André Nunes Correia Gouveia

Master of Science

Machine Learning Applications on Algorithmic Trading in the Foreign Exchange Market

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Mathematical Finance

Advisers: Miguel dos Santos Fonseca,
Assistant Professor,
NOVA University of Lisbon
Pedro Alexandre da Rosa Corte Real,
Assistant Professor,
NOVA University of Lisbon

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

November, 2020

Machine Learning Applications on Algorithmic Trading in the Foreign Exchange Market

This document was created using the (pdf)LaTeX processor, based in the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Acknowledgements

I would like to firstly express my gratitude to my advisors, Prof. Dr. Miguel Fonseca and Prof. Dr. Pedro Corte Real, for their continued support and guidance during the elaboration of this thesis.

I also want to thank my mom and dad, for their effort in raising me and doing their best to bring me up until this point.

To my brother, who is just now starting a degree in Computer Science in this very same institution, for all his support and friendship. I hope one day he will help me further explore the ideas presented in this document.

Finally, to the teacher's assistant in my object-oriented programming classes, Vanessa, who wears many hats. She helped me fully review this dissertation, served as a personal machine-learning consultant, and was an amazing partner through this whole process. A special thanks for her companionship.

Abstract

Nowadays, the largest share of trades done in market exchanges are made by computers. This has been proving to be the main way to invest in the various exchanges. Since the turn of the 20^{th} century, the total volume of trades performed automatically by machines in the United States stock market as gone up from 15% to around 80%. Similarly, in the foreign exchange market, the largest market of the world with over 6 trillion US dollars in daily trade volume during 2019, it is estimated that the large majority of trades are also made by computers.

With the possibility of using machines to trade for us, it makes sense to consider a mathematical theory that deals with modeling prices and financial products, and to program a software to take advantage of this information. Since the last century, another type of models have also been developed that have the capability of adapting themselves, or *learn*, with the information that they are provided.

The objective of this thesis is to implement a strategy that benefits from the information generated by a machine learning model. This required an in-depth research on the underlying theory for this type of models, which is carefully defined here. Besides this, we developed a system that trades automatically for us, including a detailed backtesting engine that permitted to test this strategy, among others, in a simulated environment before using it in the market. This automatic trading system was meticulously designed to ensure extensibility and robustness purposing to explore as many strategies and models as needed, including machine learning approaches, based on a large set of user configurations. Subsequently, the foreign exchange market was used to live-run our strategies, which is open 24h a day during weekdays and is highly liquid. As a benchmark, other more common strategies were also tested and the predictive capability of the machine learning model was compared with an established mathematical model, the autoregressive integrated moving average model.

Keywords: Machine learning, Neural Networks, Automatic trading systems, Backtesting, Algorithmic trading, Financial modeling, Time-series modeling, Quantitative analysis, Foreign exchange market, ARIMA.

Resumo

Hoje em dia, a maior parte dos negócios em bolsa são feitos por computadores. Esta tem vindo a provar-se ser a forma principal de investir nas várias bolsas. Desde o virar do século 20, o volume total de negócios feitos por máquinas no mercado de ações dos Estados Unidos aumentou de 15% para cerca de 80%. Da mesma forma, no mercado de câmbio, o maior mercado do mundo com mais de 6 triliões de dólares americanos em volume de negócios diariamente durante 2019, é estimado que a larga maioria do total de negócios seja também feita por computadores.

Com a possibilidade de usar máquinas para fazer negócios por nós, faz sentido considerarmos uma teoria matemática que trate de modelar preços e produtos financeiros, e desenvolver um programa que tome partido desta informação. Desde o século passado, tem-se desenvolvido também outro tipo de modelos que têm a capacidade de se adaptar, ou aprender, com a informação que lhes é passada.

O objetivo desta dissertação passa por implementar uma estratégia que tome partido da informação gerada por um modelo de aprendizagem automática. Para tal, realizou-se uma pesquisa aprofundada sobre a teoria subjacente a este tipo de modelos, que definimos cuidadosamente aqui. Para além disto, foi desenvolvido um sistema que faz os negócios automaticamente por nós, incluindo um mecanismo de *backtesting* que permite testar esta estratégia, entre outras, num ambiente simulado antes de a usar no mercado. Este sistema de negociação automático foi projetado meticulosamente para garantir extensibilidade e robustez com o intuito de explorar tantas estratégias e modelos quanto necessárias, incluindo abordagens de aprendizagem automática, baseado num conjunto de configurações definidas pelo utilizador. Subsequentemente, usámos o mercado de câmbio para correr as nossas estratégias ao vivo, que está aberto 24h por dia durante os dias de semana, e é altamente líquido. Como referência, foram também testadas outras estratégias mais comuns e a capacidade preditiva do modelo de aprendizagem automática foi comparado com um modelo matemático estabelecido, o modelo auto-regressivo integrado de médias móveis.

