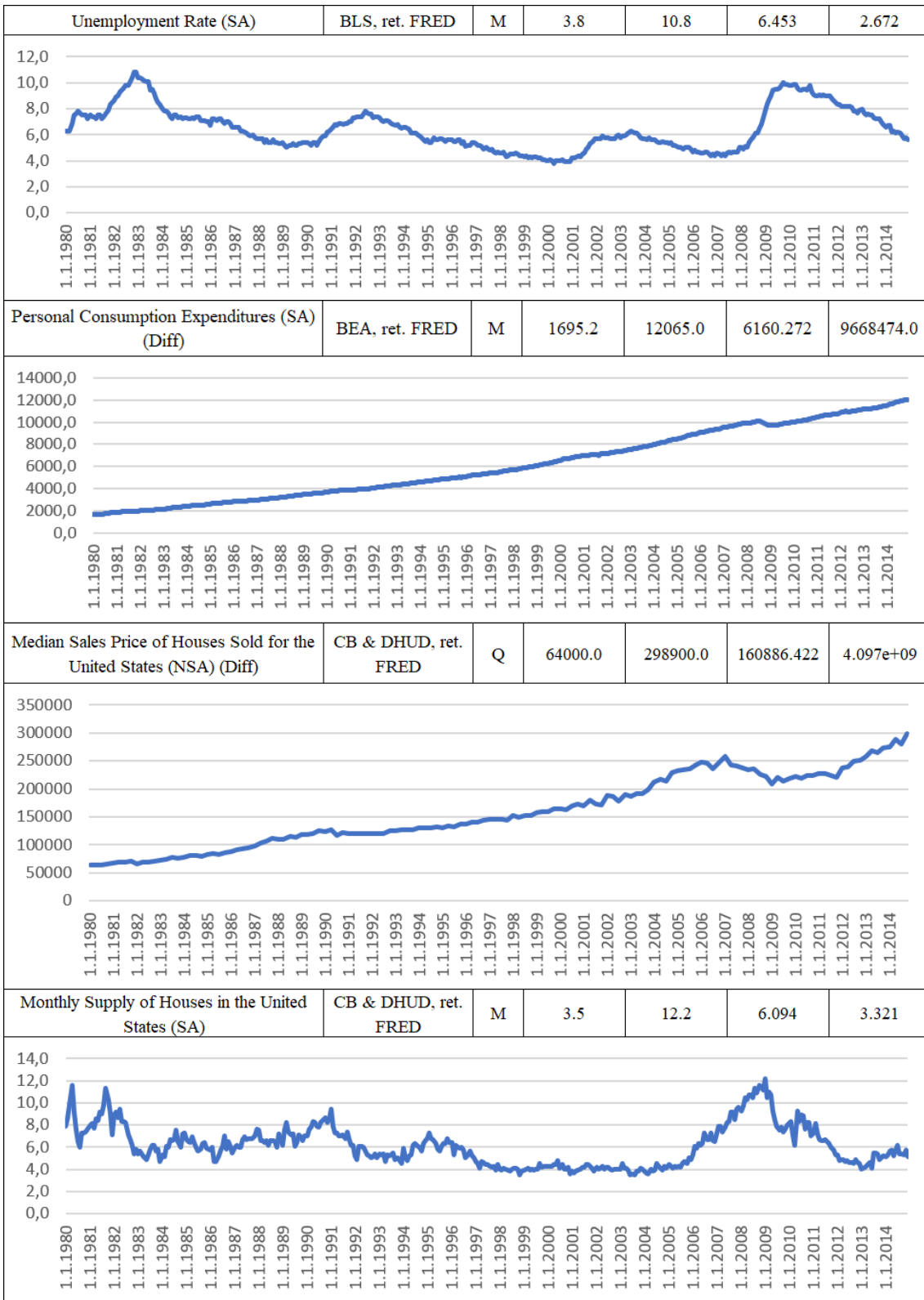
Because of the first requirement, many, possibly otherwise prominent, indicators have not been chosen. The reason for this significant decision is that an extended period of missing values would probably affect the models' performance negatively, and it is out of the scope for this study to generate some estimates to fill these large gaps.

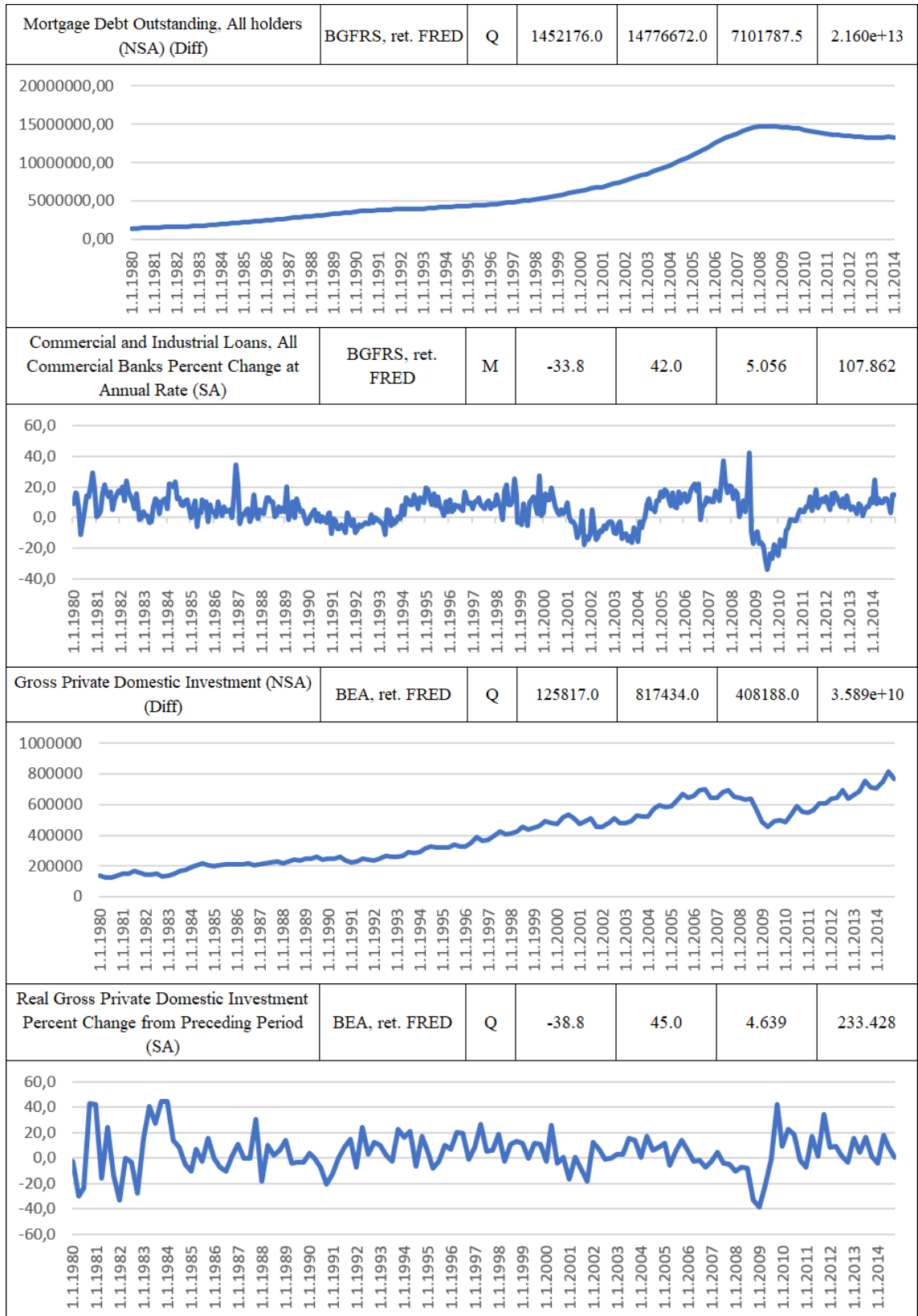
Table 2 – Abbreviations for the data tables

Data source	Abbreviation
U.S. Bureau of Economic Analysis	BEA
Board of Governors of the Federal Reserve System	BGFRS
U.S. Bureau of Labor Statistics	BLS
U.S. Congressional Budget Office	CBO
U.S. Census Bureau	CB
Federal Reserve Bank of Chicago	CF
U.S. Department of Housing and Urban Development	DHUD
Federal Reserve Bank of St. Louis	FRED
London Bullion Market Association	LBMA
University of Michigan	UM
Frequency	Abbreviation
Daily	D
Weekly	W
Monthly	M
Quarterly	Q
Others	Abbreviation
Instead of the primary source, the time-series is re-trieved from the following source	ret.
Seasonally adjusted	(SA)
Not seasonally adjusted	(NSA)
Will be differenced during the preprocessing	(Diff)

Table 3 – Economic variables

Name	Source	Freq	Min	Max	Mean	Var
NBER based Recession Indicators for the United States from the Period following the Peak through the Trough (NSA)	FRED	D	0	1	0.133	0.115
Real Gross Domestic Product Percent Change from Preceding Period (SA)	BEA, ret. FRED	Q	-8.4	9.4	2.727	8.914





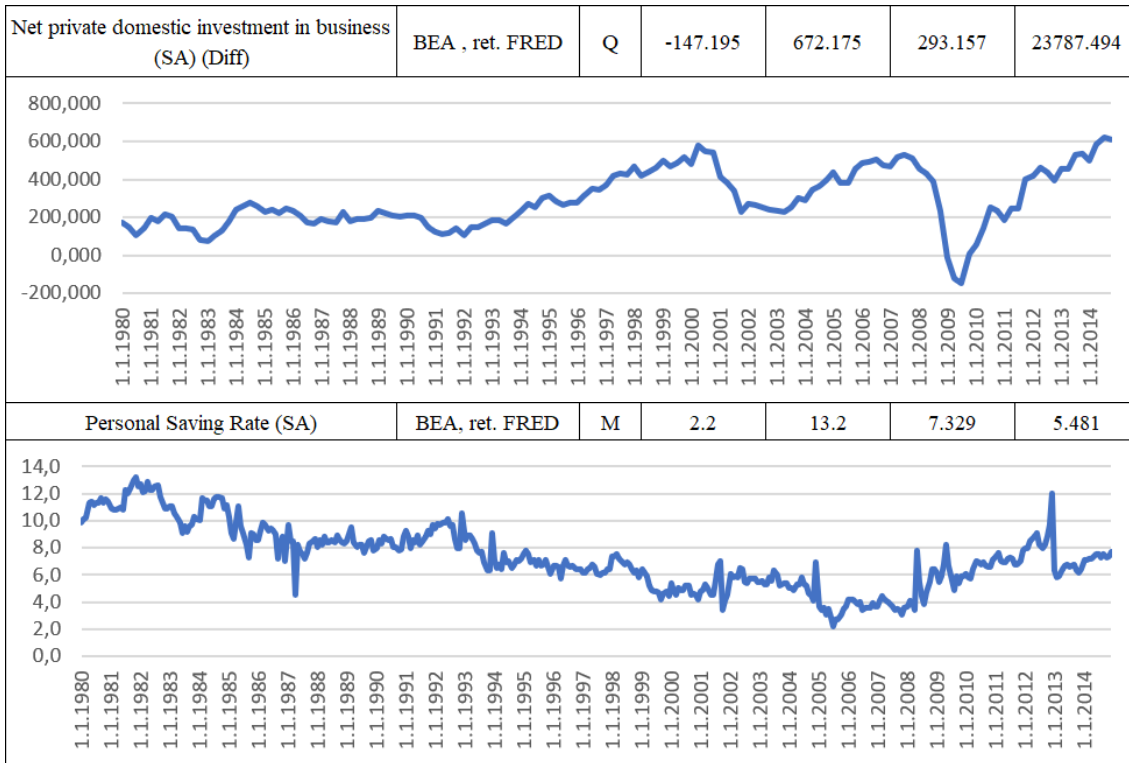
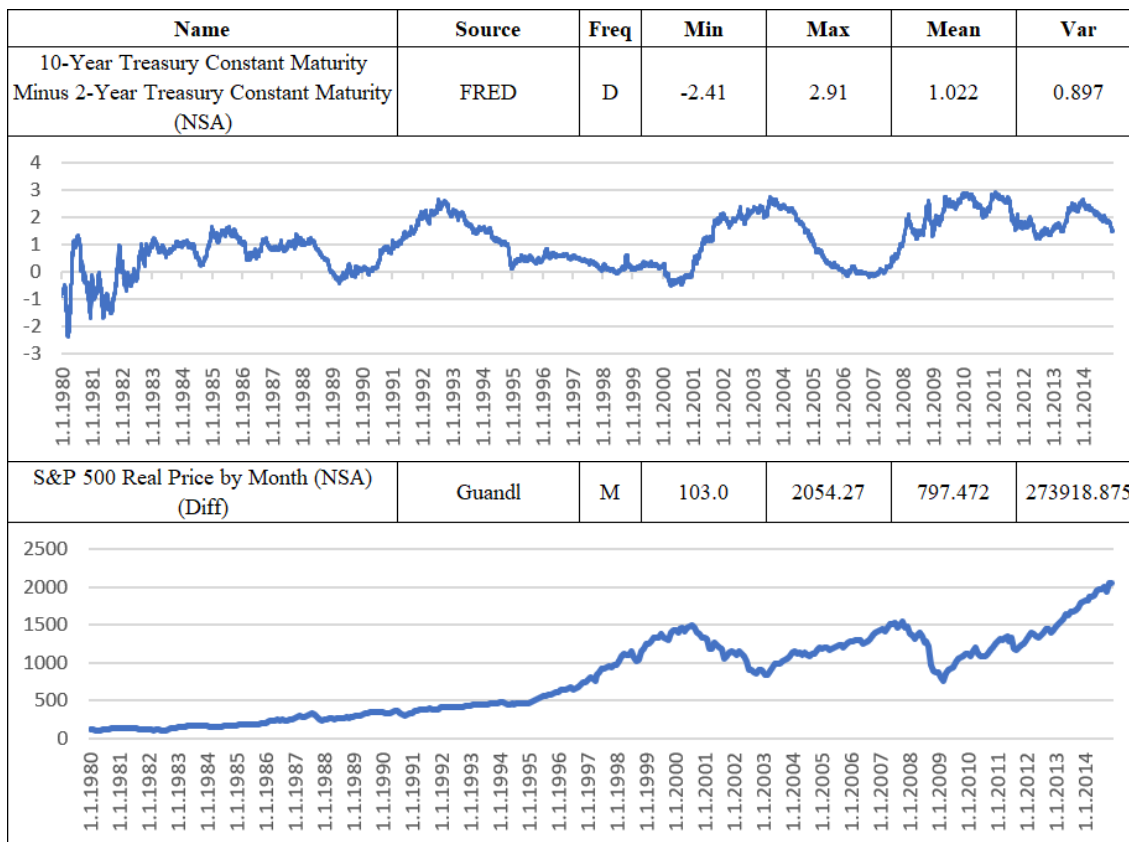
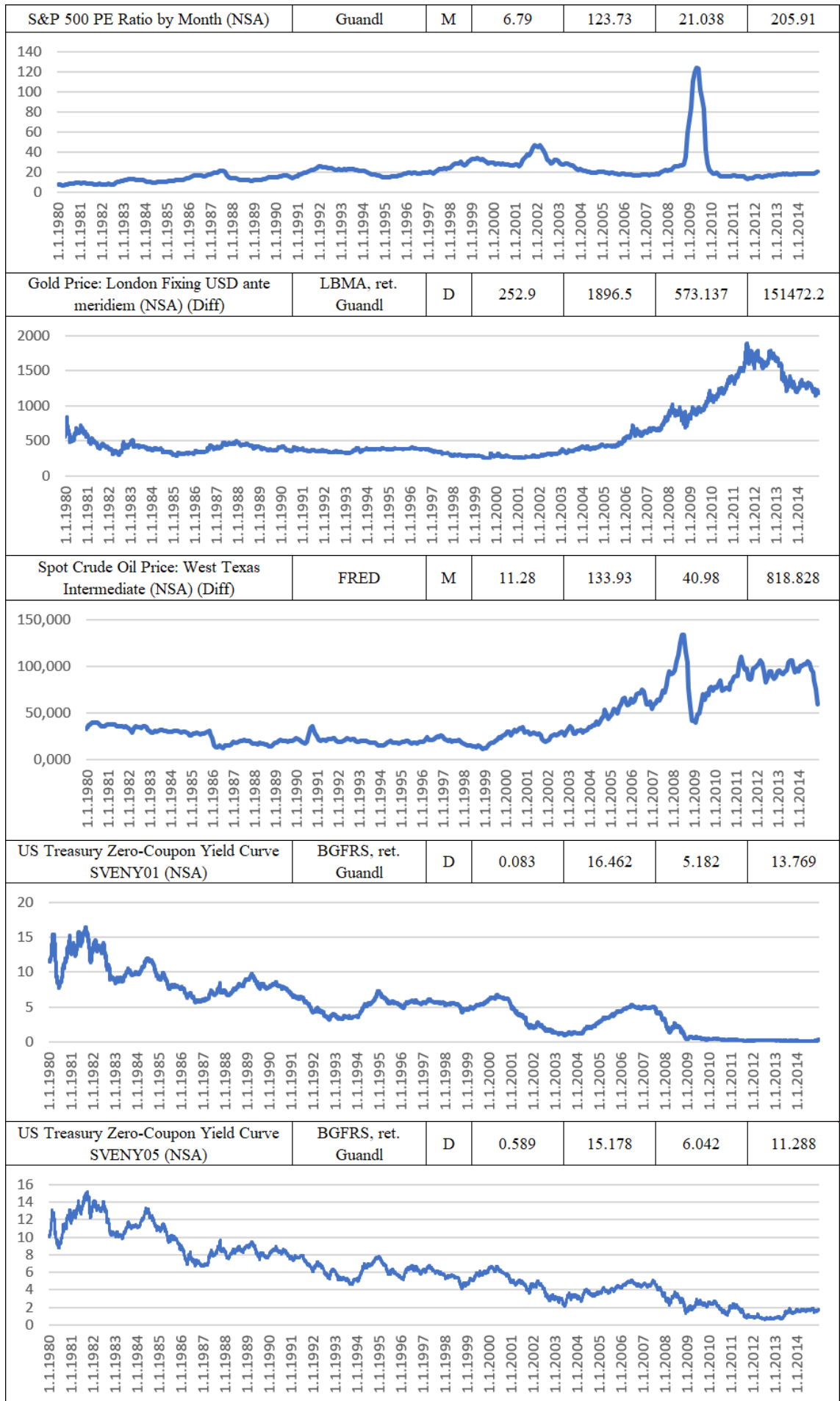


Table 4 – Financial variables





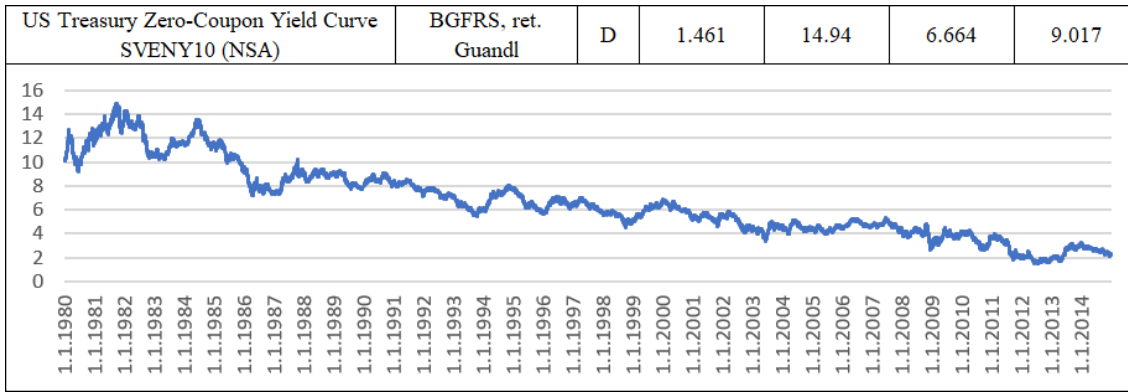
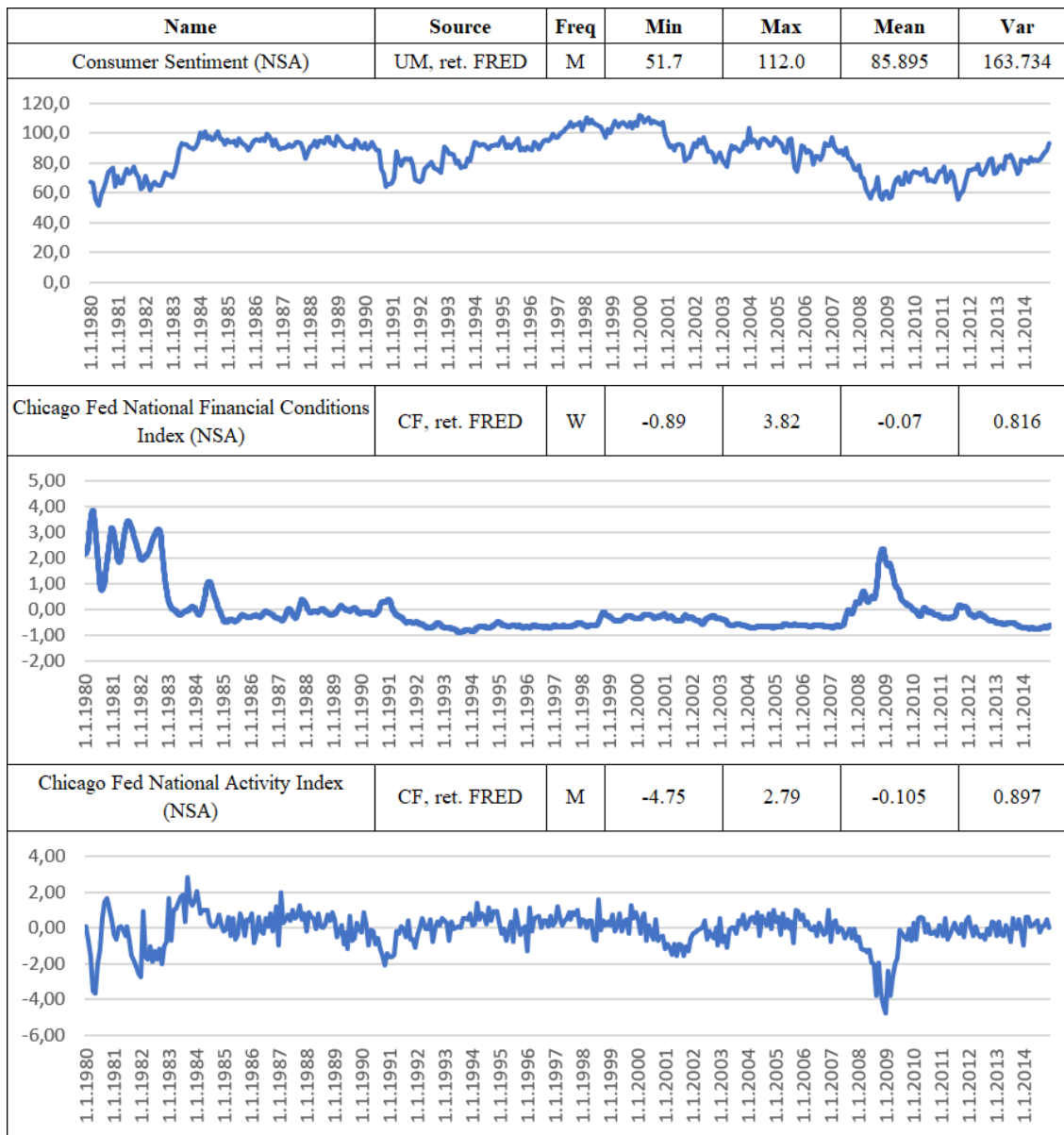
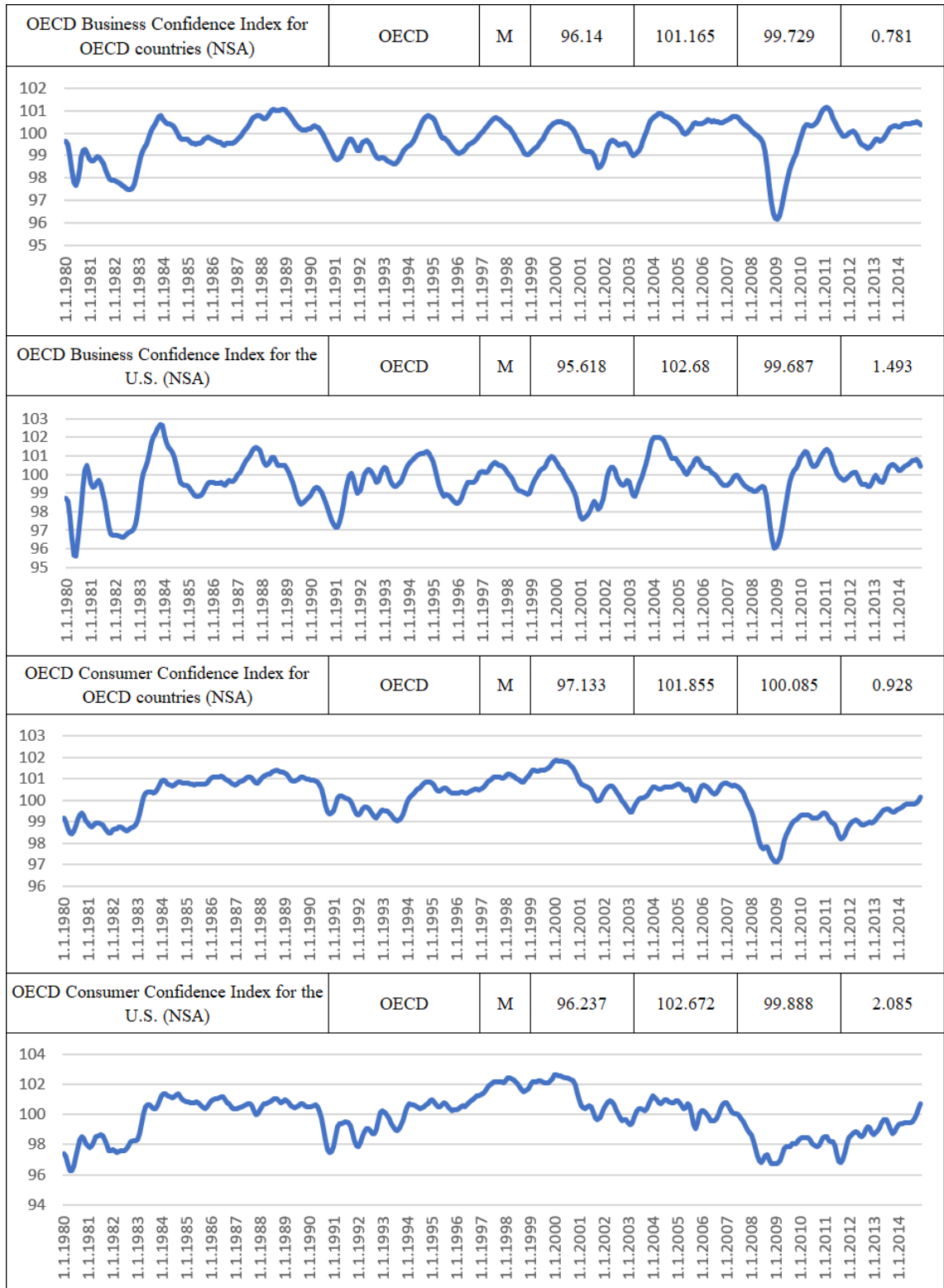


Table 5 – Behavioral variables





4.2 Preprocessing

In this section, the selected data preprocessing methods are introduced. The preprocessing algorithm, depicted in figure 21, shows the different preprocessing methods and the order of these procedures. Two aims for the preprocessing algorithm are recognized. Firstly, the data should follow the common rules and best practices in time-series analysis and deep learning. Secondly, lookaheads should be prevented by creating a preprocessing structure that simulates a real-world application, where the future information is unavailable.

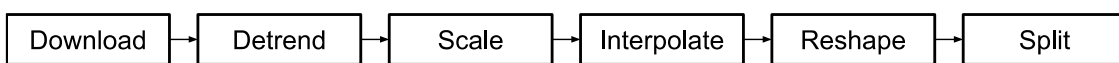


Figure 21 – Preprocessing phases

The algorithm starts by downloading the data, followed by the detrending phase, where the variables with strong trends are processed by differencing. Variable scaling, a typical phase, especially when using deep learning methods, is introduced next. After the values have been scaled, the missing values are handled by using linear interpolation. Finally, the data is reshaped and split into training and testing set for the following training and evaluation phases. The following sections introduce these steps in more detail, along with some other data-related matters and motivations.

4.2.1 Detrending and scaling

Some of the time-series used in this study contain a strong trend that should be handled by using some detrending method. The chosen method is differencing by using the previous time step. Since ANNs do not require stationary data, detrending is applied only to the time-series that showed strong trends. Differencing is executed by using Pandas Python library. The following time-series were detrended:

- Personal Consumption Expenditures,
- Median Sales Price of Houses Sold for the United States,
- Mortgage Debt Outstanding, All holders,
- Gross Private Domestic Investment,
- Net domestic investment: Private: Domestic business,
- S&P 500 Real Price by Month,
- Gold Price: London Fixing USD ante meridiem and
- Spot Crude Oil Price: West Texas Intermediate.

After detrending, the data is scaled between 0 and 1. In order to simulate a situation where a model is used during the testing set's time period and prevent lookaheads, the training set is scaled first, and then the testing set is scaled by using the training set's minimum and maximum values. The actual scaling is done by using Scikit-learn's `MinMaxScaler` function. At this point of the process, the exact split date is already decided and used for determining the two separate sets for scaling, but the actual split is executed later, after interpolation and data shaping.

4.2.2 Interpolation and reshaping

One of the most important hyperparameters when using time-series data is the number of lags provided to the model. By determining the number lags, one can affect the examples' noisiness and nonstationarity, but there is a trade-off. The longer the sequence or, in other words, the more lags per example are provided to the model, the more it can contain nonstationarity. In turn, the smaller the time window, the less there tends to be information, compared to noise. (Moody 2012, 345.) In addition to these findings, Nielsen (2019, 413) suggests that there is a positive relationship between the number of lags for prediction and input sequences. This means that the further one wants to predict, the further back one should look in history. Typically, the predictions have fewer lags than the inputs, but any common ratio does not exist for the problem and, therefore, some method-specific rationale and intuition are typically used instead.

A quarter year is a very common unit of measure in economic research and business overall and, hence, it is used as the length for the prediction sequence. To cover the calendar effects and include information concerning the long-term macro movements, the input sequence is set to include 400 lags. With these properties and the chosen time period, the data transforms into input and output data objects in the following manner.

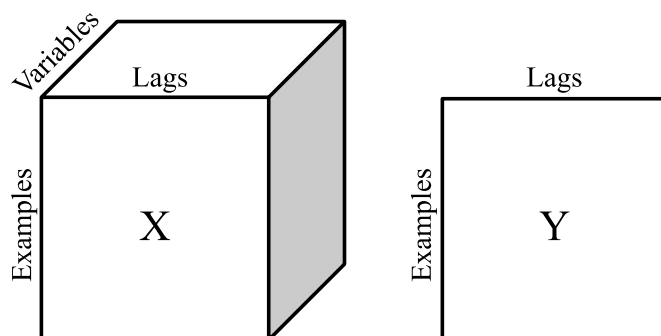


Figure 22 – A depiction of X and Y

The tensor X contains 12293 examples and each example consists of 400 lags of each explanatory variable. In turn, the array Y for the dependent variable also contains 12293 examples, each having 91 lags of one variable. As depicted in figure 22, X has three dimensions and it can be depicted as a cube, whereas Y has only two dimensions and can be shown as a rectangle.

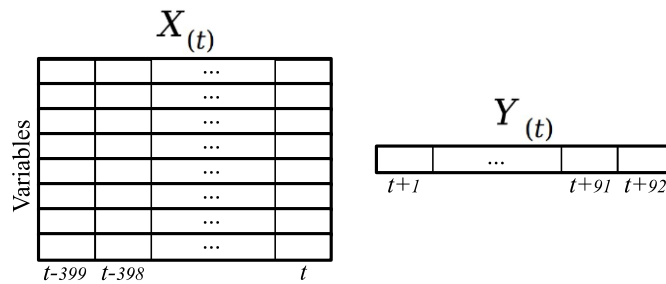


Figure 23 – A depiction of X and Y examples

Each example in X represents some particular date t and is a data matrix that contains the values for every 26 explanatory variables over the last 400 days. In turn, each example in Y forms a data vector that contains the values of the upcoming 91 days for the dependent variable, as depicted in figure 23.

```

create an empty array Y
create an empty three-dimensional tensor X
for each variable
  if dependent variable
    for each time step t
      create a sequence of the values between (t+1-interpolation buffer) and (t+92)
      interpolate this sequence
      cut out the interpolation buffer
      add the sequence into Y
  else (explanatory variable)
    create an empty array X-help
    for each time step t
      create a sequence of the values between (t-399-interpolation buffer) and (t)
      interpolate this sequence
      cut out the interpolation buffer
      add the sequence into X-help
    add X-help into X
    
```

Figure 24 – Combined interpolation and reshaping function

The linear interpolation function by Pandas is applied for handling the missing values in the data. In order to avoid lookaheads, the interpolation is combined with the data shaping function that processes each variable, one by one. This reshaping function creates either 400 or 91 lag sequences from the original time-series, depending on whether the time-series belongs to an explanatory or dependent variable, and creates the final X and Y data objects. The combined process is described in detail in figure 24.

4.2.3 Train-test split

According to Murphy (2012, 23), approximately 80 percent of the data is typically used for the training set and the remaining 20 percent for the testing set. In this study, the aim of the models is to learn to predict business cycles. Hence, the interesting phenomena in the data are the five U.S. recessions between the years 1980 and 2015. By applying the 80-20 rule, the training set should include four of the five U.S. recessions, and the last recession, 2007 U.S. sub-prime financial crisis, should be included in the testing set. The split date is set to 1.1.2005, shown in red in figure 25. In this way, the testing set also includes slightly more examples of the expansions. This is favorable since their share of the time period is considerably larger and, thus, should test better how well the models predict the more common expansion.

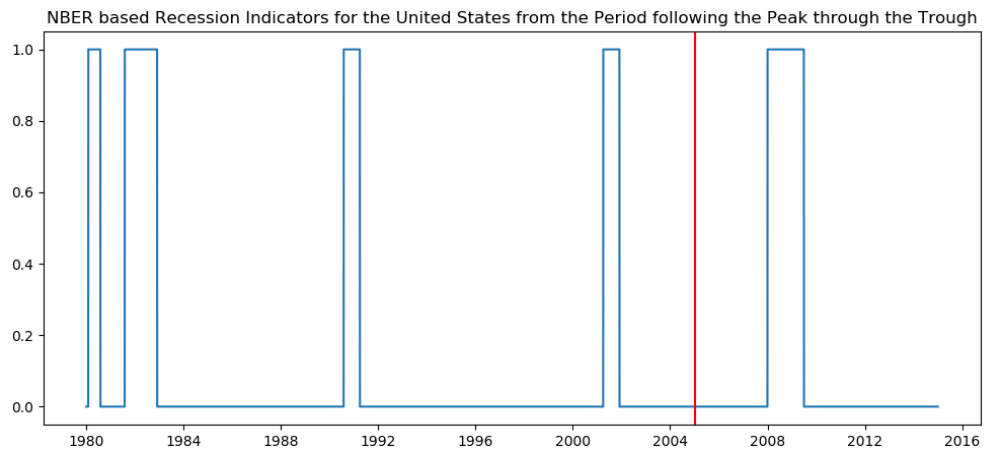


Figure 25 – The train-test split

In order to gain useful information from the train-test split procedure, the training and testing set should contain similar patterns. Over the last 40 years, the world has changed considerably, but some of the basic dynamics have stayed the same. For example, the spread between the short and long-term interest rates for the U.S. Treasury bonds appears to have produced valuable information concerning the timing of a U.S. recession over many years (Qi, 2001), and they might also continue to be prominent for this particular application in the future. By including the information about these types of indicators into the training data, the models should be able to learn the correlations between the explanatory and dependent variables and, thus, they should learn to predict the general pattern at some level. Therefore, the proposed train-test split should be adequate for testing how well the models generalize, even though the economy's evolving process includes arguably a considerable amount of noncyclical progressions.

However, using only one U.S. recession for evaluating some ANN architectures' ability to predict U.S. recessions introduces several possible shortcomings. Every recession is in some way unique, for example, depending on its cause, and in this case, the 2007 financial crisis emerged from a somewhat untypical situation comprised of lax financial regulation and moral hazard in the financial industry. Nonetheless, Reinhart and Rogoff (2008) found the 2007 financial crisis having similar properties to the previous financial crises, suggesting that the event could be predicted by using the previous cycles. However, it should be acknowledged that the proposed testing setup might be rather challenging for the models and, thus, it might yield pessimistic results concerning the models' ability to perform in the future. Although this weakness is known, the available data does not allow using more cycles without applying some advanced methods for filling a large number of missing historical values. However, applying this type of method would not serve the scope of this study, that is to produce baseline findings for future research and improvements, which could be affected too much by some unusual method.

4.3 Recurrent neural network selection

In DL, the model training process discovers some set of parameters for a given ANN model, but, typically, it does not find a desirable set of hyperparameters that defines the parameter search. The potential hyperparameters form a vast search space of different possible configurations, each introducing some effect on the parameter search. Therefore, some sort of search should be performed to find a set of hyperparameters that enable settling upon satisfying parameter values in the training process.

This study is essentially a hyperparameter search with three main research questions related to finding a suitable RNN architecture. Firstly, do these novel RNN methods appear useful in forecasting U.S. business cycles? Secondly, from the model capacity's perspective, how large architecture suits best this prediction problem? Thirdly, does either LSTM or GRU networks outperform the others in this task? The research concentrates on examining these different types of networks' effect on forecasting U.S. business cycles.

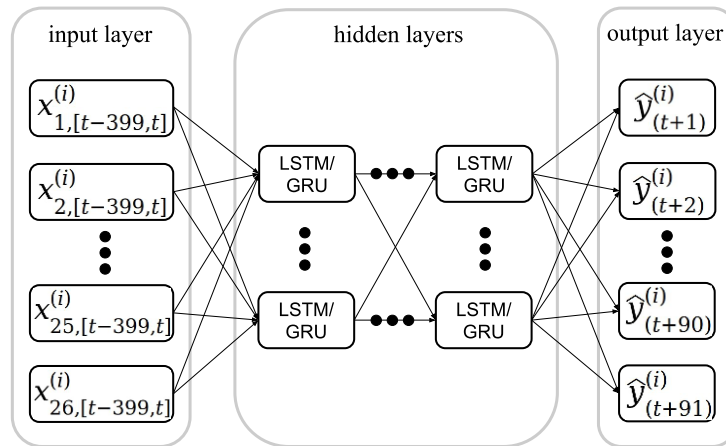


Figure 26 – Recurrent GRU or LSTM neural network architecture

As shown in figure 26, an artificial neural network typically has three different types of layers: input, hidden and output layers. In this study, the input layer receives the input values for each explanatory variable and feeds them forward to the following hidden layer one time step at a time. In turn, the hidden layers apply some arithmetic operations to transform this data and, in the end, the last hidden layer feeds forward the last hidden activations for the example at index i , that are multiplied with the last weight matrix before ending up to the output layer. The ANN has then transformed i^{th} instance's input values, that is a matrix comprising 400 sequential values for every 26 explanatory variables, into a sequence of 91 values that represents the forecast for the dependent variable for the following 91 time steps, as outlined in the following equation:

$$f(X_{t-399,t}^{(i)}, \theta) = \hat{y}_{t+1,t+92}^{(i)}.$$

The exact implementation of these methods might vary between different DL Python libraries, but the presented basic principles still apply.

The model architecture is recognized as one of the hyperparameters in DL. By keeping all the other hyperparameters constant, the hyperparameter search conducted in this study is able to put its emphasis on examining the different ANN architectures' effect on forecasting performance. In practice, this relationship is studied by training a set of ANNs with different architectures and testing their ability to generalize by using the testing set. As introduced earlier, the architecture can be changed, for example, by increasing or decreasing the number of hidden layers and the units on these layers. The intuition is that the more there are hidden layers and units, and thus also the parameters, the more the network can express different types of functions, meaning a higher modeling capacity.

As introduced in section 2.5, models with different amounts of modeling capacity should be tested. However, according to Bengio (2012, 450-451), in practice, it is fundamental to have enough units in the network and, therefore, if sufficient regularization methods are used, one should rather have them too many than too little. Since two regularization methods are used, L1 and early stopping, the conducted hyperparameter search also includes several networks that one could find unnecessarily large for the problem. Although large networks are typically recommended, small ANN architectures should still be tested. It is possible that only a few patterns in the data are very significant when predicting business cycles and, hence, using models that are capable of modeling fewer relationships in the data can give information about the problem's complexity, or even outperform the larger architectures.

Table 6 – The selected recurrent neural network architectures

	LSTM network	GRU network
1	2	2
2	4	4
3	4-2	4-2
4	8-4	8-4
5	16-8	16-8
6	32-16	32-16
7	64-32	64-32
8	128-64	128-64
9	256-128	256-128
10	256-128-64	256-128-64
11	256-256-256	256-256-256

The selected ANN architectures are shown in table 6, where the number of units or cells is declared only for the hidden layers, the most left number being the first hidden layer after the input layer, and the most right number is related to the last hidden layer before the output layer. Each unit or cell on the previous layer is always connected to all the following layer's units or cells. Long short-term memory cells and gated recurrent units

are not mixed in the proposed architectures in order to keep the number of different architectures sufficiently small. Also, to enable a comparison between the LSTM and GRU networks, they are tested with identical architectures.