Palavras-chave: Aprendizagem automática, Redes neuronais, Sistema de negócios automático, *Backtesting*, Negócio algorítmico, Modelação financeira, Modelação de Séries Temporais, Análise quantitativa, Mercado de câmbio, ARIMA.

Contents

List of Figures

List of Tables

Acronyms

API Application Programming Interface.

AR Autoregressive.

ARIMA Autoregressive Integrated Moving Average.

ARMA Autoregressive Moving Average.

AUD Australian Dollar.

CAD Canadian Dollar.

CHF Swiss Franc.

CNN Convolutional Neural Network.

Conv1D 1-Dimensional Convolution.

CSV Comma-Separated Values.

EMA Exponential Moving Average.

EUR Euro.

fx/forex Foreign Exchange.

GBP British Pound Sterling.

GMT Greenwich Mean Time.

HTTP Hypertext Transfer Protocol.

JPY Japanese Yen.

JSON JavaScript Object Notation.

LSTM Long Short-Term Memory.

MA Moving Average.

MAE Mean Absolute Error.

ML Machine Learning.

mse Mean Squared Error.

NN Neural Network.

NZD New Zealand Dollar.

pip Price Interest Point.

RMSE Root Mean Squared Error.

RNN Recurrent Neural Network.

SGD Stochastic Gradient Descent.

STD Standard Deviation.

USD United States Dollar.

CHAPTER 1

Introduction

"Essentially, all models are wrong, but some are useful."
– George E. P. Box (1987)

In 1900, Louis Bachelier introduced a new domain of mathematics in his PhD thesis, one that would not see breakthrough developments until 73 years later, when Black and Scholes presented their famous option pricing formula [7]. Bachelier created from scratch many ideas behind modern stochastic analysis, including the mathematical description of the Brownian motion, so he could model stock market prices and valuate options on them.

Thus was born Mathematical Finance, the field of mathematics that contemplates modeling financial markets and their products. It is also known as quantitative finance and the application of it is usually called financial engineering. The natural extension of applying mathematical finance nowadays is computer science, as it is through it that the statistical and numerical models, imagined and designed by mathematicians, are given life. There are those who find the grass equally green on both sides of the proverbial fence, so we also have computer science models that are based in mathematics. It is models of this type that will be the subject of our study, without forgetting the mathematical theory behind it, and apply them in finance, thus touching all three corners of applied mathematical finance.

1.1 Objectives

The purpose of this dissertation is twofold. Firstly, we aim to produce a completely automated trading system, with support for simulated and live trading, which will allow us to compare trading strategies in different environments. During testing we will seek to adjust our strategies and models in controlled conditions, while trading in the live market will allow us to measure them based on their performance in the real world. The second objective of this study is, using our trading program, to verify if we can with a computer science model produce a result at least as good as mathematical models (either simply prediction errors or performance in a strategy).

1.2 Structure

In the following chapters, we will examine three entirely different fields of study and expose how they can be connected. As said above, they are: Computer Science, Finance and Mathematics. All of them are essential tools for a fully fledged study and/or practice of mathematical finance. We will go over the most important details on each of them, building a theoretical basis in which we will rely when building a custom-made automated trading system.

The way we will structure this document will retain this idea, with each chapter detailing what we will use from each field, having in themselves fundamental concepts specific to the relevant domain. However, as is the purpose of this text, there will be often a crossover of subjects.

Chapter 2 (Foreign Exchange Market) In this chapter we will detail how the foreign exchange market operates, which financial products are traded and how trades resolve.

Chapter 3 (Algorithmic Trading) Here we will discuss what it means to automatize trading, which strategies can be used and how to test them.

Chapter 4 (Description of the Algorithmic Trading System) The software built for the purpose of this dissertation will be described in full detail in this chapter.

Chapter 5 (Statistical Models in Time-Series Forecasting) This chapter will establish a mathematical basis for the model we will use as a benchmark for predictions.

Chapter 6 (Neural Networks) Focusing on computer science topics, this chapter will present how we can construct machine learning models for regression, starting from the most elementary components.

Chapter 7 (Results) The final configurations for the strategies and models and the results obtained with them, along with their performance in the market will be displayed here.

Chapter 8 (Conclusion and Future Work) In the final chapter we will discuss our findings and some ideas for the future.

One common presence we will find throughout chapters are algorithms or procedures. These are displayed in pseudo-code and we will present hereunder some conventions used as to make reading them easier. When two items are separated by a dot, it means the second item is an attribute (value or property) or a method (a sub-routine that performs an action which might or not return something based on an input) of the first item. As an example, an attribute of a model can be a parameter, denoted by *model.parameter*, and a method might be predicting a value, with the respective outcome being represented by *model.predict(input)*. Also, when an item is followed by square brackets, it means it is a vector or a dictionary, this being indicated by the type of item we are using to access its content. For instance, we can have a dictionary of vectors representing the price history for different assets: $prices[asset_1][0]$ is referencing the first price (position 0) in the list of prices for asset_1.

Also, throughout the document we will define many topics. Whenever a word in the middle of the text is presented in bold, a definition for it will presented below.

CHAPTER 2

Foreign Exchange Market

Currencies are stated using the ISO 4217, a standard first published by International Organization for Standardization in 1978, which delineates currency designators, country codes (alphabetic and numeric), and references to minor units [36]. The alphabetic code is the one we will use throughout this text. It consists of three letters, where the first two are the same as the ISO 3166 code for country names, and the third is the first letter of the currency name, where possible (*e.g.* USD stands for United States Dollar, but EUR stands for Euro).

2.1 Currency Pairs

The market for currency trading, the foreign exchange (abbreviated as forex or fx) market, is currently the largest financial market in the world, with the daily volume of trades reaching 6.6 trillion USD in 2019 [18].

When trading fx, we are always buying a currency and selling another simultaneously [8]. Therefore, the "price", or quotation, for a currency is always given in relation to another currency and is more precisely called an exchange rate. It follows that, in the forex market, we are not actually buying or selling a currency, but rather trading a **currency pair**. We will, throughout this text, often refer to trading currencies as buying or selling their respective currency pair, also referred to as a fx instrument or simply instrument when unambiguous [35].

Definition 1 (Currency Pair) *A currency pair is defined by a base currency c_b and a quote currency c_q, and is represented by concatenating the ISO currency codes, often separating them with a slash (c_b/c_q) or another symbol [36]. The exchange rate r for the pair c_b/c_q means that 1 unit of c_b can be exchanged for r units of c_q. Buying the pair c_b/c_q means* *that we are exchanging c_q for c_b and conversely selling the pair translates to exchanging c_b for c_q [8].*

The currency pairs composed of main currencies (EUR, CHF, GBP, JPY, NZD, CAD and AUD) versus USD are called majors [2]. Currency pairs that do not include USD are called crosses, like NZD/JPY. The pair configuration is just the information about how these currencies are traded, that is, which one is presented as base and quote currency.

The exchange rate of the pairs (like just about any financial product) is quoted slightly differently for both sides of the trade [41]. This means that if, for instance, we are trading EUR/USD, we will see a marginally different rate whether we want to exchange EUR into USD or USD into EUR. The two quotes are the **ask** and the **bid** prices for the pair, respectively.

Definition 2 (Bid-ask quotes) *At any given point in time, for a given moderately liquid security, we can assume there is a number of orders in the market for buying and selling said security. In fact, this is assured by market makers which are institutions or individuals that have the job of actively providing bid and ask quotes for a large array of securities [8, 17].The highest price of all unfulfilled buy orders is called the bid price for the security and the lowest price of all unfulfilled sell orders is called the ask price. The difference between the bid and ask price is called the bid-ask spread [41]. Crossing this spread is what drives the price of a security up or down.*

It follows from the definition above that the bid is always strictly smaller than the ask, thus the bid-ask spread is always positive. We will discuss below why we can safely assume this.

As an example, let us imagine there is a large number of orders in the market to sell 1 EUR for 1.1867 USD but only one offering to sell 1 EUR for 1.1866 USD (ask) and a large number of orders to buy 1 EUR for 1.1864 USD (bid, so the current bid-ask spread is 2 **pips**). We are in the market for 1 EUR and one of these buy orders is ours. If we decide that we need to buy 1 unit of EUR right now instead of waiting for our order to be fulfilled by someone willing to sell at the current bid price, we might "pay" the spread and immediately fill someone else's order. Specifically, we will of course fill the lowest ask, so the the order to sell 1 EUR for 1.1866 USD is filled and disappears from the open market, making the new lowest ask price 1.1867 USD for 1 EUR, effectively driving the price of EUR/USD up.

Definition 3 (Pip) *A pip is short for Price Interest Point and it is an unit of measurement usually used to express price moves in foreign exchange rates. It is an absolute value (as opposed to a percentage of a certain value) defined as 1% of 1% for most pairs, which equates to 0.0001 units [8]. For JPY pairs, for instance, one pip means 0.01 units. For example, if the ask price of EUR/USD at t_0 is* 1.1783 *and at t_1 the ask for the same pair is* 1.1788*, we say that from t_0 to t_1 the EUR/USD ask price increased by 5 pips.*

2.2 Buying and Selling Pairs

Using the pair EUR/USD as an example, we will detail how buying and selling currency pairs resolve, taking into account both sides of the trade. Lets say that the current bid/ask is 1.1842/1.1843 (spread equal to 1 pip).

Imagine that we have a forex portfolio with a balance of 118.43 US dollars. Buying 100 units of the pair EUR/USD means, in practice, that we are using our own US dollars to buy euros. When placing a buy order in the market, we either wait for it to be filled, or fill an existing sell order. Either way, when it settles, our account will have $118.43 less and 100€ more. In this case, a simplified view of our portfolio would look like this:

Table 2.1: Portfolio after buying EUR/USD.