These particular architectures are chosen for a few reasons. Firstly, as motivated previously, the selected architectures should cover a wide range of different model capacities, while being limited by scarce computing resources. Secondly, the range of model capacity under inspection should be traversed in a somewhat smooth manner, without leaving wide gaps between any selected architectures. Thirdly, there is no widely accepted theory for selecting the number of units per hidden layer. So, in order to traverse as wide a model capacity range as possible, also the smoothness in mind, the log scale is used to decide the network architectures. Moreover, one could use any integer as a hidden layer's width, but here the powers of two are used that can offer some pattern to the different architectures and a quite smooth parameter increase on the log scale, as depicted in figure 27.

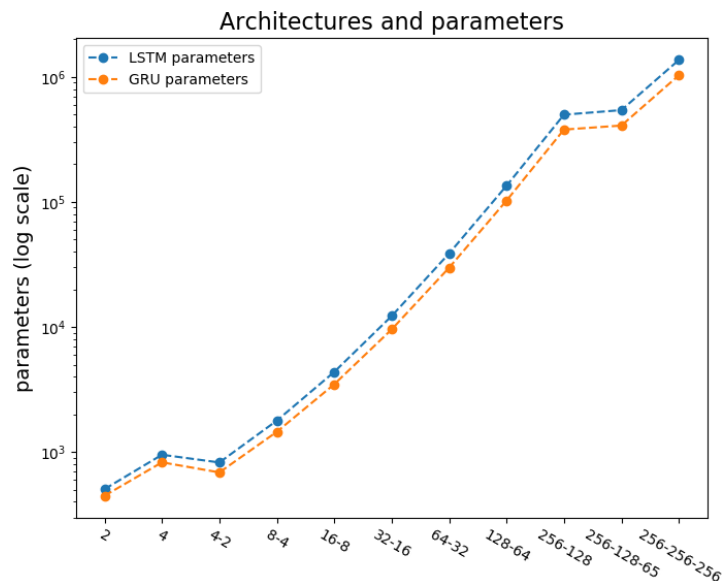


Figure 27 – The architectures and the number of parameters

The interruptions to the parameter curves in figure 27 come when an additional layer is added with fewer units when compared to the previous last hidden layer. This yields fewer connections between this new last hidden layer and the output layer than previously and, therefore, the curve appears irregular when depicted by using only the number of parameters. However, it should not be as significant a problem from the model capacity's perspective because the additional layer offers an additional amount of model capacity.

Hence, the model capacity can be thought to increase smoothly between the selected architectures.

As introduced in section 2.7.10, the initial parameter values should be small and determined by using some random distribution. This randomness scatters the results and, thus, the training and evaluation should be repeated in order to get a better understanding of the different architectures' general performances. (Graves 2012, 30.) For this reason, each architecture option is trained three times, and the emphasis is on evaluating these three models' average performance. Therefore, each architecture option forms an ensemble of three RNNs, whose predictions are the average of these three models' estimates.

During the training of the selected 66 RNN architectures, the optimization setup is kept unchanged. MSE is chosen as the loss function for training, and an L1 regularization method is introduced to drive weight decay. The L1 coefficient hyperparameter is set to 0.00001. The cost function adds the MSE loss function and the L1 regularization term together. The gradients are computed by using backpropagation through time algorithm. Nesterov accelerated stochastic gradient descent with a typical 0.9 momentum coefficient with minibatches of 32 examples are used for updating the parameter values. The learning rate is gradually decreased between the training epochs e according to the following equation

$$a_{(e)} = 0.1 * 0.95^{\lfloor e/2 \rfloor}.$$

The maximum number of epochs for each iteration is set to 200, but early stopping is executed whenever there is no improvement in the test set cost during the last 25 epochs. The last parameter values are kept after the training process. The initial parameter values are determined by using a Gaussian distribution with a mean of 0 and a standard deviation of 0.1. Now, when all the other hyperparameters than the model architecture are fixed, the emphasis can be put on examining the effect of different RNN architectures on the forecasting performance.

5 RESULTS

The results yielded a considerable amount of information about different types of phenomena. These phenomena are examined by dividing these findings into smaller sections that present their relationship to the three main research questions, or in other words, to the models' capacity, type and overall performance. First, the training processes are evaluated to assess information concerning the selected optimization methods' suitability to train the proposed models. For example, did the chosen methods and their hyperparameter values provide a sufficient environment for the different RNNs to learn to predict U.S. business cycles? This examination answers partly to the first research question, by evaluating the chosen methods' suitability for the task at hand. After the training process is evaluated, the performance-related findings are inspected in smaller segments. First, the bigger picture is examined, followed by sections that concentrate on some particular architecture class' results, along with some other significant findings that serve in understanding the process and results further. In addition, appendix 1 and 2 provide comprehensive tables of all the iteration and ensemble specific results.

5.1 Architectures and the training process

Before examining the performance-related results, one should examine how consistently the training algorithm, which is the combination of methods used for optimizing the parameter values, such as the cost function, BPTT, NAG and their hyperparameter values, could drive the proposed models towards better performance. For example, did it allow the models to find reasonably well some minima on the cost plane, thus enabling a fair and valid comparison between the proposed architectures and their types? One way of evaluating the training algorithm's suitability for the task at hand is to examine, are the findings in line with the existing theories and other reasonable expectations that are drawn from the training algorithm's characteristics and the proposed RNNs, such as the chosen cost function and the number of optimizable parameters.

One should expect that large RNNs take more epochs to train on average because there is more cost to decrease. This is due to a large number of parameters, L1 regularization and random parameter initialization. Also, large networks might introduce more variance to the training process because of a large number of epochs, that introduces more chances to face troubles on the cost plane, though the learning rate is reduced throughout the process to make the parameter updates more gradual towards the end of the process.

In turn, small networks might also introduce slightly more variation in the training process because the learning rate stays rather high throughout their typically shorter training process, making them less capable of settling to some minima on the cost plane.

Table 7 – Architectures and epochs

Architecture	LSTM epochs avg.	GRU epochs avg.	LSTM epochs var.	GRU epochs var.
2	39.667	28.334	376.334	0.334
4	32.667	35.334	2.334	162.334
4-2	34.667	29.0	6.334	1.0
8-4	50.667	29.334	114.334	1.334
16-8	33.334	28.334	85.334	2.334
32-16	35.667	27.0	21.334	0.0
64-32	41.667	37.0	126.334	300.0
128-64	52.0	32.0	76.0	61.0
256-128	70.667	113.334	94.334	4266.334
256-128-64	94.334	152.0	134.334	1828.0
256-256-256	129.334	177.334	3889.334	1541.334
Mean	55.879	62.636	447.848	742.182

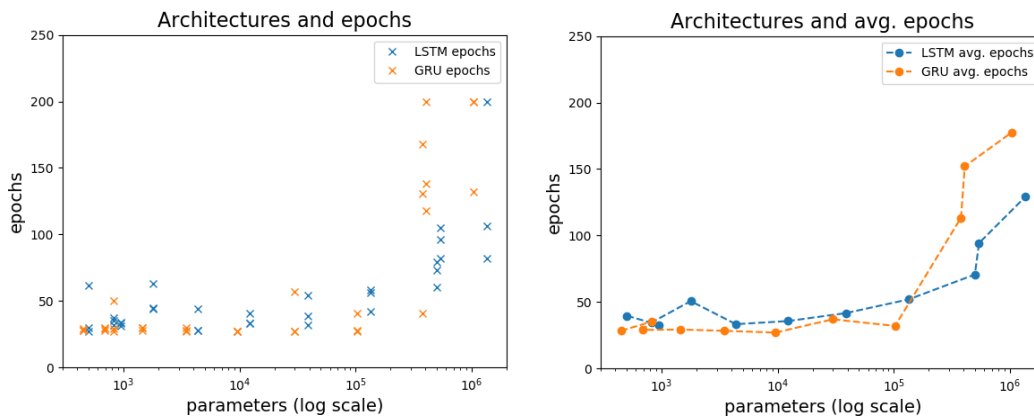


Figure 28 – Architectures and epochs

Some of these aforementioned phenomena can be verified in table 7 and figure 28. The larger networks in this study typically took more epochs to train, although only three iterations reached the maximum of 200 epochs. They also introduced more variance in the training duration when compared to other options; the three largest architectures had five of the ten highest epoch variance scores. In turn, the smaller networks took considerably fewer epochs to train. To support the assumptions related to the smaller networks’

problems to settle during the training process, the results show that the three smallest architectures accounted for two of the ten highest epoch variance scores, though in contrast, most of them offered very low values. This indicates that, while some variation among the smaller models' training process occurred, it appears overall rather mild and, therefore, the training algorithm seems to have suited the smaller networks satisfactorily. This is also verified in the following sections that show that the smaller architectures were able to deliver similar performance to the larger networks.

When comparing the two different techniques, LSTM and GRU, one can find that the GRU networks showed much more variance on average when the number of epochs is examined. Also, while the small and midsize networks using GRUs seem to have consumed fewer epochs during the training process than the matching LSTM networks, the largest GRU networks used remarkably more epochs than their LSTM counterparts. At first, this difference appears out of place, but, as the further examination shows, it seems justifiable if LSTM networks are considered to possess more modeling capacity or demand more regularization than their GRU equivalents. Hence, they would be more prone to overfit the data, which, in turn, would have yielded shorter training processes due to early stopping.

These findings suggest that the chosen training algorithm seems to have suited the task at hand, but did it offer a level playing ground for all the proposed architectures? It is good to acknowledge that, ideally, each network should be trained by using a tailored training algorithm that would achieve the best possible performance of that particular network for some particular problem. However, as is traversing all the hyperparameter options recognized impractical or even impossible with the current computing costs, so is finding the best possible training algorithms for each network architecture. Hence, one has to settle for less, and here, the focus was more on the general results than on finding the absolute best architecture for the problem. The chosen training algorithm satisfied this need by allowing a basic comparison between the proposed network architectures and, also, it appears to have offered valid results that can be used in future work.

As the following examination shows, the midsize LSTM architectures offered the best performance when measured with the test set MSE and MAE, but the other networks were not far behind. This suggests that the playing field was quite even between the different models, though some of the networks showed unideal behavior, such as overfitting and unstable training, that might have been possible to avoid by using a different training algorithm.

5.2 Model capacity and performance

The estimates of the generalization error for the proposed RNN models are depicted in figure 29, where the model capacity is represented on the x-axis by the number of optimizable parameters. The log scale is chosen in order to make the graphs easier to interpret. On the y-axis, the chosen error measure is displayed, which is either MSE or MAE for the testing set. On the left side graphs, the results for each iteration are shown to illustrate how much variation exists between all the 66 iterations and each architecture. On the right side, the average values over all the three separate iterations for each architecture are pictured. These two graphs show the relationship between the generalization error and the model capacity. According to the bias-variance trade-off theory, the relationship is typically an upward opening parabola if under- and overfitting exists. The data pictured by these graphs is also available in tables 8 and 9 for more accurate inspection.

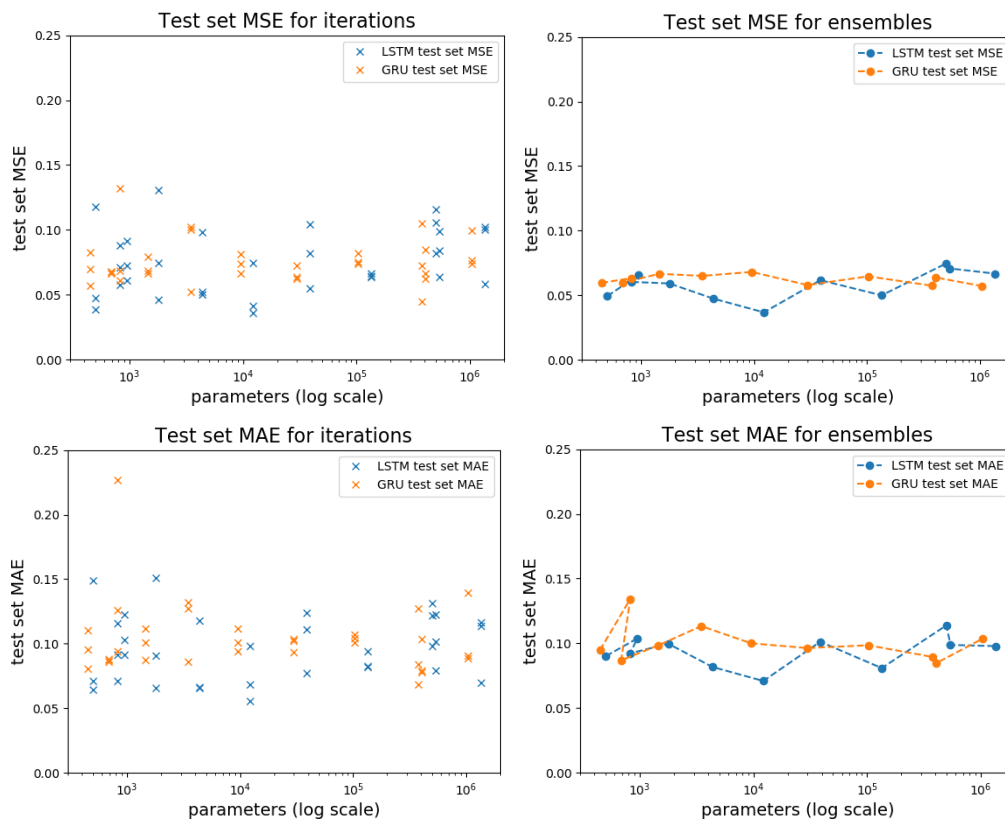


Figure 29 – Errors for single iterations and ensembles

MAE yielded higher error values due to reasons explained in section 2.7.6.1, but also because it was not included in the cost function, unlike MSE. Hence, the models ended up concentrating on minimizing MSE instead of MAE, that generated slightly differently behaving models than what would have been achieved by using MAE. Now, when using

MSE instead of MAE, the models were encouraged to output more early warning signs. Because some of the models ended up offering more early warning indications than others, MAE showed slightly more variance between the separate iterations and the average values over the three iterations for each architecture. Also, when measured with MAE, one outlier appears in the result data, which was not as striking when measured with MSE. This one particular model was very different from the other models since it did not predict any cyclic motion. This distinctive model is examined in more detail in section 5.4.

When observing the general pattern in the graphs, the GRU networks appear to have offered quite similar average performance with each proposed architecture option, unlike the LSTM networks, that offered the best performance with the three midsize architectures: 16-8, 32-16 and 128-64. This finding suggests that these midsize LSTM models were able to offer the most suitable model capacity for the task at hand with the current setup. As is examined in more detail in section 5.5, the large LSTM networks showed signs of overfitting that might explain the slightly worse performance. In turn, even the largest GRU-256-256-256 model did not show signs of overfitting in the training graphs. Hence, it is possible that the three largest GRU architectures were more robust against overfitting, which could also explain why they were able to offer generally better results than their LSTM equivalents. Otherwise, the LSTM architectures could offer on average similar or even slightly better performance than their GRU counterparts.

When examining the GRU networks, it appears that the largest networks may not have overfitted and even the smallest networks seem to have handled the task quite nicely when measured with MSE and MAE. Therefore, the estimated generalization error curve for the GRU networks did not resemble a curve all that much. On the other hand, when examining the LSTM networks, the three midsize architectures: 16-8, 32-16 and 64-32 offered better performance than the smaller ones, and the larger LSTM architectures could not take advantage of the additional model capacity and, instead, ended up overfitting the training data. Hence, the proposed upward-opening generalization error curve seems to exist for the LSTM architectures. This finding suggests that LSTM cells might offer more modeling capacity than GRUs. It can possibly be explained by the fact that the GRUs are simplifications of the LSTM units and they have slightly fewer parameters, but this difference has not been verified in the previous studies. Instead, they have typically fared somewhat similarly (Greff et al. 2017; Jozefowicz et al. 2015).

Table 8 – Architectures and MSE for ensembles

Architecture	LSTM MSE avg.	GRU MSE avg.	LSTM MSE var.	GRU MSE var.
2	0.049334	0.059608	0.001863	0.000163
4	0.065772	0.063266	0.000234	0.001547
4-2	0.060348	0.059747	0.000234	0.000001
8-4	0.059149	0.066610	0.001875	0.000050
16-8	0.047388	0.064997	0.000743	0.000807
32-16	0.036788	0.068064	0.000428	0.000056
64-32	0.061680	0.057906	0.000617	0.000028
128-64	0.049957	0.064641	0.000002	0.000021
256-128	0.074477	0.057347	0.000295	0.000894
256-128-64	0.070708	0.063725	0.000314	0.000140
256-256-256	0.066753	0.057201	0.000624	0.000203
Mean	0.058396	0.062101	0.000657	0.000355

Table 9 – Architectures and MAE for ensembles

Architecture	LSTM MAE avg.	GRU MAE avg.	LSTM MAE var.	GRU MAE var.
2	0.090180	0.094714	0.002221	0.000221
4	0.103717	0.134354	0.000243	0.004808
4-2	0.092278	0.086870	0.000499	0.000001
8-4	0.099685	0.098398	0.001916	0.000148
16-8	0.081731	0.113291	0.000885	0.000644
32-16	0.070914	0.100011	0.000480	0.000077
64-32	0.101216	0.096346	0.000572	0.000032
128-64	0.080880	0.098514	0.000048	0.000008
256-128	0.113900	0.089432	0.000291	0.000937
256-128-64	0.098769	0.084791	0.000476	0.000205
256-256-256	0.097856	0.103426	0.000688	0.000824
Mean	0.093739	0.100013	0.000756	0.000719

The LSTM ensemble models achieved lower MSE and MAE values on average, but they typically introduced more variance between the three iterations of each architecture than

the GRU counterparts. However, the gap between these two methods' performance was not wide and, in fact, they typically included similar behavior in their predictions.

MAE penalized more from deviations from the real values than MSE. These deviations can be either useless or useful, based on how well it gives information about the future values. When examining the graphs in figure 29 or tables 8 and 9, it appears that the GRU networks' predictions varied more from the real values. Moreover, based on the MAE results, the GRU networks were slightly weaker than the LSTM networks, but because useful early warning signs are recognized as an advantage, the comparison cannot be decided by using only the MAE and MSE scores. These deviations could either be noise, false predictions or some useful information for early warning purposes, possibly suggesting that a probability for a peak or a trough to happen has risen. However, the quality of these deviations can only be determined by viewing the actual predictions, as is done in the following sections.

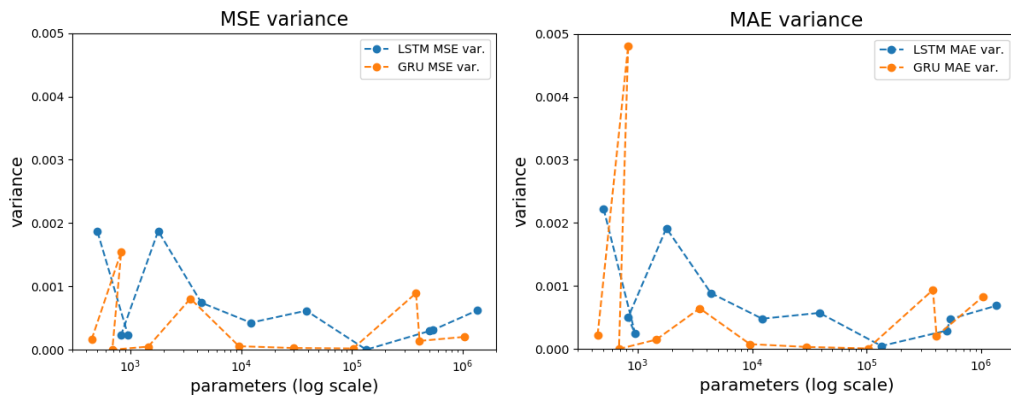


Figure 30 – Variance per architecture

These performance-related results appear to have correlated with the previously examined training durations. Commonly, the more problems occur during a training process, the more its error measures oscillate during it, and, due to the early stopping, the training process' length can vary significantly between the iterations of the same architecture. This variation typically had an effect on the trained models' performance. The results show that, the more there was variation in the number of epochs for some particular architecture, the more there seems to have been variation between these iterations' performance. However, as can be found from the graphs in figure 30, the smaller LSTM networks' performance between the iterations of each architecture varied more when compared to the larger LSTM networks. Whereas, apart from the outlier, the GRU networks introduced

the opposite behavior, where the performance results between the iterations of each architecture varied more among the larger GRU networks than the smaller ones. Because the GRU models offered relatively good results also with the larger architectures, it appears that much of the higher variation between the performance was not due to actual problems during the training process. Instead, it might be related to where on the cost plane the training process ends. Because of the longer training processes, the larger GRU networks could have moved more on the cost plane and, thus, eventually ended up on very different local minima. This, in turn, would have generated very different parameter values and, hence, different behavior in the predictions.