Currency	**Units**
EUR	100
USD	0

In this case, we say that we have a **long** EUR/USD position (more precisely, we are long EUR and USD neutral). Selling back the 100€ we just bought, assuming the same quotes, would yield us only $118.42, costing us the bid-ask spread.

Lets consider now that we had instead only 68.43 US dollars in our account, but would still want to buy 100 units of EUR/USD. We could try to borrow 50 US dollars from someone, for a fee, and using them to buy euros. In that case, our portfolio would show:

Table 2.2: Portfolio after borrowing USD to buy EUR/USD.

Currency	**Units**
EUR	100
USD	-50

Here, we still have a long EUR/USD position, but have instead a **short** position in USD, despite being long EUR, since we have to give $50 back to our lender.

Finally, lets exemplify a case where we want to have a short position in EUR/USD. Starting with the same $50, if we want to increase our long USD position to $100, we would need to exchange $1.1842 \times 50 = 59.21$ euros into US dollars. Borrowing this amount, we would sell the same units of EUR/USD. The resulting portfolio would be:

Table 2.3: Portfolio after borrowing EUR to sell EUR/USD.

Currency	**Units**
EUR	-59.21
USD	100

This time we are short EUR/USD as intended, having a short EUR position while being long USD.

Definition 4 (Long and Short positions) *A long position in a security in a given trade is defined by holding a positive amount of units of said security. A short position is inversely holding a negative amount of units [43]. This can be achieved by borrowing the security and then selling it, keeping in mind that we will have to buy it back to return it to our lender [17]. Longing or shorting a security can also more generically mean buying or selling it, respectively.*

If we imagine that we want to be neutral (holding 0 units) on this security at some point in the future, being long means first buying and then selling it, while being short implies selling first and then buying it back.

2.2.1 Symmetry in the Quote

Let c_b and c_q be two currencies, a base currency and a quote currency respectively, forming the pair c_b/c_q with an exchange rate r. We can equivalently define the currency pair c_b/c_q as c_q/c_b with an exchange rate $1/r$. Expressing the exchange rate as the amount of base currency needed to exchange for one unit of the quote currency is called an indirect quotation.

Although it might be necessary to use the rate $1/r$ when trading the original pair, using indirectly quoted pairs should be avoided, as it is important to keep standards across countries where the base currency is different [8]. For instance, imagine that someone in the Eurozone where the base currency is EUR is trading EUR/USD with someone in the US where he could assume the base currency to be USD. If the trader in the US says that he wants to short USD/EUR, it could be confusing to his Eurozone counterpart, as that would mean he wants to have a long EUR/USD position, that is, he wants to buy euros.

The standard convention for currency pairs is that for all currencies, the USD is the base currency, with exception to NZD, AUD, GBP and EUR which are the base currency against USD [8]. Also, for any currency pair that contains the euro, EUR is the base currency.

2.3 How to Calculate Profit and Loss

Let $C = \{c_1, \ldots, c_n\}$ be a set of currencies. For two currencies $c_i, c_j \in C$, we will assume that when defining the pair c_i/c_j, c_i is the base currency and c_j is the quote currency according to the standard convention. The amount of units traded of the pair c_i/c_j are denoted by u_i, which corresponds to the amount of c_i units bought or sold.

We will define the spot price at instant $t \in \mathbb{R}_0$ for the pair c_i/c_j by the vector $(B_{i,j,t}, A_{i,j,t})$, stating the bid and ask prices, respectively. The spread is given by $s_{i,j,t} = A_{i,j,t} - B_{i,j,t}$.

In the forex market, the bid price $B_{i,j,t}$ is the number of c_j units bought by the seller of c_i, per unit sold. The ask price $A_{i,j,t}$ is the number of c_j sold by the buyer of c_i, per unit bought.

As stated earlier, we consider the spread $s_{i,j,t} = A_{i,j,t} - B_{i,j,t} \geq 0$, otherwise a negative spread would create an arbitrage opportunity. This means we could buy at $A_{i,j,t}$ and immediately sell at a higher price of $B_{i,j,t}$, which would entail a counterpart to be willing to sell at a lower price, $A_{i,j,t}$, than what the market is quoting for buying, $B_{i,j,t}$. If this would actually happen in practice, everyone would immediately take advantage of it, making the arbitrage opportunity disappear almost as soon as it was created. Another way to look at it is that the buyers are bidding at a higher price than the minimum that the sellers are asking or, equivalently, the sellers are asking a lower price than the bids offered. In sum, there is no logical reason for it to happen, and even if it did, the market would correct itself faster than we could take advantage of it.