When compared to their LSTM counterparts, the large GRU networks introduced notably longer training processes. This was due to the large LSTM networks' tendency to overfit, that, together with the applied early stopping method, made the training processes stop sooner than their GRU equivalents, which did not show significant overfitting effect, as is examined in more detail in section 5.5. In addition to their shorter training processes, it appears that all the three largest LSTM architectures' iterations had somewhat similar training durations with each other and, hence, the variance between their performance is not striking. However, even though the large GRU networks' results varied more than those of the smaller GRU networks, their variance in MSE and MAE was similar to the large LSTM networks.

Apart from the three largest architectures, the GRU networks showed, per architecture, typically less variance between the training duration and performance of each iteration when compared to the LSTM networks. The training duration was similar between small and midsize architectures of both types and, therefore, it is not a probable explanation for this phenomenon. Moreover, as was also suggested earlier, if the training process is short, the learning rate does not receive low values, making settling to some minima on the cost plane difficult. Instead of settling, the parameter values might continue to move arbitrarily on the cost plane and end up with an uncommon parameter setup. However, it does not explain why mostly the small and midsize LSTM networks appear to have suffered from this type of phenomenon and, thus, is assumedly not a suitable theory. As is found in section 5.4, even the GRU outlier, which also had a small architecture, most likely did not suffer from this type of defect in the training algorithm. To point out, even the best performing architectures, the midsize LSTM networks, produced more variance than their GRU equivalents. Also, the random parameter initialization in the training pro-

cess might have generated some untypical results, but the difference between these networks appears quite systematical and, therefore, it was probably not created by chance. No satisfactory explanation was found for why GRU networks were more stable in general than their LSTM counterparts. This suggests that more work should be done, for example, in testing different hyperparameter values and other methods related to the training algorithm to find if the cause is in the selected training setup.

So, it appears that, commonly, the more there was variation between the iterations' training durations, the more their performance varied. In addition to this finding, the large GRU networks showed that the longer training processes also introduced additional variance to the performance. However, the small LSTM networks seem to have suffered more from performance-related variance, possibly because the training process did not reach small learning rate values, but this cannot be verified because this behavior did not appear as significant with the small GRU networks. Furthermore, the reason for why the LSTM networks appear to have varied more than the GRU networks remains unclear. All the proposed GRU networks seem to have offered quite similar performance. In turn, the midsize LSTM networks generated typically better performance than the small and large LSTM networks.

5.3 The performance of the small architectures

Surprisingly, the small networks were able to find relatively good parameter values, even though the training process was typically quite short and, hence, the learning rate had only relatively large values. The training process evolved quickly because the learning rates were large and the cost from L1 regularization was small. With the three smallest architectures, the training process typically lasted less than 35 epochs. Because the early stopping method introduced an extra 25 epochs after the smallest test set cost was achieved, the actual number of needed epochs was close to ten. After that point, it could not improve the accuracy of the model. Instead, it drifted away from the lowest value.

The small GRU and LSTM networks fared quite differently in this study, as seen in tables 8 and 9. Where the small GRU networks could offer relatively stable performance values for the test set, apart from the aforementioned outlier, the LSTM equivalents seem to have altered more from each other when measured with either MAE or MSE. Although there is a notable difference when examining the performance-related variance, the actual estimates for the test set appeared quite similar.

The graphs in figure 31 show the average estimates from all the three networks of each architecture option. All the forecasts for each time step in the testing set are shown in one graph on top of each other. The color cyan is used for illustrating the estimates, where the more estimates are located in some area, the darker the color. The red color represents the real values. Apart from the GRU-4 architecture's results, all the networks could identify the recession and its ending quite well. The biggest differences came from the number of early warning signals before the recession. The less there were notable early warning signs, the smaller was the MAE error for that particular RNN architecture, as shown in table 9. When comparing these six architectures' average estimates, GRU-4-2 seems to have been the most stable before the recession, and it also received the lowest MAE score in this group. In turn, LSTM-2 networks could offer the smallest MSE value.

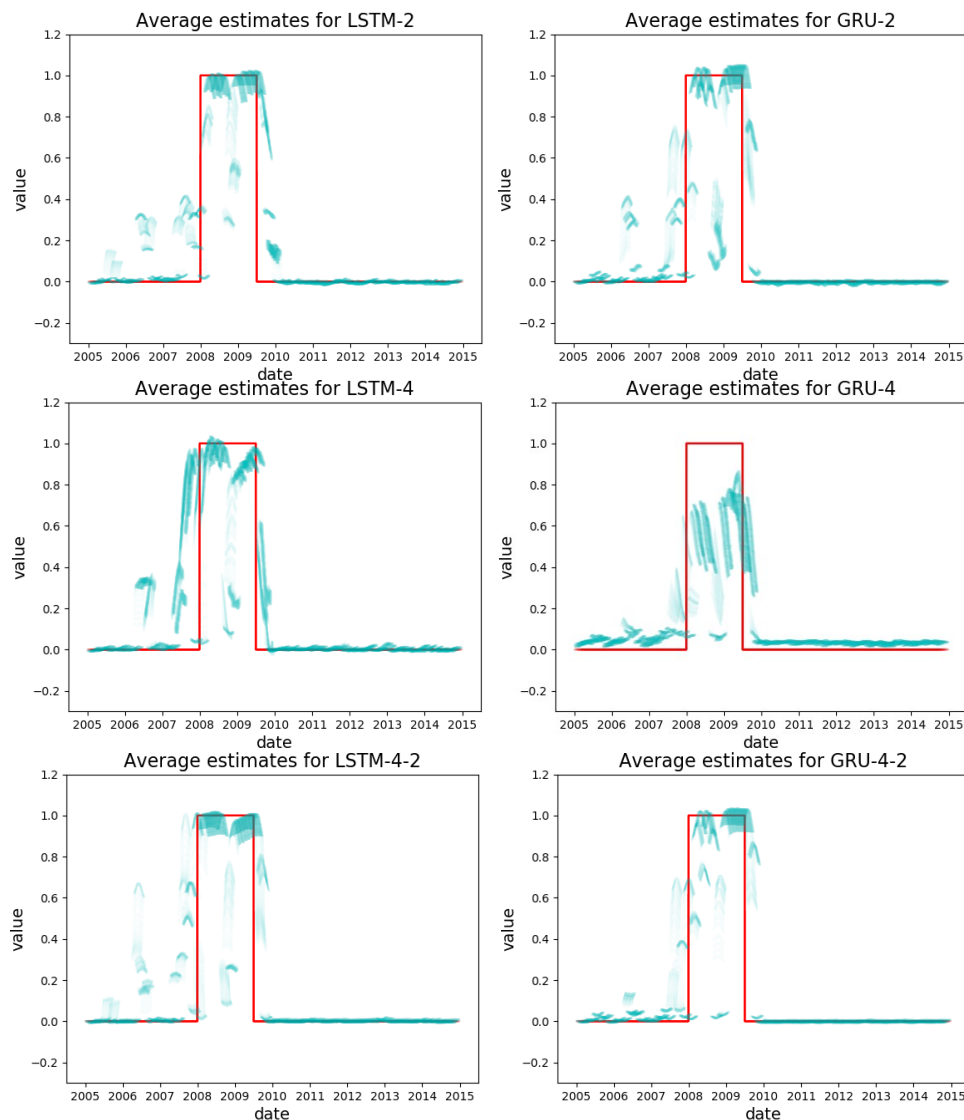


Figure 31 – The average estimates for the three smallest LSTM and GRU architectures

It was common for all the different architectures, small or large, to show some early warning signs before the recession. Interestingly, many of these models forecasted a peak between 2006 and 2007. One possible explanation for this behavior comes from a dip in S&P 500 during that time period, along with some other possible triggering conditions, such as decreasing OECD Business Confidence Index for the U.S. and a low yield spread. When comparing the forecasts before and after the recession, one can find that the models' predictions varied typically considerably less after the recession. This indicates that the probability of a recession to occur was very low for the U.S. during that period, which is more or less correct. Furthermore, this finding suggests that these models did not suffer much from pure noise, and the early warning signs are due to relevant patterns in the data.

In addition to low noise and notable early warning signs for the 2007 peak, it was common for all the networks to forecast the recession to end slightly before 2009, which could be classified as a clear mistake. This defect might be due to the fact that the previous recessions in the training data typically lasted less than ten months. In turn, the recession in the testing set lasted for 18 months, that was unprecedented for the models and, thus, could explain why they predicted it to end so early. However, after this common blunder, all the networks continued to forecast a recession, sometimes even slightly longer than necessary. The midsize and large networks could generally signal slightly better when the recession ends than the smaller ones.

When the shape of the predictions is considered, the small networks' forecasts tended to have a strong curve instead of a straight line, which is rather perplexing when predicting a binary time-series. No definite explanation for this type of behavior is offered. One possible reason is that the smaller networks were able to concentrate only on a limited number of phenomena in the data and, hence, they formed their forecasts from a smaller set of concepts that were only enough to position and twist some preferred shape. In turn, the larger networks could create more sophisticatedly shaped forecasts, though they had problems positioning them, as is shown in figures 35 and 36. However, this theory cannot be verified with the available results.

5.4 The third GRU-4 iteration

The worse iteration in this study was the GRU-4 architecture's third iteration, whose estimates for the testing set and the cost curves from the training process are shown in figure 32. It appears that the training got stuck at a local minimum, which made the model predict only the values between 0.100 and 0.111. Interestingly, these values are close to the

recessions' share of realizations in the training set, which is approximately 12.667%. This indicates that a cost plane can include local minima that make an ANN able to fit the data similarly to some other model options, here a straight line. To answer how this particular network's parameters ended up having these particular values, one can examine the cost curves depicted on the right side in figure 32.

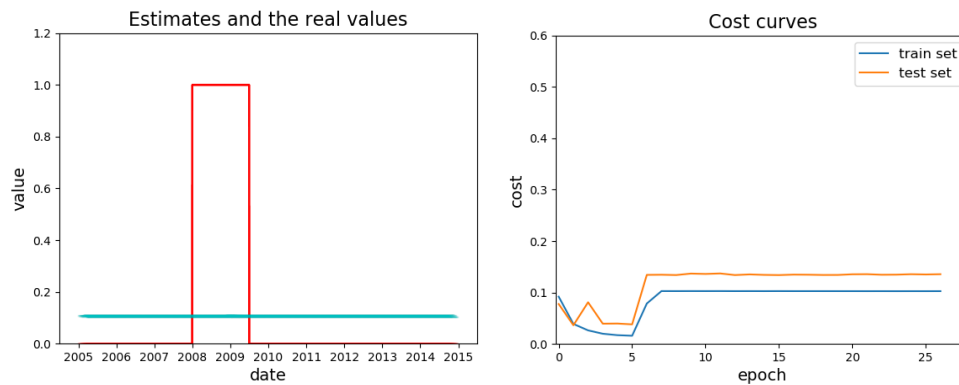


Figure 32 – The third GRU-4 iteration

It is possible to derive from the cost curves that the training process happened to evolve the model parameters to a direction where it ended up stuck at some high-cost area on its cost plane. This incident is probably not due to an especially steep shape on the cost plane because that would have yielded very high gradient values, that, in turn, would have continued to show in the following gradients due to the momentum term. What the cost curves indicate is that, after the rise between epochs five and seven, the process did not vary much, which suggests relatively steady gradients. The training process got stuck immediately after climbing on top of the high-cost area, suggesting that the process missed the steep hill and ended up straight on top of this particular area.

As is shown in appendix 1, MSE generated significantly less error than MAE for this particular model. When considering the training process, the smaller error also meant smaller cost and, hence, less significant gradients that were not enough to move out from the undesired location on a cost plane. Therefore, the use of MSE can be questioned, and testing MAE as the loss function might be desirable in future work.

Although this result is exciting, it is far from ideal when trying to predict business cycles. These unorthodox predictions yielded, not only very high MSE and MAE values for this particular iteration, but also a significant negative effect on the GRU-4 networks' combined performance, as seen in figure 30 and the performance-related tables 8 and 9. This exceptional outcome also makes interpreting the general pattern less convenient. Moreover, nothing similar exists in these results. Several heavy oscillations in the test set

performance occurred during a few training processes, that made them stop at an unideal point with unideal parameter values, but also in these cases, these networks were able to find the strongest patterns, though being often too pessimistic and predicting a peak too early. Also, when these networks were combined with one or two successful iterations, the ensemble fared reasonably well and the outcomes are somewhat in line with the related theories. In contrast, the GRU-4 ensemble could find the general pattern, thanks to the other two networks, but it was severely distorted. For this reason, its test set MSE and MAE received high values, that did not reflect this particular architecture's true performance accurately, suggesting to regard it as an outlier.

5.5 The large architectures and overfitting

As shown in the graphs in figure 33, all the GRU architectures achieved a somewhat similar performance apart from the outlier, and no significant under- or overfitting is found. On the other hand, when examining the LSTM networks' results, more signs of the bias-variance trade-off appears. The smaller LSTM networks with less modeling capacity tended to be outperformed by the midsize LSTM networks, and after some point when the network size was increased enough, the performance began to drop due to overfitting.

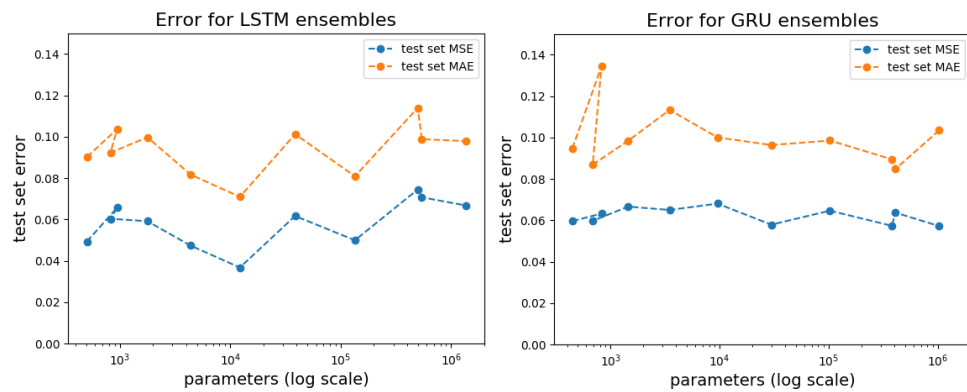


Figure 33 – Model capacity and fit

Smaller networks' weaker performance was probably mostly due to the underfitting effect that was caused by insufficient model capacity, it may also have been caused by an unideal training algorithm, as suggested previously. Therefore, unfortunately, more experiments should be done before a clear understanding of the underfitting effect can be obtained.

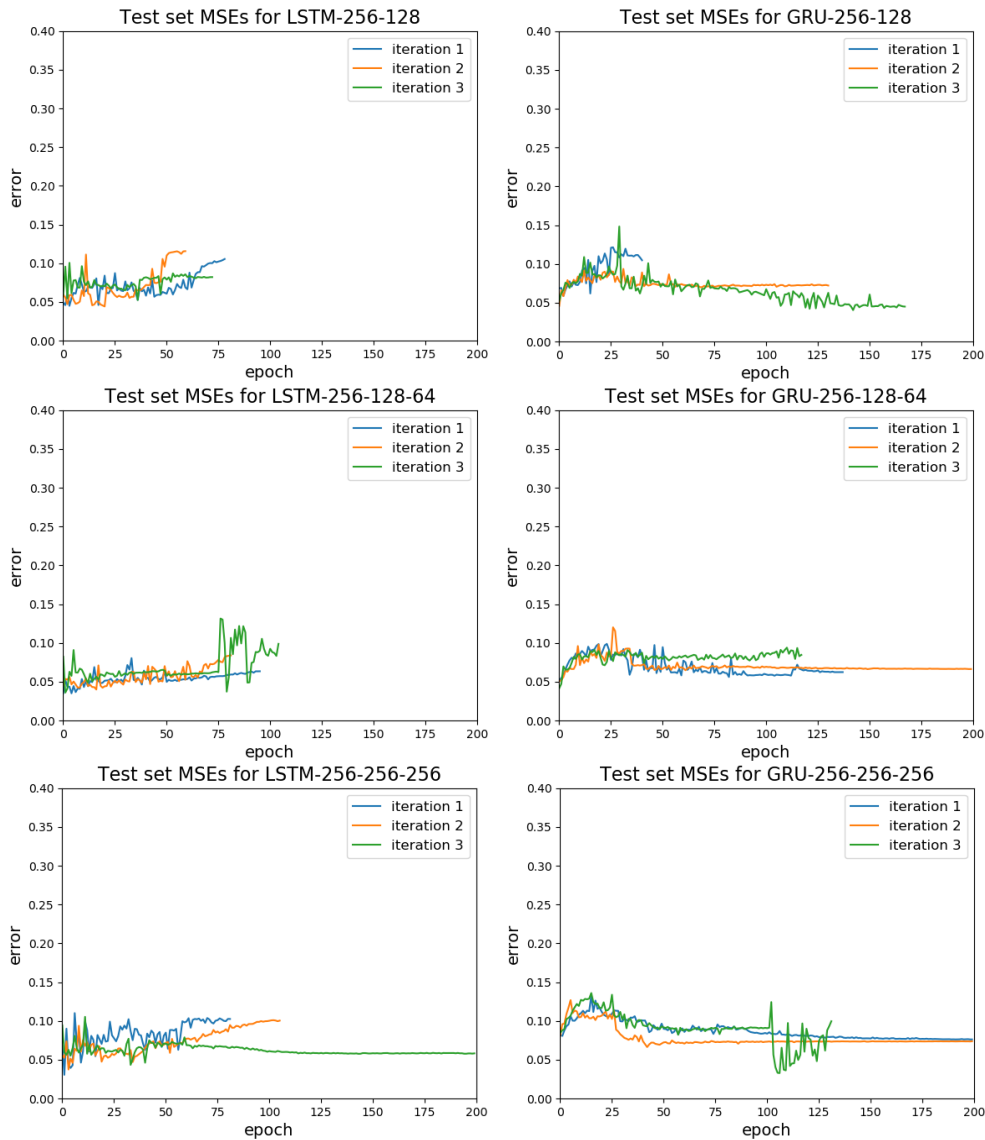


Figure 34 – Test set MSE over the training epochs the three largest LSTM and GRU architectures

The overfitting effect can be assessed by examining the large networks' training graphs. In figure 34, the evolution of the test set MSE during all the three networks' training process for each of the three largest LSTM and GRU architectures are shown. When comparing the chosen training graphs between the three largest LSTM and GRU architectures, there seems to have been two major differences. Firstly, the large GRU networks generated longer training sequences and, secondly, the test set MSE curves for the LSTM iterations tended to increase towards the end of the training process. As proposed earlier, the second phenomenon causes the first one to occur due to early stopping. Also, a rising error graph is essentially a strong indicator of overfitting. Although it does not apply to every LSTM iteration, it is common enough to show in the average performance scores.

In addition to the findings related to overfitting, one can notice that the third training iterations of GRU-256-256-256 and LSTM-256-128-64 faced problems during the process. These types of oscillations indicate that the cost plane includes strong shapes that might affect the training significantly, as depicted in figure 15. A steep edge on the cost plane may move the training process considerably on the cost plane if the gradient values are not limited. Limiting the gradient values, a method called gradient clipping, would have made the training process more robust against these types of heavy oscillations. In this study, the momentum dampens this effect, but, apparently, it was not enough to make it inconsequential.

The large LSTM networks that suffered from overfitting should be slightly too sensitive to the changes in the data. When examining the average predictions of these proposed large architectures in figure 35, one can find that the overfitting effect on the predictions was not very significant when compared to their GRU counterparts. For example, during the expansions, the large LSTM networks' predictions were generally as close or even closer to zero than the equivalent GRU networks' predictions. Furthermore, both methods introduced early warning signs extensively. Interestingly, the clear early warning signals of the GRU-256-256-256 networks were mostly generated by the third network, that suffered from severe oscillations during the training process and, thus, ended up unideal. Moreover, the networks that did not have a smooth training process were typically more pessimistic, predicting recessions more than necessary. See appendices 1 and 2 for more examples. This suggests that the models learned to predict less recessions when they iterated towards the low-cost areas on their cost planes, and that the initial setting was quite neutral for predicting these two output values. Neither, recession or expansion, appears to have been favored significantly by the chosen methods. When these unideal models were combined with the other two networks, they offered more early warning signals and, therefore, this behavior was less harmful.

The reason why there does not seem to be a significant difference between these two methods is arguably due to their similar performance when measuring with MSE and MAE. Although the large LSTM networks' predictions should have been slightly too sensitive to noise and ungeneralizable phenomena in the data, the large GRU networks could not do much better. Thus, the overfitting effect is clearer when compared to the midsize LSTM networks' predictions in figure 36, which offered the best performance. Also, this indicates that the large GRU networks might have somehow overfit the data even though they did not show rising training error curves. Unfortunately, this phenomenon is difficult

to examine further with the obtained results and, thus, is left out of the scope of this study, but should be revised in future work.