We will consider a function $r : \mathbb{R}^3 \to \mathbb{R}$ yielding the profit or loss for a given trade which, assuming the deal is eventually closed, is always obtained by subtracting the amount we paid when buying from the value we received when selling. Therefore, for a 3-tuple consisting of the units u bought or sold, the price O at which we opened the trade and the price C at which we closed it, we can define the return function as:

$$r(u,O,C) := \begin{cases} u(C-O) & \text{if long position} \\ u(O-C) & \text{if short position} \end{cases}$$

While the trade is live, we can estimate how much we are gaining or losing with the trade, and this is said to be unrealized profit or loss. When we actually close the trade, the return obtained is the realized profit or loss.

2.3.1 Long Positions

At the current moment t_0, the market quotes the pair c_i/c_j at $(B_{i,j,t_0}, A_{i,j,t_0})$ and we decide, during this instant, to buy u_i units of the pair c_i/c_j, *i.e.* we want to be long c_i/c_j. This means we are accepting to sell $u_i A_{i,j,t_0} = u_j$ units of c_j to buy u_i units of c_i at the price A_{i,j,t_0}. As soon as the trade settles, we can compute the unrealized profit or loss of this deal at instant $t \geq t_0$, in the quote currency, by determining how many units of c_j we can buy, by selling the u_i units of c_i we bought: $r(u_i, A_{i,j,t_0}, B_{i,j,t}) = u_i(B_{i,j,t} - A_{i,j,t_0}) = u_i B_{i,j,t} - u_j$.

At some point in time, $t_h \geq t_0$ we decide to close the deal, so the realized profit or loss of the deal is $r(u_i, A_{i,j,t_0}, B_{i,j,t_h}) = u_i B_{i,j,t_h} - u_j$.

Now, if we have an account in currency c_k and assuming we made a profit, we just sell the $u_i B_{i,j,t_h} - u_j$ units of c_j and buy c_k, which we can do by buying the pair c_k/c_j. In our account's currency, the return of the trade is given by $(u_i B_{i,j,t_h} - u_j) A^{-1}_{j,k,t_h}$, since we are looking to convert a specific amount of c_j units using a pair where it is the quote currency, following the rationale in 2.2.1.

2.3.2 Short Positions

Similarly, at t_0 the market quotes the pair c_i/c_j at $(B_{i,j,t_0}, A_{i,j,t_0})$ and we want now to short c_i/c_j and sell u_i units of the pair c_i/c_j. This time we will buy $u_i B_{i,j,t_0} = u_j$ units of c_j while selling u_i units of c_i at the price B_{i,j,t_0}.

As before, after some time we decide to close the deal, at $t_h \geq t_0$. The realized profit or loss is $r(u_i, B_{i,j,t_0}, A_{i,j,t_h}) = u_i(B_{i,j,t_0} - A_{i,j,t_h}) = u_j - u_i A_{i,j,t_h}$.

Let us assume, as before, that our account is in a different currency, c_k, but that our trade has now resulted in a loss. In this case, we will proceed analogously and and buy $|u_j - u_i A_{i,j,t_h}|$ units of c_j (so we end up c_j neutral) while selling c_k (realizing the loss in our currency), achieved by selling the pair c_k/c_j. In our account's currency, the return of the trade is then given by $(u_j - u_i A_{i,j,t_h}) B^{-1}_{k,j,t_h}$.

2.4 Hedging Positions

Since usually the market quotes prices with a very small spread, at least for the majors, it is possible to make almost perfect **hedging** as we will see.

Definition 5 (Hedging Forex Risk) *Hedging is part of an investment strategy that aims to manage some type of risk in said strategy [17]. One class of risk that naturally occurs in forex trading, and one that we might want to hedge, is the foreign exchange risk. In our case, it is the risk associated with trading pairs denominated in a currency different than our account's currency. For example, if we are trading GBP/USD and our account's currency is in EUR, we are exposed to the performance of EUR/GBP since we will have to convert the profit or loss in GBP to our account's currency. A strategy that hedges fx risk will make additional trades for the purpose of minimizing the undesired exposure.*

We say the broker allows hedging when it is possible to open a long and short trades without the broker making any arrangements in our exposure [6]. For instance, if a broker allows hedging it means that we can open two trades in EUR/USD, one buying x euros and another selling x euros, and maintain both open (at the loss of the spread, of course). If the broker does not allows for hedging, the second trade will be consolidated with the first and, in this case, it will close both trades. In our case however, with the broker we will use it is possible to do this.