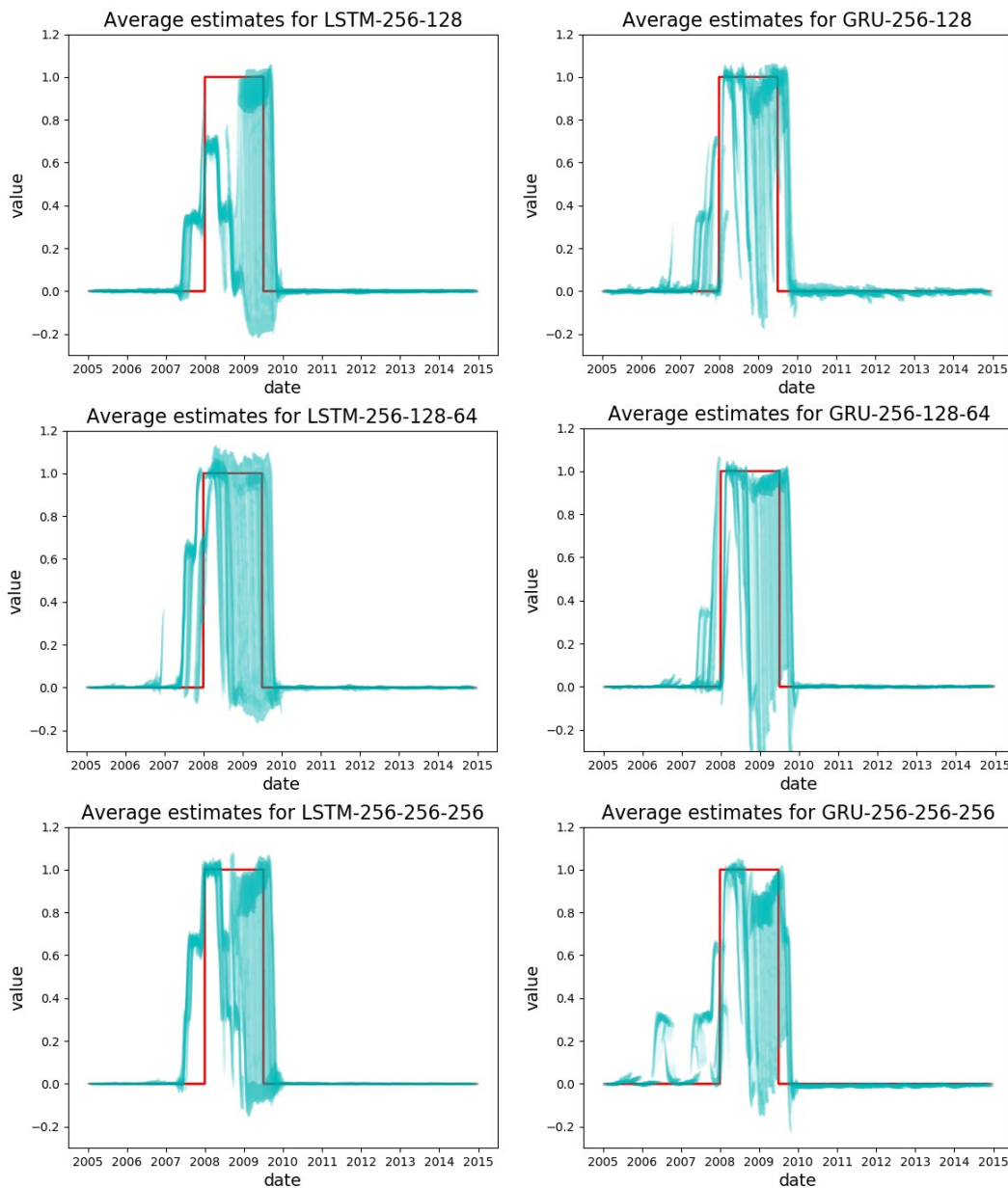


Figure 35 – The average estimates for the three largest LSTM and GRU architectures

As proposed earlier, the networks with high model capacity should have been able to modify their predictions more than the smaller networks. Here it shows as more scattered predictions, which are not desirable, unless if they offer some meaningful information. Furthermore, some of the predictions were unnecessarily low or high (below 0 and over 1). It appears that these larger networks found out that it is useful to shoot predictions almost along the vertical line, either upwards or downwards, but for some reason, timing

and placing these predictions on the right spot on the vertical axis emerged difficult, especially when predicting a trough.

If the early stopping indications are considered useful, the networks appear to have been relatively successful when predicting the beginning of the recession. However, predicting when it ends was challenging. In fact, it appears that the smaller networks fared rather well against the other proposed networks because their predictions were more stable during the recession. It is possible that the larger networks began to overfit to the training data more than the less capable models and, hence, emphasized more, for example, the finding that the recessions did not typically last more than ten months. Hence, instead of focusing on the general relationships in the data, the large networks possibly started to apply these training set specific findings more on the testing set and, thus, could not generalize well. When compared to the other sized architectures' predictions, the small and midsize networks appear to have focused better on more general relationships in the data because their predictions are less dispersed. If these findings are correct, it further suggests that the large GRU networks might have overfitted the data.

5.6 The performance of the midsize architectures

The architectures 16-8, 32-16 and 64-32 are considered in this study midsize. As suggested earlier, the midsize LSTM networks offered on average the best performance when measured with the test set MSE and MAE. In contrast, the equivalent GRU architectures had typically very small deviation between their training iterations' performance, but slightly worse performance. Moreover, two out of three best MSE and MAE scores belonged to these three LSTM ensembles, which suggests that these midsize LSTM networks suited the task at hand rather well with the chosen setup.

The midsize LSTM networks could outperform their GRU equivalents when MSE and MAE are considered, and it also shows in figure 36 as more stable predictions. For example, after 2010, the GRU predictions vary considerably more than the LSTM predictions, which was also typical among the larger networks, shown in figure 35. The reason for this phenomenon remains unclear, but it might be related more to a GRU itself than on the network size because it did not appear when LSTM cells were used. However, when examining the smaller networks' estimates in figure 31, this behavior did not seem to appear, suggesting that it needs a large enough network with GRUs to occur, though it appears stronger with these midsize network architectures than with the three largest architectures and, hence, it might not relate strictly to modeling capacity and overfitting.

Furthermore, as the midsize networks began to favor a vertical line as the shape of the predictions when the probability of a peak or trough to occur increased enough, the GRU networks had more troubles placing these predictions on the y-axis. For example, these midsize GRU networks tended to predict considerably lower values than necessary, whereas this effect was notably smaller with the LSTM counterparts. The midsize GRU networks appear to have been too sensitive to changes in the input data, which is typical for models that suffer from overfitting. In turn, the best models in this study could stand out by limiting these defects, suggesting that they possessed a suitable modeling capacity.

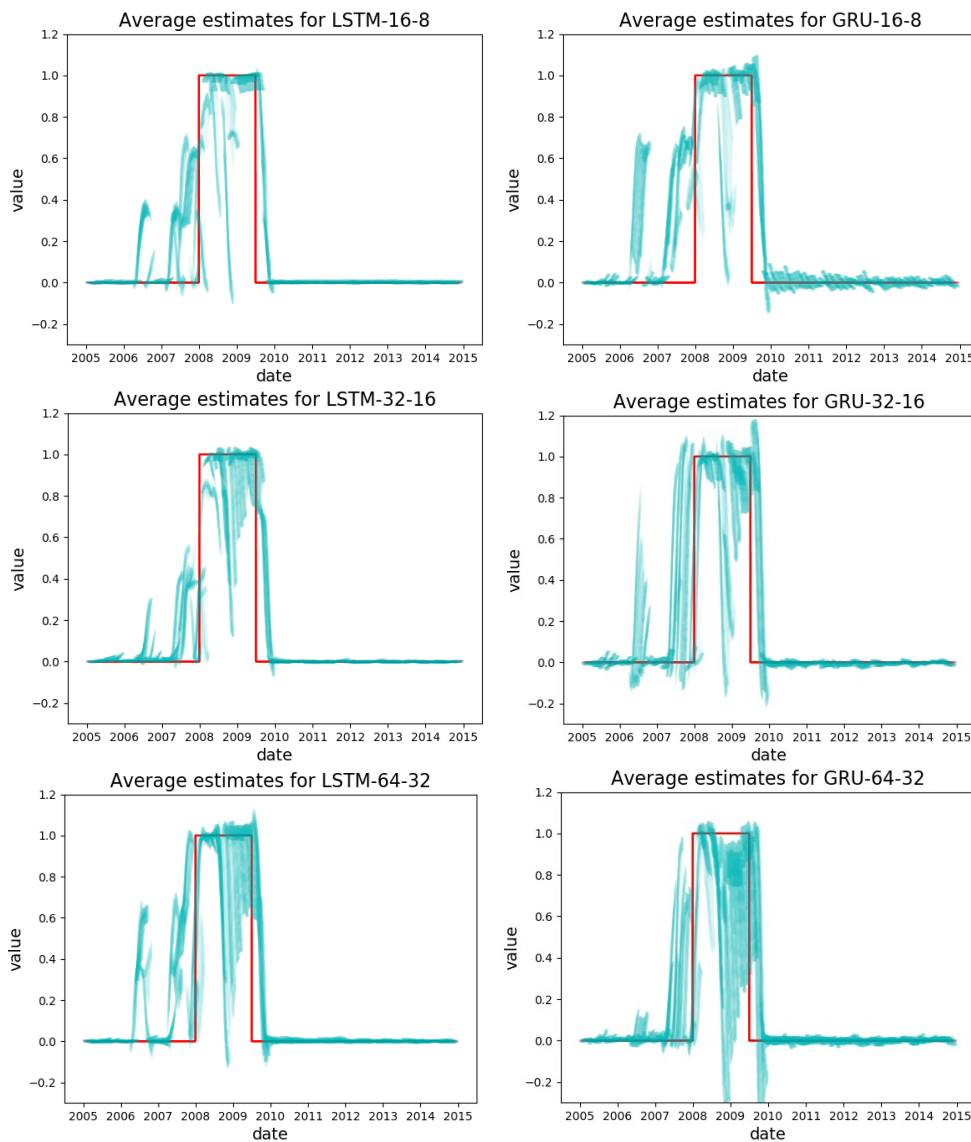


Figure 36 – The average estimates for three midsize LSTM and GRU architectures Similarly to the other networks, the midsize networks also tended to predict a peak between 2006 and 2007 and a trough too early. Again, the notable difference here was that,

the more the network possessed modeling capacity, the worse it tended to predict the trough. This finding is true for both network types, though it was slightly more severe for the GRU networks. This suggests that much higher modeling capacity would not probably suit the problem and, here, LSTM-16-8 and LSTM-32-16 appear to have offered the best performance based on both: the performance scores and the plotted predictions.

5.7 The best architecture

When measured either with test set MSE or MAE, LSTM-32-16 ensemble achieved the lowest value. Although the performance for the ensemble was desirable, the related variance values were slightly up, suggesting that some variation between these networks' predictions existed. Also, because of this variation, their predictions may have corrected each other just right when averaged and, thus, yielded an excellent ensemble performance. Although this effect was somewhat common with the proposed ensembles, it was not particularly strong with LSTM-32-16. Two out of the three lowest test set MSE scores between all the separate iterations belonged to LSTM-32-16 networks, indicating that the first place is somewhat deserved. However, the performance was slightly worse when MAE is considered, suggesting that some benefits have been gained from combining this ensemble when the effect of the model specific noise and other behavior was reduced, but it was somewhat common for all the ensembles.

Table 10 – LSTM-32-16 performance

Architecture	Epochs	Test set MSE	Test set MAE
LSTM-32-12 ensemble	-	0.0368	0.0709
LSTM-32-12 iteration 1	41	0.0743	0.0983
LSTM-32-12 iteration 2	33	0.0359	0.0686
LSTM-32-12 iteration 3	33	0.0417	0.0556

The graphs in figure 37 are in line with the related MSE and MAE scores in table 10. The second network's predictions offered notable early warning signs that generated some MAE, but it was able to keep the predictions otherwise relatively consistent with the real values. In addition, the second network favored more symmetrical curve shaped predictions than the other two networks. These types of predictions were common among the small networks and, as proposed earlier, the larger the network, the more the predictions'

shape appears to have varied, possibly because of their ability to take more patterns into account. Thus, this second LSTM-32-16 network might have concentrated on fewer phenomena in the data than the two other networks and, therefore, could generalize better.

The third network's predictions offered considerably less early warning signs than the other two networks' predictions and, hence, it generated less MAE. However, it was not able to be as consistent during the recession as the second network and, thus, was penalized slightly more. Although the first network received weaker MSE and MAE scores, it offered the best early warning signs from these three networks. Interestingly, the first and third networks generated unnecessarily high outputs. It seems that these networks learned to form a suitable output shape for predicting a peak and trough, as opposed to the second network, but they were, along with several other midsize and larger networks, rather awkward at placing it correctly on the y-axis.

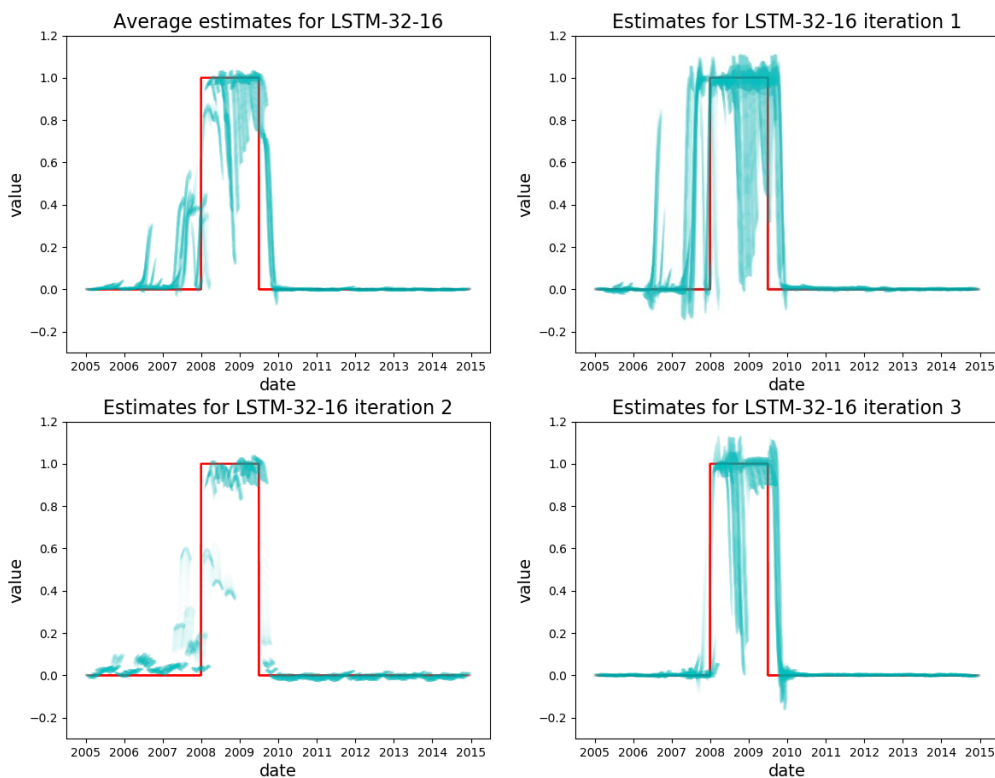


Figure 37 – LSTM-32-16 predictions

When these three relatively different predictions were averaged, the end result offered quite nice behavior. Apart from predicting a trough too early, which was also somewhat common among all the proposed networks, the average predictions were rather pleasing and yielded the best performance of all the chosen architecture options. This suggests that the proposed methods are suitable for predicting business cycles if this type of architecture is used in conjunction with some ensemble solution.

6 CONCLUSIONS

This study focuses on how several different recurrent neural networks (RNNs), comprising either long short-term memory cells (LSTMs) or gated recurrent units (GRUs), can handle predicting U.S. business cycles between the years 2005 and 2015. The set of proposed architectures covers a wide range of possible model capacities from networks with one hidden layer and less than 600 optimizable parameters to ones with three hidden layers and over one million adjustable parameters. Each different architecture is trained three times and these three separate networks' predictions are averaged together to create ensembles that could represent better the potential performance of each architecture. However, even with the applied precautions, the results show notable signs of volatility.

According to the bias-variance trade-off, the out-of-sample prediction performance can indicate which model is the most suitable for finding the general pattern in the selected dataset and, thus, reveal what type of models one should apply for predicting the future examples. However, while these results help finding a suitable RNN architecture from a vast set of viable alternatives, they were generated by using only one possible group of methods and hyperparameters. Therefore, it is highly probable that these methods' performance does not appear identical in some other setup.

With the proposed setting, which comprised typical machine- and deep learning methods, the results recommend using LSTM networks with two hidden layers and the number of optimizable parameters should be between 1 000 and 100 000. Interestingly, where some of the LSTM networks showed clear symptoms of under- and overfitting, their GRU counterparts' performance did not vary significantly between different GRU architectures and their under- and overfitting effect remains slightly obscure. For example, the large GRU networks did not offer evident signs of overfitting in their training graphs, but their predictions included behavior that is similar to the large LSTM networks' predictions, which suffered from overfitting.

The LSTM and GRU networks fared differently depending on the measure and neither could outperform the other in all the examined areas. When based on the preferred performance metrics: mean squared error and mean absolute error, the midsize LSTM networks could offer the best performance. However, surprisingly, even the smallest networks in this study could achieve good performance, suggesting that one does not need a vast RNN architecture for modeling business cycles if LSTM cells or GRUs are used.

Because a wide range of architectures are able to learn the general pattern satisfactorily, these novel RNN methods can be recognized as noteworthy solutions for business cycle forecasting. However, more research should be conducted to verify these results and, also, to find even more enhanced algorithms. This study examines only a small set of possible deep learning methods and hyperparameter values and it is very likely that better combinations exist. In the future, more focus should be put on different types of preprocessing methods that can reduce the amount of noise and other unnecessary short-term movements in the data that are redundant, or even harmful, for predicting long-term macroeconomic developments. Also, different options in the training algorithms should be studied, such as the popular adaptive learning rate methods: Adam, RMSprop and others. Furthermore, using mean absolute error instead of mean squared error in the cost function could provide more stable predictions and less early warning signs. If early warnings are wanted, the business cycle time-series in the training set could be manipulated so that it gradually increases and decreases before every peak and trough to encourage the model to show more clues of a turn in the cycle.

REFERENCES

- Abadi, Martín – Agarwal, Ashish – Barham, Paul – Brevdo, Eugene – Chen, Zhifeng – Citro, Craig – Corrado, Greg S. – Davis, Andy – Dean, Jeffrey – Devin, Matthieu – Ghemawat, Sanjay – Goodfellow, Ian – Harp, Andrew – Irving, Geoffrey – Isard, Michael – Jia, Yangqing – Jozefowicz, Rafal – Kaiser, Lukasz – Kudlur, Manjunath – Levenberg, Josh – Mané, Dandelion – Monga, Rajat – Moore, Sherry – Murray, Derek – Olah, Chris – Schuster, Mike – Shlens, Jonathon – Steiner, Benoit – Sutskever, Ilya – Talwar, Kunal – Tucker, Paul – Vanhoucke, Vincent – Vasudevan, Vijay – Viégas, Fernanda – Vinyals, Oriol – Warden, Pete – Wattenberg, Martin – Wicke, Martin – Yu, Yuan – Zheng, Xiaoqiang (2015) TensorFlow: Large-scale machine learning on heterogeneous systems. <<http://tensorflow.org/>>, retrieved 17.7.2019.
- Barnett, William A. – Serletis, Apostolos – Serletis, Demitre (2015) Nonlinear and Complex Dynamics in Economics. *Macroeconomic Dynamics*, Vol. 19, 1749–1779.
- Basuchoudhary, Atin – Bang, James T. – Sen, Tinni (2017) *Machine-learning Techniques in Economics*. Springer International Publishing.
- Baumohl, Bernard (2012) *The Secrets of Economic Indicators: Hidden Clues to Future Economic Trends and Investment Opportunities (3rd Edition)*. FT Press, New Jersey.
- Bengio, Yoshua (2012) Practical recommendations for gradient-based training of deep architectures. In: Montavon, Grégoire – Orr, Geneviève B. – Müller, Klaus-Robert (Eds.) *Neural Networks: Tricks of the Trade*. Springer-Verlag, Berlin, Heidelberg, 437–478.
- Bengio, Yoshua – Simard, Patrice – Frasconi, Paolo (1994) Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, Vol. 5 (2), 157–166.