Basing on the example above of trading EUR/GBP, lets consider three currencies, c_i, c_j and c_k, the latter being our account's currency. Lets assume that the standard convention for the pairs is $c_i/c_j, c_i/c_k$ and c_j/c_k, and that we will close all trades at some point in the future. If we want to buy u_i units of c_i/c_j at t_0, we first have to spend some of our own currency to buy c_j, which we will sell for c_i. At $t \geq t_0$, we are exposed to c_j/c_k:

Table 2.4: Naked long position on c_i/c_j

Time	**Direction**	**Pair**	$\boldsymbol{c_i}$ **units**	$\boldsymbol{c_j}$ **units**	$\boldsymbol{c_k}$ **units**
t_0	buy	c_j/c_k	0	u_iA_{i,j,t_0}	$-u_iA_{i,j,t_0}A_{j,k,t_0}$
t_0	buy	c_i/c_j	u_i	$-u_iA_{i,j,t_0}$	0
t	sell	c_i/c_j	$-u_i$	$u_iB_{i,j,t}$	0
t	sell	c_j/c_k	0	$-u_iB_{i,j,t}$	$u_iB_{i,j,t}B_{j,k,t}$
Total			0	0	$u_i(-A_{i,j,t_0}A_{j,k,t_0} + B_{i,j,t}B_{j,k,t})$

Indeed, our unrealized profit or loss in c_k, $u_i(-A_{i,j,t_0}A_{j,k,t_0} + B_{i,j,t}B_{j,k,t})$, scales freely with respect to the bid price for the pair c_j/c_k at t. This price is unknown, and we do not know how much it will affect our profit or loss. Thus, we are exposed to its performance, while we intended to be solely exposed to the performance of the pair c_i/c_j.

However, we could hedge this position and lock in the price we pay for converting in pairs that are not denominated in our account's currency. To do this we can sell the conversion pair when we open our main trade, thus simultaneously selling and buying our currency for c_j. With this strategy, our positions at t are:

Table 2.5: Hedhed long position on c_i/c_j

Time	**Direction**	**Pair**	$\boldsymbol{c_i}$ **units**	$\boldsymbol{c_j}$ **units**	$\boldsymbol{c_k}$ **units**
t_0	buy	c_j/c_k	0	u_iA_{i,j,t_0}	$-u_iA_{i,j,t_0}A_{j,k,t_0}$
t_0	sell (hedge)	c_j/c_k	0	$-u_iA_{i,j,t_0}$	$u_iA_{i,j,t_0}B_{j,k,t_0}$
t_0	buy	c_i/c_j	u_i	$-u_iA_{i,j,t_0}$	0
t	sell	c_i/c_j	$-u_i$	$u_iB_{i,j,t}$	0
t	sell	c_j/c_k	0	$-u_iB_{i,j,t}$	$u_iB_{i,j,t}B_{j,k,t}$
t	buy (hedge)	c_j/c_k	0	u_iA_{i,j,t_0}	$-u_iA_{i,j,t_0}A_{j,k,t}$
Total			0	0	*

$$
\begin{aligned}
* &= u_iA_{i,j,t_0}(B_{j,k,t_0} - A_{j,k,t_0}) + u_i(-A_{i,j,t_0}A_{j,k,t} + B_{i,j,t}B_{j,k,t}) \\
&= u_iA_{i,j,t_0}(B_{j,k,t_0} - A_{j,k,t_0}) + u_i(-A_{i,j,t_0}A_{j,k,t} + [A_{i,j,t_0} + (-A_{i,j,t_0} + B_{i,j,t})]B_{j,k,t}) \\
&= u_iA_{i,j,t_0}(B_{j,k,t_0} - A_{j,k,t_0}) + u_i(-A_{i,j,t_0}A_{j,k,t} + A_{i,j,t_0}B_{j,k,t} + [-A_{i,j,t_0} + B_{i,j,t}]B_{j,k,t}) \\
&= u_iA_{i,j,t_0}(B_{j,k,t_0} - A_{j,k,t_0}) - u_iA_{i,j,t_0}(A_{j,k,t} - B_{j,k,t}) + u_i(-A_{i,j,t_0} + B_{i,j,t})B_{j,k,t} \\
&= -u_iA_{i,j,t_0}(s_{j,k,t_0} + s_{j,k,t}) + u_i(-A_{i,j,t_0} + B_{i,j,t})B_{j,k,t}
\end{aligned}
$$

While being far from perfectly hedging the transaction fx risk, we have mitigated it. In fact, considering the resulting profit or loss in c_k above, we see that the our initial purchase of c_i/c_j has virtually no exposure to c_j/c_k, costing us only the spreads s_{j,k,t_0} and $s_{j,k,t}$. The part we could not hedge, is the profit or loss in c_j (the units sold for c_i minus the units bought back with the profit or loss in c_i) which evidently has exposure to c_j/c_k.

Using the same reasoning, we can analogously extend this idea for hedging short positions.

CHAPTER 3

Algorithmic Trading

3.1 What is it and how does it work?

Putting it quite briefly, algorithmic trading is the autonomous trading of securities based on the decisions of a programmed algorithm [4]. Traders might design and implement strategies that they themselves would not be able to perform if they are after an high-frequency solution, such as placing thousands of orders a second or reacting to market moves within milliseconds. This low latency in reaction time is so precious that a network company stealthily spent an astounding amount of effort and money to build an 1300-kilometer tunnel housing a fiber-optic cable between Chicago and New Jersey [25]. Unveiled in 2010, this tunnel runs through mountains and under rivers, in order to be absolutely straight, and its purpose is to reliably connect the Chicago Mercantile Exchange and the Nasdaq data center with the lowest possible latency. The access to this network was then commercialized and made accessible to high-frequency trading houses which paid very well for a reduction of 17 to 13 milliseconds in latency.

The work described in this text does not try to delve in this kind frequencies but we will still be placing multiple orders in a single day, which is called intraday trading [4]. This is a type of trading that aims to profit off of small price changes during the day. This can be done by an human as well as a computer but our purpose is to find if we can use the high processing speeds of a machine to our advantage and therefore we will build a software to perform this kind of trading in the forex market.

Orders in algorithmic trading are placed when a set of conditions are verified or flags are raised, which are programmed according to a certain strategy [4]. For instance, we could simply say that when EUR/USD closes a **candle** with a 2 pips drop, we will buy the instrument (meaning financial instrument, in this text used as synonym to a currency pair) and sell it at the close of the following candle whether we have a profit (to realize it) or a loss (to stop further losses). This is a extremely simplistic strategy of course and would likely lead to significant losses over time.

Definition 6 (Candle) *A candle, or candlestick, of a given instrument and time-frame is an object that represents the open, high, low and close prices of that instrument during the given time-frame [43]. For example, if we take EUR/USD in 1 minute time-frames, then if we look at the latest candle we will find the first price that the candle took into consideration (open), the highest and lowest values that EUR/USD achieved during the 1 minute that followed the open and the last price during this 1 minute time-frame (close). They usually have a look similar to this:*

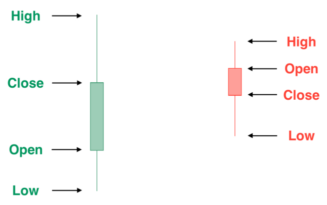


Figure 3.1: A pair of candles. The candle on the left symbolizes a net price increase while the one on the right indicates that during its time frame the price decreased.

Note that the difference in colors denotes if during the time-frame the price increased or decreased. In other words, the color difference allows you to identify the close from the open.

Our objective is then to implement a few well known and simple strategies and compare their performance against a prediction obtained via a neural network. How we will execute and where we will place the orders raised by our algorithm is another important topic. Usually small investors or individuals like us do not have direct market access so in order to trade the wide array of securities available in the market, someone has to trade them on our behalf; this is the business of a broker [19]. This is how we will enter the forex market, we will decide which orders we want to place and communicate them to a broker so they can execute them for us. In our case, we chose to go with OANDA which is a well established US broker dedicated to foreign exchange, regulated by top global authorities and has a good track-record for financial transparency [28]. Our communications with OANDA will be completely electronic and for that we will rely on their own **API**.

Definition 7 (API) *API is an acronym for Application Programming Interface, which is an intermediary software that defines the interactions between different applications in order to build an application that integrates another.*

3.2 Strategies as benchmark

The strategies we will use to benchmark the performance of the neural network model will be mostly based on simple **indicators** such as moving averages while taking into account, for instance, the recent volatility.

Definition 8 (Indicator) *A technical indicator is an analysis tool consisting of one or several values derived from a number of financial information such as the historical price, volume or volatility [43]. They are mostly used in technical analysis, a methodology for forecasting future price moves based on charts, indicators and patterns.*

3.2.1 Buy and Hold

One strategy we will use as a benchmark is a simple buy and hold where we buy a given instrument at the start of our trading activity and hold until the end regardless of how the market behaves. This is important to have into account and to not overlook since we want to make sure that at least we can match the performance of the market by replicating it via a passive strategy. This means that, if our trading horizon is 1 day and we are trading EUR/USD, if this instrument has increased by 7 pips at close of business, then this strategy has earned us 7 pips if we realized our profits, net of the **broker fees** and the bid-ask spread.

Definition 9 (Broker fees) *A broker fee is the commission the broker charge their clients for its service. They could take many forms such as being linked to each transaction as percentage of its value, a flat rate on deposits or withdrawals from the brokerage account or incorporated into the bid and ask prices offered in their platform [19].*

In our case, having OANDA as a broker, we will pay fees when withdrawing from the brokerage account and if we remain inactive for more than 12 months [31]. Also, and most importantly, we will pay a flat rate of +/- 0.5% when a trade settles and we are delivered or owe a currency that is different than our account's denomination (EUR).

3.2.2 Exponential Moving Average Retest

By contrast, another strategy we will implement has an active management of orders and price monitoring. It is going to be a type of scalping, which is a group of strategies that try to profit of small price movements [6]. The indicator we will use for that is a moving average that gives a bigger weight to the most recent values, with an exponential decay. Specifically, we will implement the following algorithm to search for entry points for long positions, based on the ideas in [6]. The entry points for short positions are obtained by reversing the inequality signs.