- Binner, Jane M. – Elger, Thomas G. – Nilsson, Birger – Tepper, Jonathan A. (2004) Tools for non-linear time series forecasting in economics: an empirical comparison of regime switching vector autoregressive models and recurrent neural networks. *Advances in Econometrics*, Vol. 19, 71–91.
- Brooks, Chris (2014) *Introductory Econometrics for Finance 3rd Edition*. Cambridge University Press, Cambridge.
- Cho, Kyunghyun – van Merriënboer, Bart – Gulcehre, Caglar – Bahdanau, Dzmitry – Bougares, Fethi – Schwenk, Holger – Bengio, Yoshua (2014) Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. Cornell University Library, arXiv.org, Ithaca.
- Chollet, Fran et al. (2015) Keras. <<https://keras.io/>>, retrieved 17.7.2019.
- Chuku, Chuku – Odour, Jacob – Simpasa, Anthony (2017) Intelligent forecasting of economic growth for African economies: Artificial neural networks versus time series and structural econometric models. "Forecasting Issues in Developing Economies 2017" IMF conference paper.
- Delalleau, Olivier – Bengio, Yoshua (2011) Shallow vs. Deep Sum-Product Networks. In *Proceedings of the 24th International Conference on Neural Information Processing Systems (NIPS'11)*. Curran Associates Inc., Red Hook, NY, 666–674.
- Donahue, Jeff – Hendricks, Lisa Anne – Rohrbach, Marcus – Venugopalan, Subhashini – Guadarrama, Sergio – Saenko, Kate – Darrell, Trevor (2017) Long-term Recurrent Convolutional Networks for Visual Recognition and Description. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 39 (4), 677–691.
- Fischer, Thomas – Krauss, Christopher (2018) Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, Vol. 270 (2), 654–669.

- Friedman, Milton (1964) Monetary Studies of the National Bureau. *The National Bureau Enters Its 45th Year, 44th Annual Report*, 7–25.
- Geman, Stuart – Bienenstock, Elie – Doursat, René (1992) Neural Networks and the Bias/Variance Dilemma. *Neural Computation*, Vol. 4 (1), 1–58.
- Géron, Aurélien (2017) *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, Inc., Sebastopol.
- Gers, Felix – Eck, Douglas – Schmidhuber, Jürgen (2001) Applying LSTM to Time Series Predictable through Time-Window Approaches. *Artificial Neural Networks-Icann 2001, Proceedings, 2001*, Vol. 2130, 669–676.
- Giordano, Frank R. – Fox, William P. – Horton, Steven B. (2013) *A First Course in Mathematical Modeling 5th Edition*. Cengage Learning, Boston.
- Goodfellow, Ian – Bengio, Yoshua – Courville, Aaron (2016) *Deep Learning*. MIT Press, <<http://www.deeplearningbook.org/>>, retrieved 3.12.2019.
- Graves, Alex (2012) Supervised Sequence Labelling with Recurrent Neural Networks. *Studies in Computational Intelligence 385*. Springer-Verlag, Berlin, Heidelberg.
- Graves, Alex – Mohamed, Abdel-rahman – Hinton, Geoffrey (2013) Speech Recognition with Deep Recurrent Neural Networks. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. Vancouver, BC, 6645–6649.
- Greff, Klaus – Srivastava, Rupesh K. – Koutník, Jan – Steunebrink, Bas R. – Schmidhuber, Jürgen (2017) LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 28 (10), 2222–2232.
- Hamilton, James D. (1994) *Time series analysis*. Princeton University Press, New Jersey.

- Hastie, Trevor – Tibshirani, Robert – Friedman, Jerome (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*. Springer-Verlag, New York.
- Hochreiter, Sepp – Schmidhuber, Jürgen (1997) Long Short-Term Memory. *Neural Computation*, Vol. 9 (8), 1735–1780.
- Hornik, Kurt – Stinchcombe, Maxwell – White, Halbert (1989) Multilayer feedforward networks are universal approximators. *Neural Networks*, Vol. 2 (5), 359–366.
- Hunter, John D. (2007) Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, Vol. 9, 90–95.
- Jahn, Malte (2018) Artificial neural network regression models: Predicting GDP growth. *HWWI Research Paper, No. 185*. Hamburgisches WeltWirtschaftsinstitut (HWWI), Hamburg.
- James, Gareth – Witten, Daniela – Hastie, Trevor – Tibshirani, Robert (2013) *An Introduction to Statistical Learning*. Springer-Verlag, New York.
- Jozefowicz, Rafal – Zaremba, Wojciech – Sutskever, Ilya (2015) An Empirical Exploration of Recurrent Network Architectures. *ICML'15 Proceedings of the 32nd International Conference on International Conference on Machine Learning*, Vol. 37, 2342–2350.
- Keynes, John M. (1936) *The General Theory of Employment, Interest and Money*. <www.hetwebsite.net/het/texts/keynes/gt/gtcont.htm>, retrieved 16.4.2020.
- Kock, Anders B. – Teräsvirta, Timo (2014) Forecasting performances of three automated modelling techniques during the economic crisis 2007–2009. *International Journal of Forecasting*, Vol. 30 (3), 616–631.

- Krauss, Christopher – Do, Xuan A. – Huck, Nicolas (2017) Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500. *European Journal of Operational Research*, Vol. 259 (2), 689–702.
- Kumar, Manish (2009) Nonlinear Prediction of the Standard & Poor's 500 and the Hang Seng Index under a Dynamic Increasing Sample. *Asian Academy of Management Journal of Accounting and Finance*, Vol. 5 (2), 101–118.
- LeCun, Yann – Bengio, Yoshua – Hinton, Geoffrey (2015) Deep learning. *Nature*, Vol. 521 (7553), 436–444.
- Luong, Thang – Sutskever, Ilya – Le, Quoc – Vinyals, Oriol – Wojciech Zaremba (2015) Addressing the Rare Word Problem in Neural Machine Translation. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, Vol. 1, 11–19.
- Lütkepohl, Helmut (2005) *New Introduction to Multiple Time Series Analysis*. Springer-Verlag, Berlin, Heidelberg.
- Längkvist, Martin – Karlsson, Lars – Loutfi, Amy (2014) A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, Vol. 42 (1), 11–24.
- Makridakis, S. – Andersen, A. – Carbone, R. – Fildes, R. – Hibon, M. – Lewandowski, R. – Newton, J. – Parzen, E. – Winkler, R. (1982) The accuracy of extrapolation (time series) methods: Results of a forecasting competition. *Journal of Forecasting*, Vol. 1 (2), 111–153.
- Makridakis, Spyros – Chatfield, Chris – Hibon, Michèle – Lawrence, Michael – Mills, Terence – Ord, Keith – Simmons, LeRoy F. (1993) The M2-competition: A real-time judgmentally based forecasting study. *International Journal of Forecasting*, Vol. 9 (1), 5–22.

- Makridakis, Spyros – Hibon, Michèle (2000) The M3-Competition: results, conclusions and implications. *International Journal of Forecasting*, Vol. 16, 451–476.
- Makridakis, Spyros – Spiliotis, Evangelos – Assimakopoulos, Vassilios (2018) Statistical and Machine Learning forecasting methods: Concerns and ways forward. *PLoS ONE*, Vol. 13 (3), e0194889.
- Marchi, Erik – Ferroni, Giacomo – Eyben, Florian – Gabrielli, Leonardo – Squartini, Stefano – Schuller, Björn (2014) Multi-resolution linear prediction based features for audio onset detection with bidirectional LSTM neural networks. *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2164–2168
- McKinney, Wes (2010) Data Structures for Statistical Computing in Python. *Proceedings of the 9th Python in Science Conference*, 51–56.
- Meese, Richard – Geweke, John (1984) A Comparison of Autoregressive Univariate Forecasting Procedures for Macroeconomic Time Series. *Journal of Business & Economic Statistics*, Vol. 2 (3), 191–200.
- Microsoft Corporation (2018) Microsoft Excel. <<https://office.microsoft.com/excel>>
- Mitchell, Wesley C. (1927) *Business Cycles: The Problem and Its Setting*. NBER Book Series Studies in Business Cycles.
- Moody, John (2012) Forecasting the Economy with Neural Nets: A Survey of Challenges and Solutions. In: Montavon, Grégoire – Orr, Geneviève B. – Müller, Klaus-Robert (Eds.) *Neural Networks: Tricks of the Trade*. Springer-Verlag, Berlin, Heidelberg, 343–367.
- Murphy, Kevin P. (2012) *Machine Learning: A Probabilistic Perspective*. The MIT Press, Cambridge.


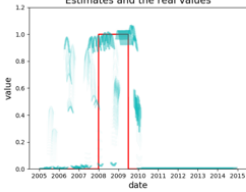
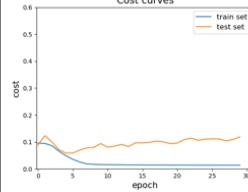
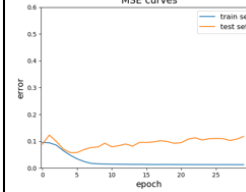
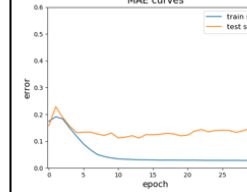
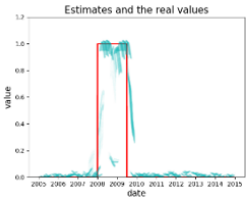
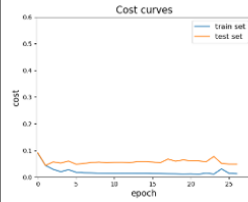
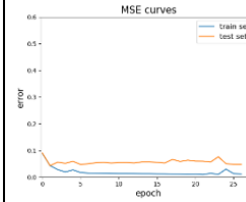
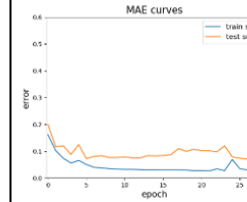
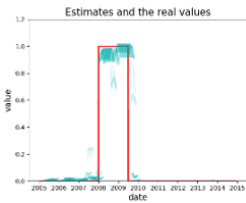
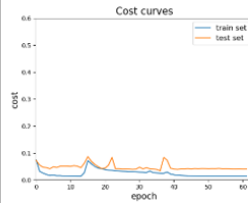
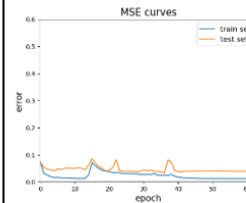
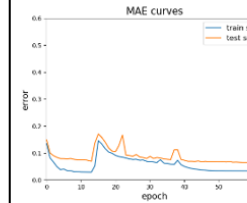
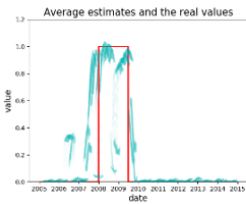
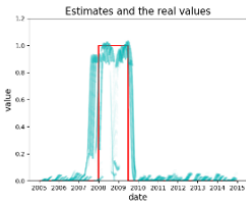
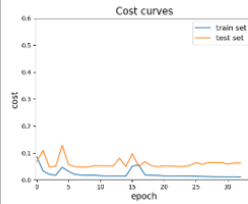
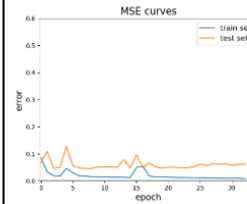
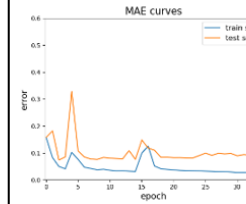
- NBER (2019) The NBER's Business Cycle Dating Committee. <<http://www.nber.org/cycles/recessions.html>>, retrieved 7.11.2019.
- Niaki, Seyed T.A. – Hoseinzade, Saeid (2013) Forecasting S&P 500 index using artificial neural networks and design of experiments. *Journal of Industrial Engineering International*, Vol. 9 (1), 1–9.
- Nielsen, Aileen (2019) *Practical Time Series Analysis: Prediction with Statistics & Machine Learning*. O'Reilly Media, Inc., Sebastopol.
- Olah, Christopher (2015) Understanding LSTM Networks. <<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>>, retrieved 12.12.2019.
- Pascanu, Razvan – Gulcehre, Caglar – Cho, Kyunghyun – Bengio, Yoshua (2014) How to Construct Deep Recurrent Neural Networks. Cornell University Library, arXiv.org, Ithaca.
- Pedregosa, Fabian – Varoquaux, Gaël – Gramfort, Alexandre – Michel, Vincent – Thirion, Bertrand – Grisel, Olivier – Blondel, Mathieu – Prettenhofer, Peter – Weiss, Ron – Dubourg, Vincent – Vanderplas, Jake – Passos, Alexandre – Cournapeau, David – Brucher, Matthieu – Perrot, Matthieu – Duchesnay, Édouard (2011) Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, Vol. 12, 2825–2830.
- Prechelt, Lutz (2012) Early Stopping — But When? In: Montavon, Grégoire – Orr, Geneviève B. – Müller, Klaus-Robert (Eds.) *Neural Networks: Tricks of the Trade*. Springer-Verlag, Berlin, Heidelberg, 53–67.
- Python Software Foundation (2019) Python 3.6.8 documentation. <<https://docs.python.org/3.6/>>, retrieved 17.7.2019.
- Qi, Min (2001) Predicting US recessions with leading indicators via neural network models. *International Journal of Forecasting*, Vol. 17 (3), 383–401.

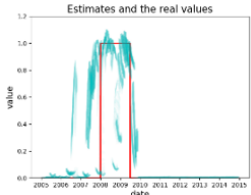
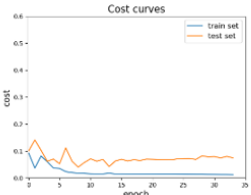
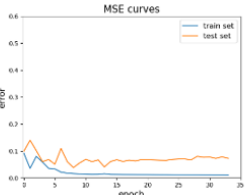
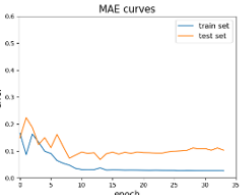
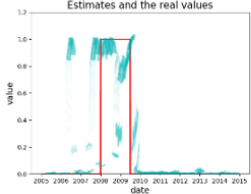
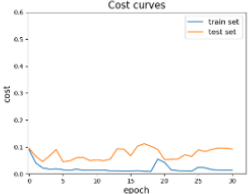
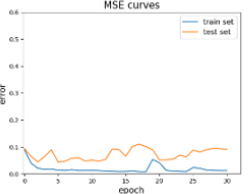
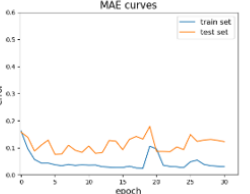
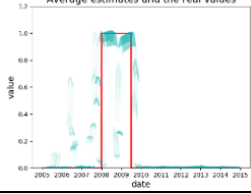
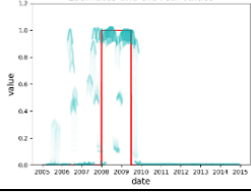
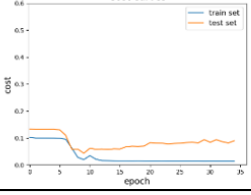
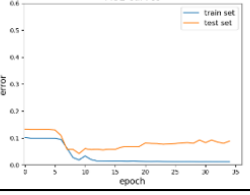
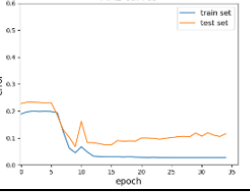
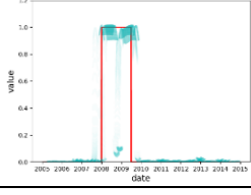
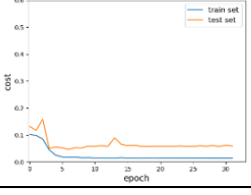
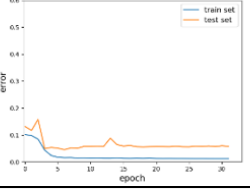
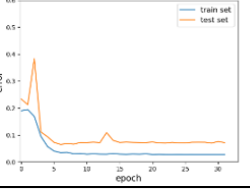
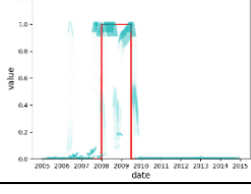
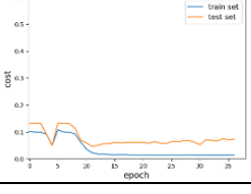
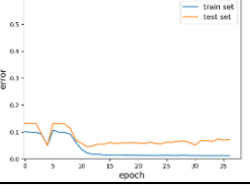
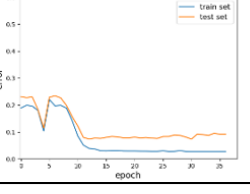
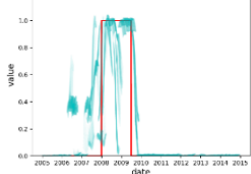
- Qi, Min – Zhang, Peter G. (2008) Trend Time-Series Modeling and Forecasting With Neural Networks. *IEEE Transactions on Neural Networks*, Vol. 19 (5), 808–816.
- Reinhart, Carmen M. – Rogoff, Kenneth S. (2008) Is the 2007 US Sub-prime Financial Crisis So Different? *American Economic Review*, Vol. 98 (2), 339–44.
- Rudebusch, Glenn D. – Williams, John C. (2008) Forecasting Recessions: The Puzzle of the Enduring Power of the Yield Curve. *Journal of Business & Economic Statistics*, Vol. 27 (4), 492–503.
- Sheta, Alaa F. – Ahmed, Sara E. M. – Faris, Hossam (2015) A Comparison between Regression, Artificial Neural Networks and Support Vector Machines for Predicting Stock Market Index. *International Journal of Advanced Research in Artificial Intelligence*, Vol. 4 (7), 55–63.
- Stock, James H. – Watson, Mark W. (1998) A comparison of linear and nonlinear univariate models for forecasting macroeconomic time series. *NBER Working Paper Series, Working Paper 6607*. National Bureau of Economic Research, Inc., Cambridge.
- Swanson, Norman R. – White, Halbert (1995) A Model Selection Approach to Assessing the Information in the Term Structure Using Linear Models and Artificial Neural Networks. *Journal of Business and Economic Statistics*, Vol. 13 (3), 265–275.
- Swanson, Norman R. – White, Halbert (1997) A model selection approach to real-time macroeconomic forecasting using linear models and artificial neural networks. *The Review of Economics and Statistics*, Vol. 79 (4), 540–550.
- Tkacz, Greg (2001) Neural network forecasting of Canadian GDP growth. *International Journal of Forecasting*, Vol. 17 (1), 57–69.
- Trask, Andrew (2019) *Grokking Deep Learning*. Manning Publications Co., USA.

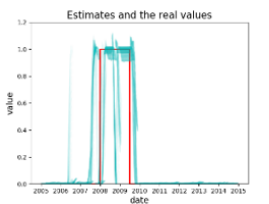
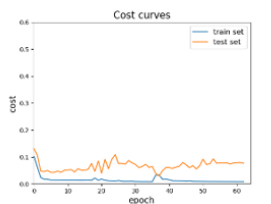
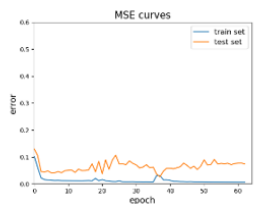
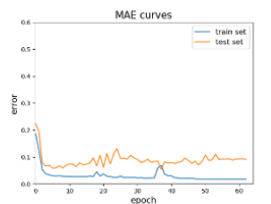
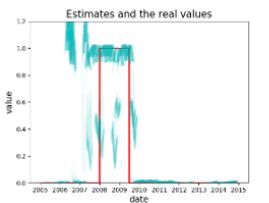
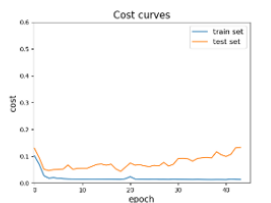
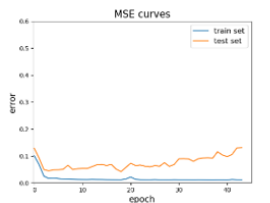
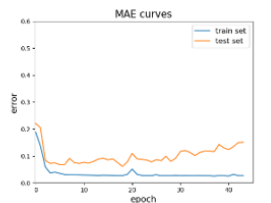
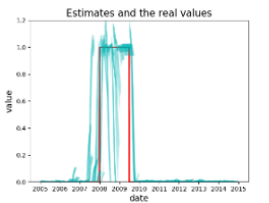
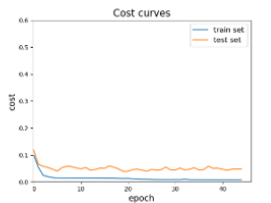
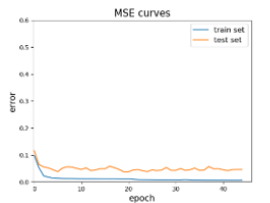
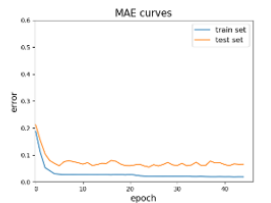
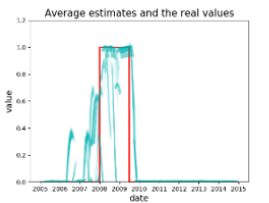
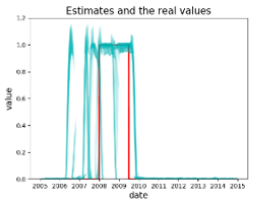
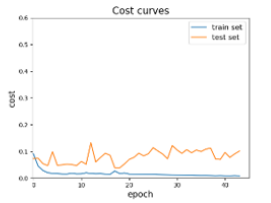
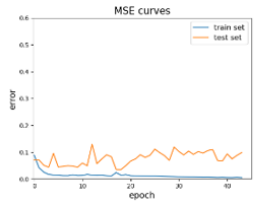
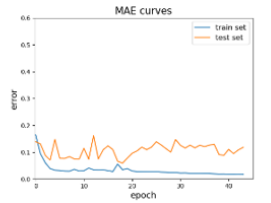
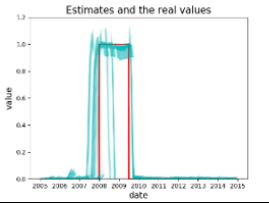
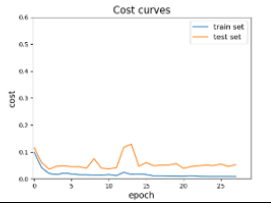
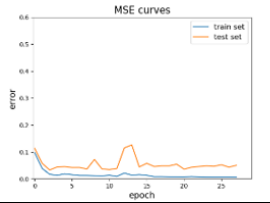
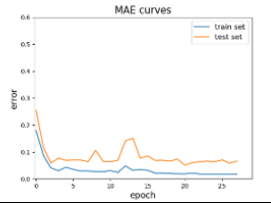
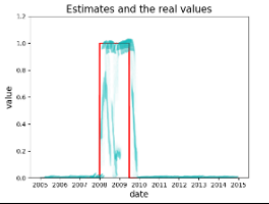
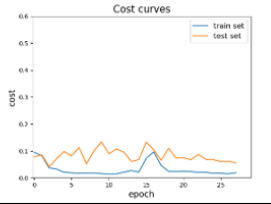
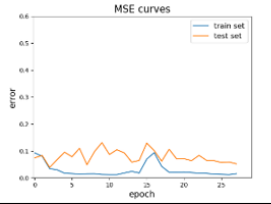
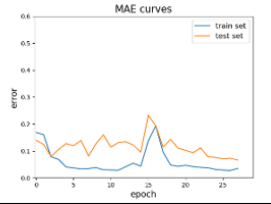
- Tsay, Ruey S. – Chen, Rong (2019) *Nonlinear Time Series Analysis*. John Wiley & Sons, Inc, New Jersey.
- van der Walt, Stéfan – Colbert, Chris S. – Varoquaux, Gaël (2011) The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering*, Vol. 13, 22–30.
- Varian, Hal (2014) Big Data: New Tricks for Econometrics. *Journal of Economic Perspectives*, Vol. 28 (2), 3–28.
- Weigand, Andreas S. – Gershenfeld, Neil A. (1994) Time series prediction: Forecasting the future and understanding the past. *International Journal of Forecasting*, Vol. 10 (4), 161–164.
- Wolpert, David H. (1996) The lack of a priori distinctions between learning algorithms. *Neural Computation*, Vol. 8 (7), 1341–1390.
- Wolpert, David H. – Macready, William G. (1997) No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, Vol. 1 (1), 67–82.
- Zarnowitz, Victor – Braun, Philip A. (1992) Twenty-two Years of the NBER-ASA Quarterly Economic Outlook Surveys: Aspects and Comparisons of Forecasting Performance. *Studies in Business Cycles*, Vol. 28.
- Zhang, Peter G. (2012) Neural networks for time-series forecasting. In: Rozenberg, G. – Bäck, T., – Kok, J. N. (Eds.) *Handbook of natural computing*. Springer-Verlag, Berlin, Heidelberg, 461–477.
- Zhang, Peter G. – Qi, Min (2002) Predicting consumer retail sales using neural networks. In: Smith, Kate A. – Gupta, Jatinder N. D. (Eds.) *Neural networks in business: techniques and applications*. Idea Group, Hershey, 26–40.
- Zhang, Peter G. – Qi, Min (2005) Neural network forecasting for seasonal and trend time series. *European Journal of Operational Research*, Vol. 160 (2), 501–514.

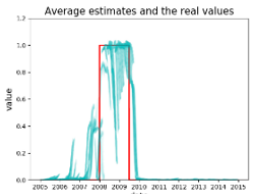
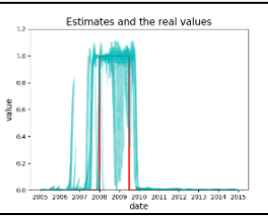
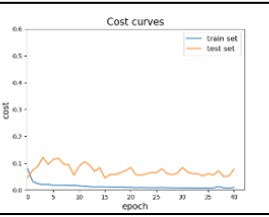
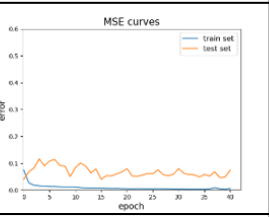
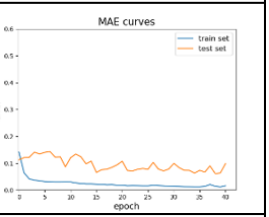
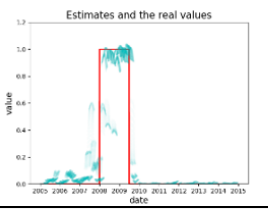
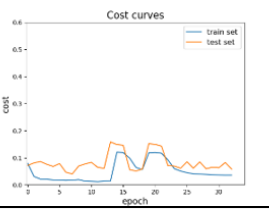
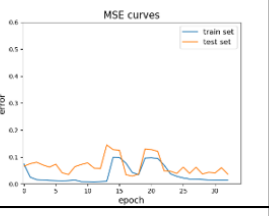
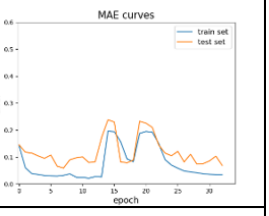
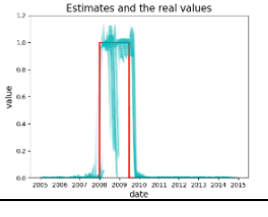
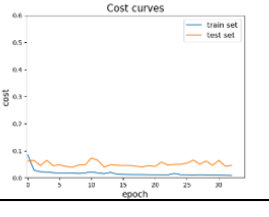
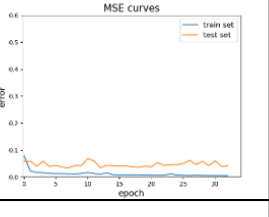
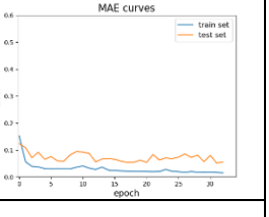
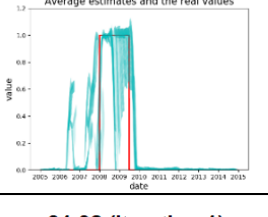
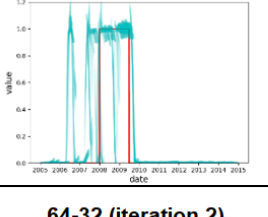
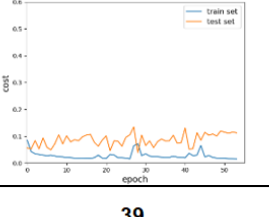
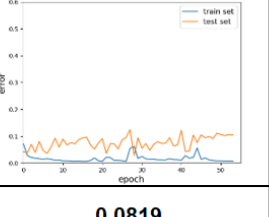
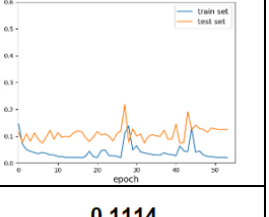
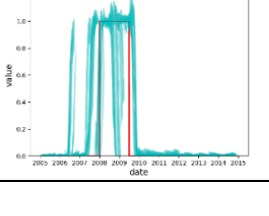
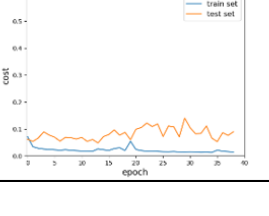
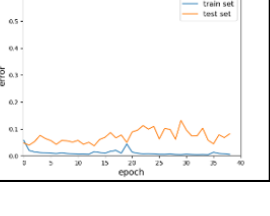
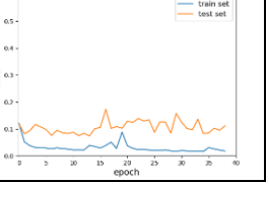
APPENDICES

Appendix 1. LSTM results

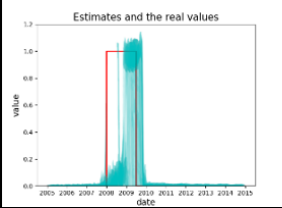
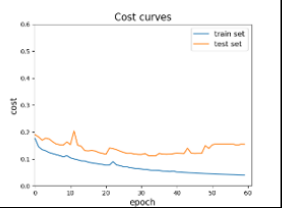
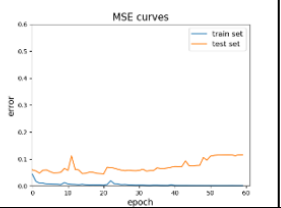
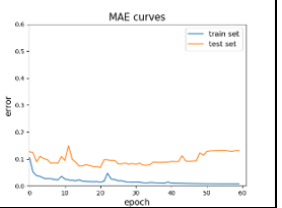
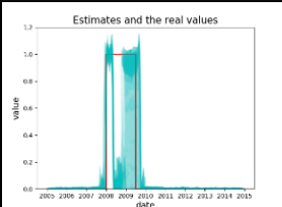
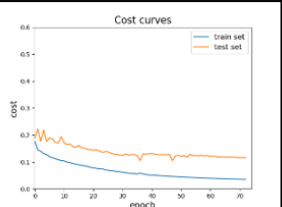
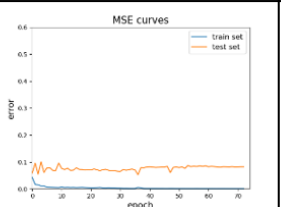
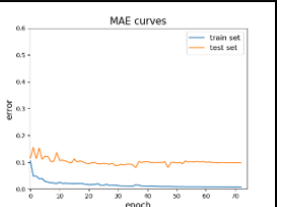
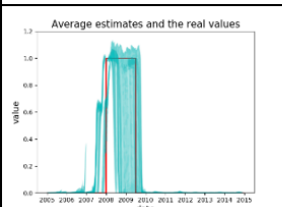
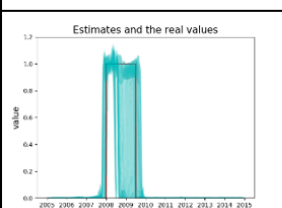
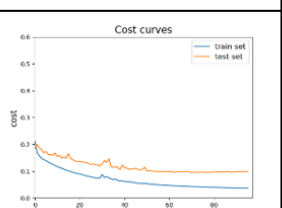
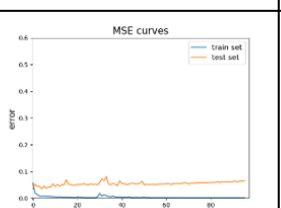
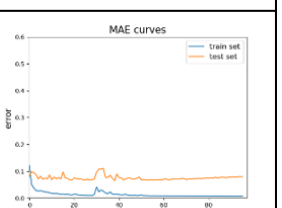
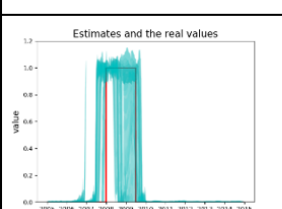
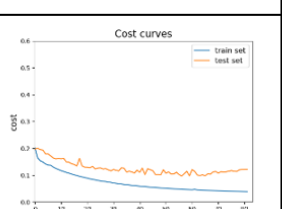
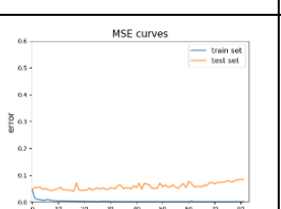
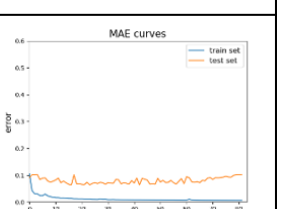
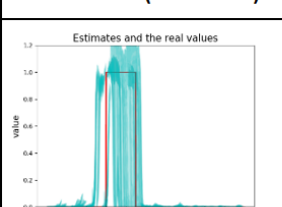
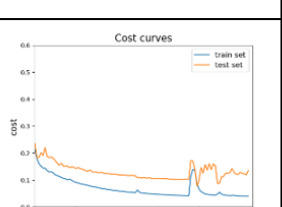
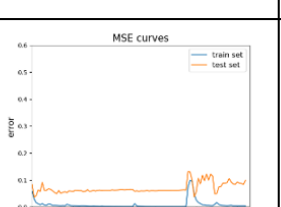
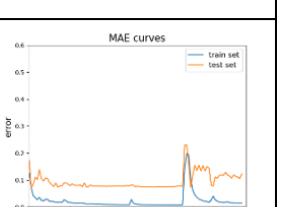
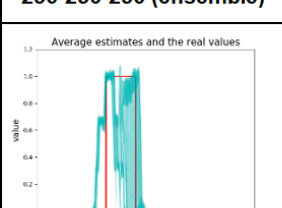
Architecture	Epochs	Test set MSE	Test set MAE
2 (ensemble)	39.6664	0.0493	0.0902
			
2 (iteration 1)	30	0.1176	0.1489
			
2 (iteration 2)	27	0.0477	0.0709
			
2 (iteration 3)	62	0.0387	0.0642
			
4 (ensemble)	32.6664	0.0658	0.1037
			
4 (iteration 1)	33	0.0608	0.0916
			

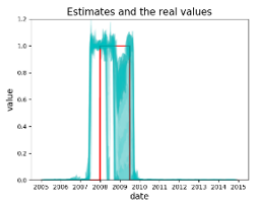
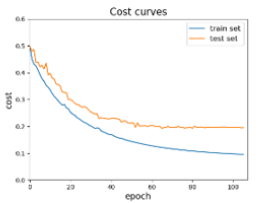
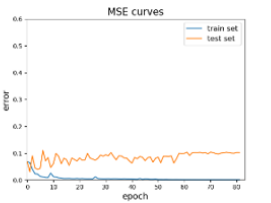
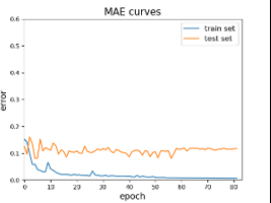
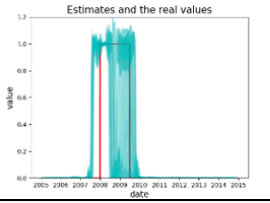
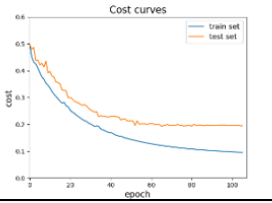
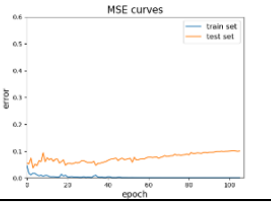
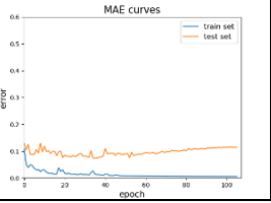
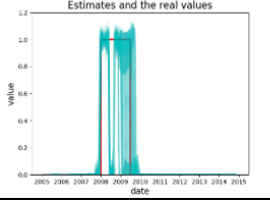
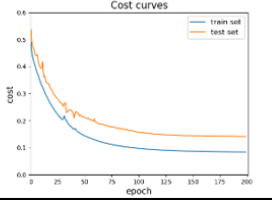
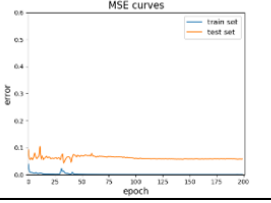
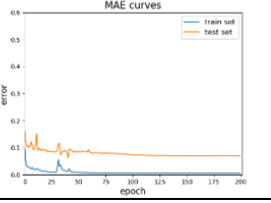
<p>4 (iteration 2)</p>	<p>34</p>	<p>0.0725</p>	<p>0.1029</p>
			
<p>4 (iteration 3)</p>	<p>31</p>	<p>0.0911</p>	<p>0.1224</p>
			
<p>4-2 (ensemble)</p>	<p>34.6664</p>	<p>0.0603</p>	<p>0.0923</p>
			
<p>4-2 (iteration 1)</p>	<p>35</p>	<p>0.0884</p>	<p>0.1159</p>
			
<p>4-2 (iteration 2)</p>	<p>32</p>	<p>0.0579</p>	<p>0.0713</p>
			
<p>4-2 (iteration 3)</p>	<p>37</p>	<p>0.0713</p>	<p>0.0916</p>
			
<p>8-4 (ensemble)</p>	<p>50.6664</p>	<p>0.0591</p>	<p>0.0997</p>
			

<p>8-4 (iteration 1)</p>	<p>63</p>	<p>0.0742</p>	<p>0.0908</p>
			
<p>8-4 (iteration 2)</p>	<p>44</p>	<p>0.1309</p>	<p>0.1507</p>
			
<p>8-4 (iteration 3)</p>	<p>45</p>	<p>0.0459</p>	<p>0.0655</p>
			
<p>16-8 (ensemble)</p>	<p>33.3334</p>	<p>0.0474</p>	<p>0.0817</p>
			
<p>16-8 (iteration 1)</p>	<p>44</p>	<p>0.0984</p>	<p>0.1178</p>
			
<p>16-8 (iteration 2)</p>	<p>28</p>	<p>0.0503</p>	<p>0.0667</p>
			
<p>16-8 (iteration 3)</p>	<p>28</p>	<p>0.0521</p>	<p>0.0660</p>
			

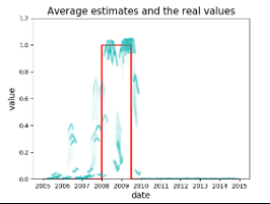
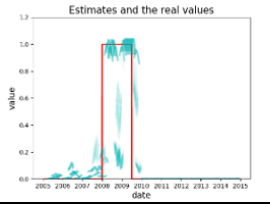
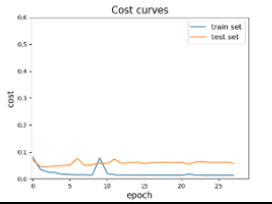
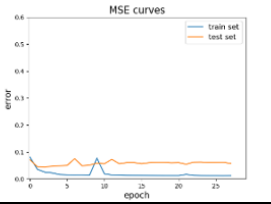
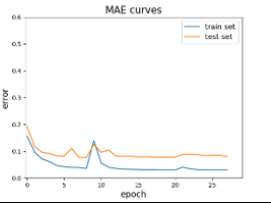
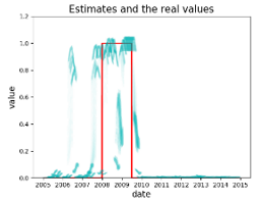
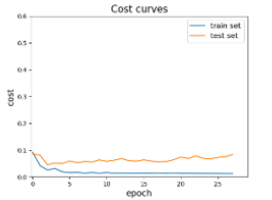
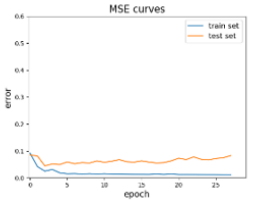
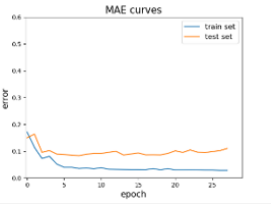
<p>32-16 (ensemble)</p> 	<p>35.6667</p>	<p>0.0368</p>	<p>0.0709</p>
<p>32-16 (iteration 1)</p> 	<p>41</p> 	<p>0.0743</p> 	<p>0.0983</p> 
<p>32-16 (iteration 2)</p> 	<p>33</p> 	<p>0.0359</p> 	<p>0.0686</p> 
<p>32-16 (iteration 3)</p> 	<p>33</p> 	<p>0.0417</p> 	<p>0.0556</p> 
<p>64-32 (ensemble)</p> 	<p>41.664</p>	<p>0.0617</p>	<p>0.1012</p>
<p>64-32 (iteration 1)</p> 	<p>54</p> 	<p>0.1042</p> 	<p>0.1237</p> 
<p>64-32 (iteration 2)</p> 	<p>39</p> 	<p>0.0819</p> 	<p>0.1114</p> 

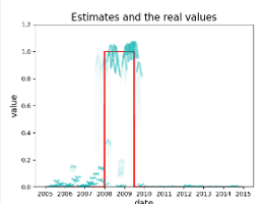
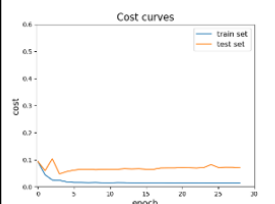
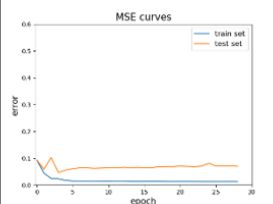
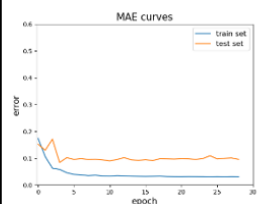
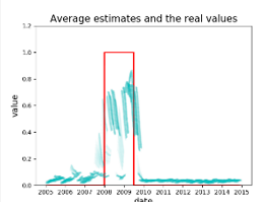



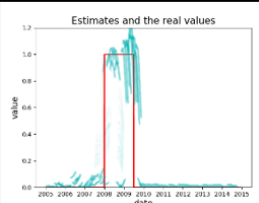
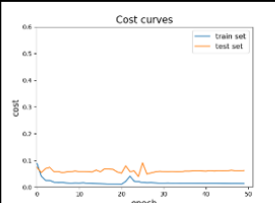
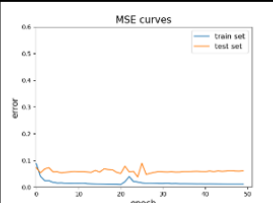
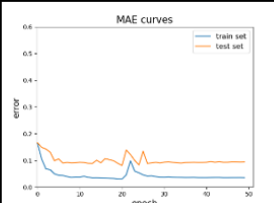
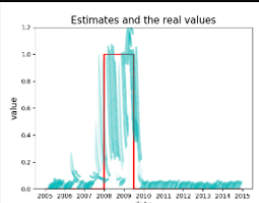
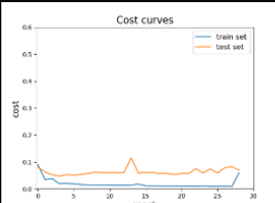
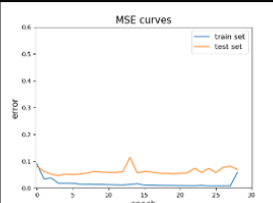
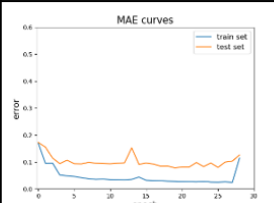
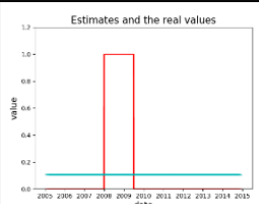
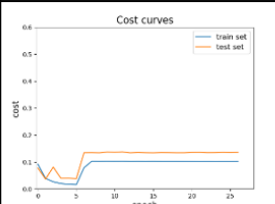
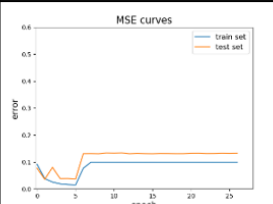
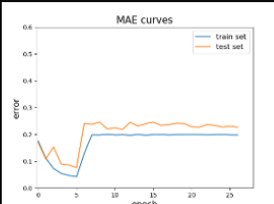
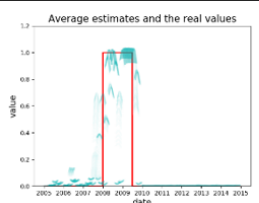



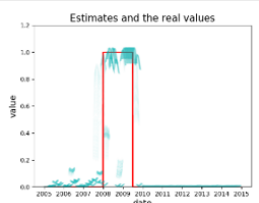
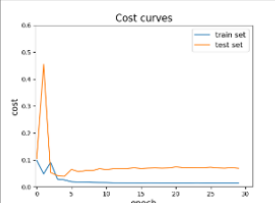
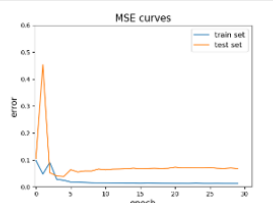
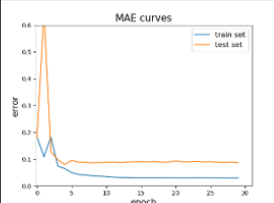
<p>64-32 (iteration 3)</p>	<p>32</p>	<p>0.0546</p>	<p>0.0775</p>
<p>128-64 (ensemble)</p>	<p>52</p>	<p>0.0500</p>	<p>0.0809</p>
<p>128-64 (iteration 1)</p>	<p>58</p>	<p>0.0636</p>	<p>0.0820</p>
<p>128-64 (iteration 2)</p>	<p>42</p>	<p>0.0645</p>	<p>0.0829</p>
<p>128-64 (iteration 3)</p>	<p>56</p>	<p>0.0661</p>	<p>0.0944</p>
<p>256-128 (ensemble)</p>	<p>70.6667</p>	<p>0.0745</p>	<p>0.1139</p>
<p>256-128 (iteration 1)</p>	<p>79</p>	<p>0.1055</p>	<p>0.1220</p>

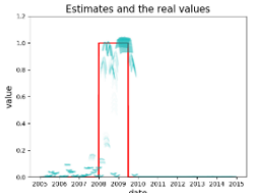
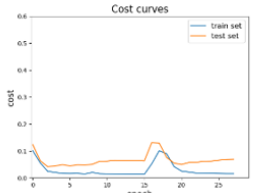
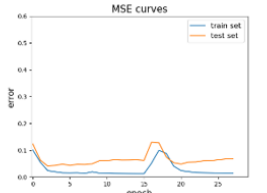
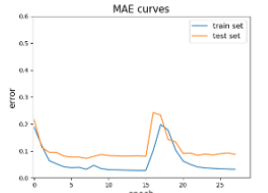
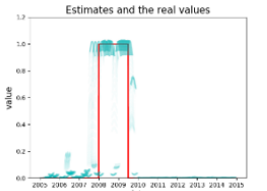
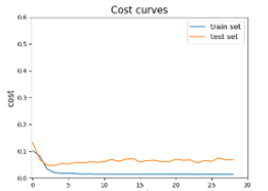
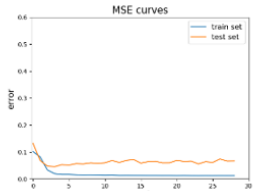
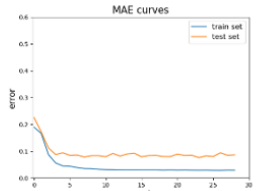
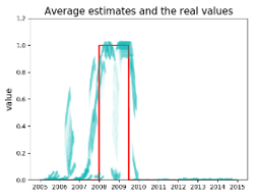
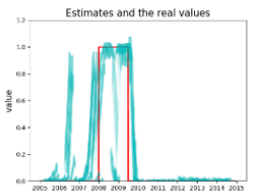
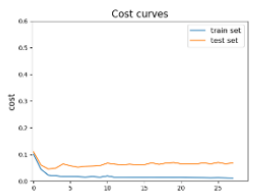
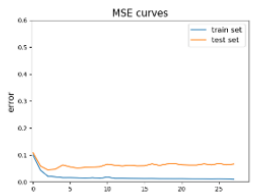
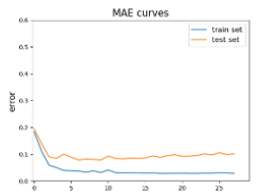

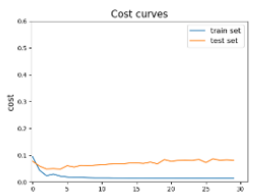
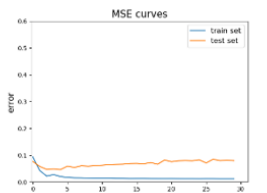
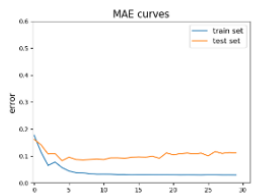
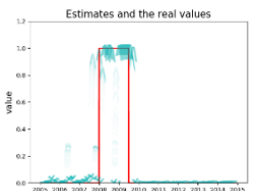
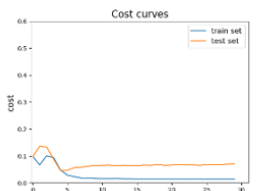
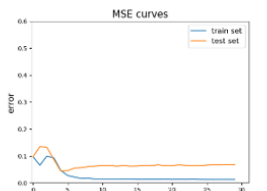
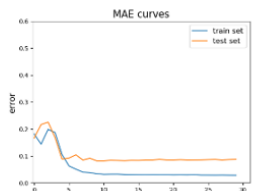
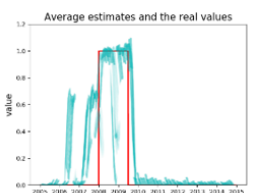
256-128 (iteration 2)	60	0.1157	0.1311
			
256-128 (iteration 3)	73	0.0822	0.0981
			
256-128-64 (ensemble)	94.3334	0.0707	0.0988
			
256-128-64 (iteration 1)	96	0.0636	0.0791
			
256-128-64 (iteration 2)	82	0.0840	0.1018
			
256-128-64 (iteration 3)	105	0.0989	0.1227
			
256-256-256 (ensemble)	129.3334	0.0668	0.0979
			

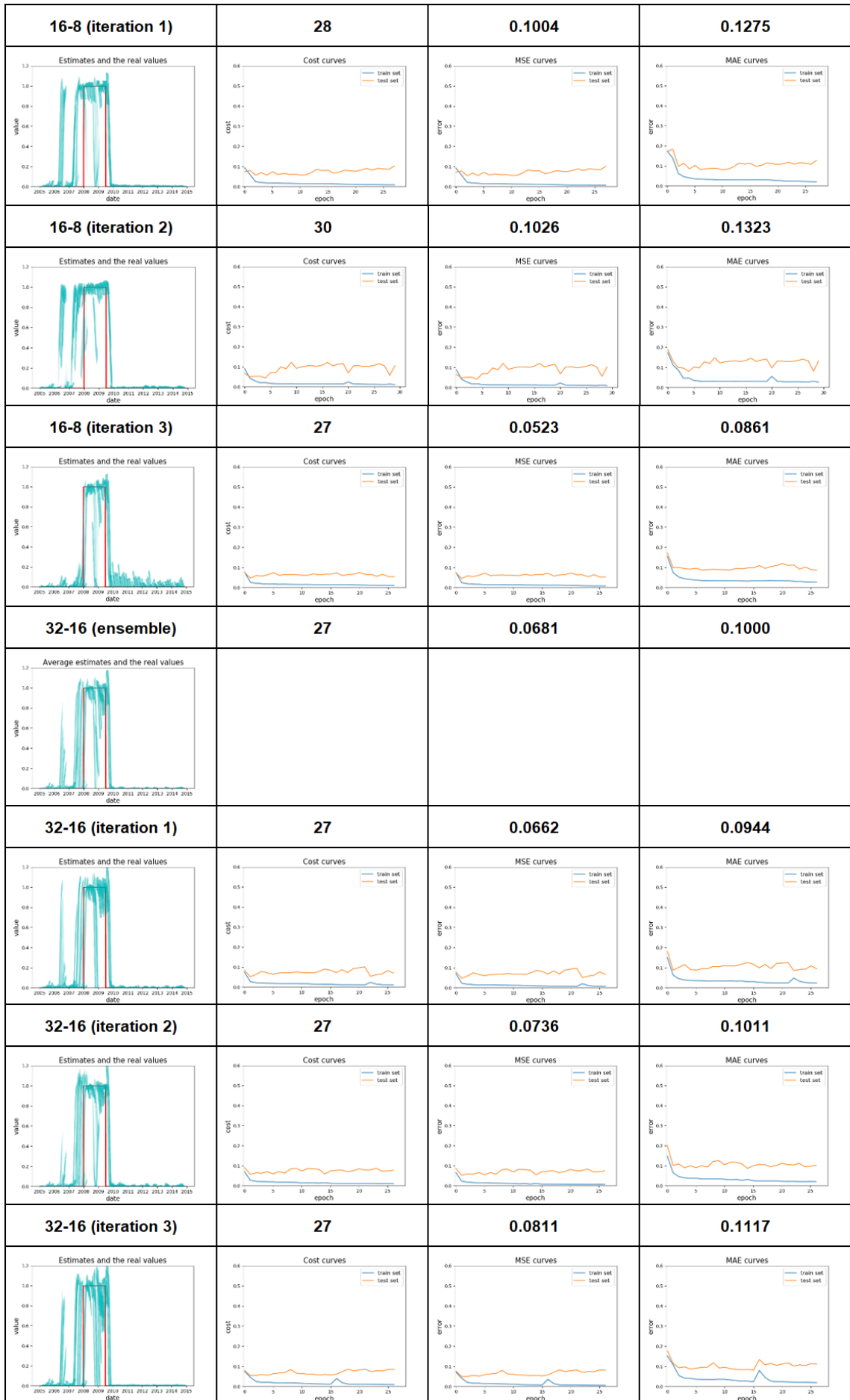
256-256-256 (iteration 1)	82	0.1026	0.1167
			
256-256-256 (iteration 2)	106	0.1005	0.1141
			
256-256-256 (iteration 3)	200	0.0583	0.0700
			

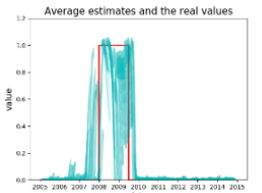
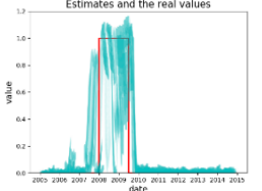
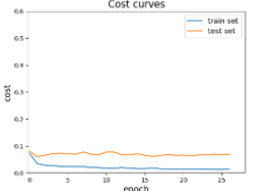
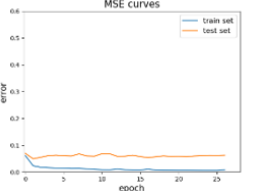
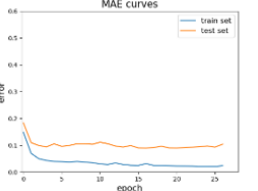
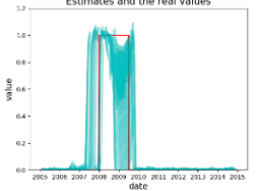
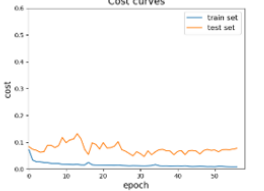
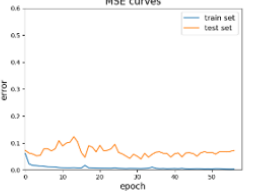
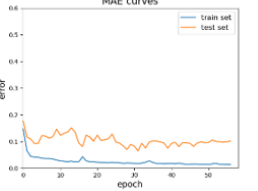
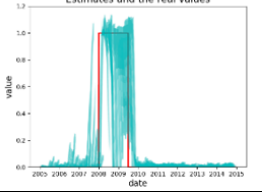
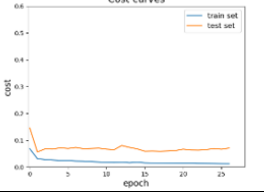
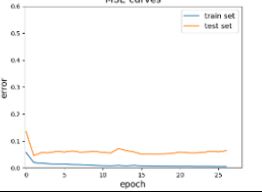
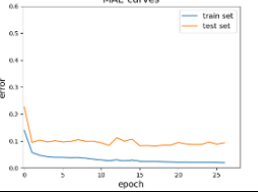
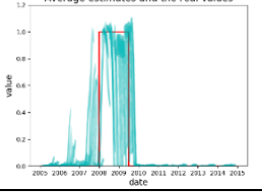
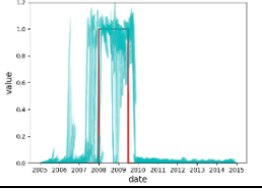
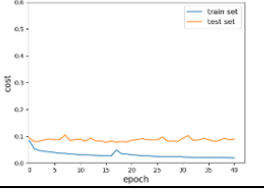
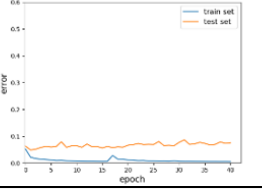
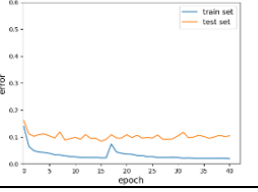
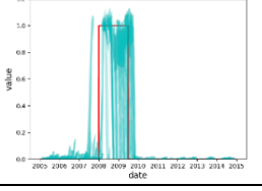
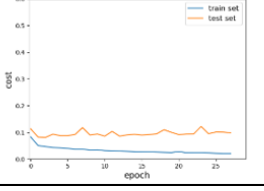
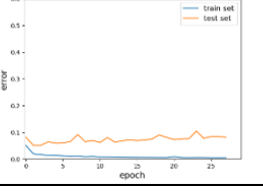
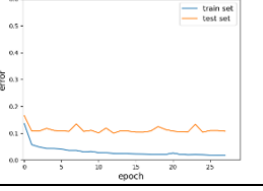
Appendix 2. GRU results

Architecture	Epochs	Test set MSE	Test set MAE
2 (ensemble)	28.3334	0.0596	0.0947
			
2 (iteration 1)	28	0.0572	0.0805
			
2 (iteration 2)	28	0.0827	0.1102
			

<p>2 (iteration 3)</p> 	<p>29</p> 	<p>0.0696</p> 	<p>0.0956</p> 
<p>4 (ensemble)</p> 	<p>35.3334</p> 	<p>0.0633</p> 	<p>0.1344</p> 
<p>4 (iteration 1)</p> 	<p>50</p> 	<p>0.0607</p> 	<p>0.0942</p> 
<p>4 (iteration 2)</p> 	<p>29</p> 	<p>0.0681</p> 	<p>0.1259</p> 
<p>4 (iteration 3)</p> 	<p>27</p> 	<p>0.1322</p> 	<p>0.2270</p> 
<p>4-2 (ensemble)</p> 	<p>29</p> 	<p>0.0597</p> 	<p>0.0869</p> 
<p>4-2 (iteration 1)</p> 	<p>30</p> 	<p>0.0679</p> 	<p>0.0878</p> 

<p>4-2 (iteration 2)</p> 	<p>28</p> 	<p>0.0675</p> 	<p>0.0875</p> 
<p>4-2 (iteration 3)</p> 	<p>29</p> 	<p>0.0664</p> 	<p>0.0863</p> 
<p>8-4 (ensemble)</p> 	<p>29.3334</p>	<p>0.0666</p>	<p>0.0984</p>
<p>8-4 (iteration 1)</p> 	<p>28</p> 	<p>0.0663</p> 	<p>0.1008</p> 
<p>8-4 (iteration 2)</p> 	<p>30</p> 	<p>0.0795</p> 	<p>0.1118</p> 
<p>8-4 (iteration 3)</p> 	<p>30</p> 	<p>0.0685</p> 	<p>0.0876</p> 
<p>16-8 (ensemble)</p> 	<p>28.3334</p>	<p>0.0650</p>	<p>0.1133</p>



<p>64-32 (ensemble)</p> 	<p>37</p>	<p>0.0579</p>	<p>0.0963</p>
<p>64-32 (iteration 1)</p> 	<p>27</p> 	<p>0.0625</p> 	<p>0.1038</p> 
<p>64-32 (iteration 2)</p> 	<p>57</p> 	<p>0.0722</p> 	<p>0.1023</p> 
<p>64-32 (iteration 3)</p> 	<p>27</p> 	<p>0.0639</p> 	<p>0.0933</p> 
<p>128-64 (ensemble)</p> 	<p>32</p>	<p>0.0646</p>	<p>0.0985</p>
<p>128-64 (iteration 1)</p> 	<p>41</p> 	<p>0.0754</p> 	<p>0.1041</p> 
<p>128-64 (iteration 2)</p> 	<p>28</p> 	<p>0.0822</p> 	<p>0.1070</p> 

<p>128-64 (iteration 3)</p>	<p>27</p>	<p>0.0735</p>	<p>0.1013</p>
<p>256-128 (ensemble)</p>	<p>113.3334</p>	<p>0.0573</p>	<p>0.0894</p>
<p>256-128 (iteration 1)</p>	<p>41</p>	<p>0.1048</p>	<p>0.1274</p>
<p>256-128 (iteration 2)</p>	<p>131</p>	<p>0.0722</p>	<p>0.0843</p>
<p>256-128 (iteration 3)</p>	<p>168</p>	<p>0.0451</p>	<p>0.0681</p>
<p>256-128-64 (ensemble)</p>	<p>152</p>	<p>0.0637</p>	<p>0.0848</p>
<p>256-128-64 (iteration 1)</p>	<p>138</p>	<p>0.0625</p>	<p>0.0776</p>

256-128-64 (iteration 2)	200	0.0664	0.0795
256-128-64 (iteration 3)	118	0.0847	0.1033
256-256-256 (ensemble)	177.3334	0.0572	0.1034
256-256-256 (iteration 1)	200	0.0762	0.0908
256-256-256 (iteration 2)	200	0.0739	0.0889
256-256-256 (iteration 3)	132	0.0996	0.1395