Algorithm 1 Exponential moving average (EMA) retest for long positions

```
while true do
  if flagged then
    if current_candle.ask.close > current_candle.ask.open then
      create_order()
    end if
    flagged ← false
  end if
  if previous_candle.ask.close > previous_ema and
  current_candle.ask.open > current_ema > current_candle.ask.low and
  current_candle.ask.close > current_ema then
    flagged ← true
  end if
end while
```

The details of the orders to be placed according to each strategy are also specified by them. We will go over how the order are structured for all strategies used, including the ones above, in Chapter 7.

3.3 Backtesting

Having the strategies we plan on using implemented, we want to have a close idea of how they are going to perform in the market. One of the advantages of algorithmic trading is to be able to easily and quickly test a strategy using a historical data feed to assess how said strategy would have performed in the past [4, 19]. Not only that, it is also a great way to check if what we intended to implement as a strategy is being correctly executed and to fine tune its details. We will build our algorithmic trading system in such a way as to allow it to take a feed of data from either OANDA or a backtesting channel despite them being in quite different formats, thus enabling us to apply strategies seamlessly whether we are benchmarking or live-trading a strategy. This will simply be done by converting both data streams to a format that is friendly to our program.

While being of great use, we should be wary of some ways of how backtesting might work against us [5]. We are trying to benchmark our strategies and infer if and by how much they would be profitable had we run them in the past. If we perform this evaluation incorrectly, we might conclude that our strategy is ready for live trading based on good historical performance while in fact it would have yielded worse results if we had actually run it during the same period of time. We want therefore to minimize the performance difference from backtesting and running the strategies in the market, at least theoretically since this is hard to measure, and we will do so by avoiding some of the bias that could lead to this. Below we will detail some well-known types of bias that are common issues in erroneous backtesting [4].

3.3.1 Look-ahead bias

This problem appears when the algorithm we have implemented tries to use some price in the future and thus having information that would not have been available during a live run [4, 5]. If this type of bias is present, it is likely originating from a failure in the logic of the strategy or a programming oversight. For instance, if the strategy states that we should place a sell order when the market price is within 5% of the day's high, either it is referring to the high from previous day or it is introducing look-ahead bias since surely we would have no way of knowing in advance, during the day, what will be the maximum price reached. Assuming it is indeed referring to yesterday's price, look-ahead bias could still be introduced simply by a wrong indexation of the backtesting data. It is quite straightforward to avoid though, if we make sure that the engine that processes the data stream is exactly the same for both backtesting and live trading, the only difference being the source of the data, our program should be free of any look-ahead bias.

3.3.2 Data-snooping bias

Most of the strategies we can find or come up with have some degree of freedom, meaning that there are parameters that are subject to tuning. Data-snooping bias might appear if we overfit these parameters to obtain good results during backtesting, overly optimizing the strategy for a restrict time-frame or dataset [4, 5, 19]. Since the market has a great deal of randomness, we cannot expect it to behave exactly as it did in the past, leaving our overtuned strategy with a small chance to perform well in a market with a different regime. It might be quite difficult or even impossible to completely avoid this type of bias since fine-tuning the strategies' parameters is itself an objective of backtesting. What we can do to minimize its effect, other than reducing the number of free parameters, is to assure we have a large enough sample size compared to the number of parameters and perform true out-of-sample tests. This means we should divide our backtesting data in two periods and keep the latest one as out-of-sample data, tuning the strategy using the first part. Hopefully the strategy will perform reasonably well in the second part of the backtesting data, but if not, we should either scrape it or simplify it by removing free parameters, since trying to optimize the strategy to also fit the second period of the data would turn the out-of-sample into in-sample data.

CHAPTER 4

Description of the Algorithmic Trading System

In this chapter we will describe in detail the foundations of the software written in Python for the purpose of this text, recurring to pseudo-code, presenting the main ideas and workflow.

It comprises over 1500 lines of code and was designed to be a complete but easily expandable infrastructure. The Python version used was 3.8.5 and we made use of the following packages, aside from the base libraries:

matplotlib (3.3.3) Used to plot dynamic graphics showing the performance live over time;

numpy (1.19.4) Needed for many numerical and vector operations with the advantage of being highly efficient;

oandapyV20 (0.6.3) The wrapper used for version 20 of OANDA's API;

pandas (1.1.4) Enables the creation and handling of dataframes, a flexible data-structure with many available tools;

plotly (4.12) Used to create interactable graphics;

statsmodels (0.12.1) Contains everything we might need to support statistical models;

tensorflow (2.3.1) Provides tools to easily create machine learning models, allowing low- and high-level customization, as well as great results visualization capabilities.

4.1 Main Routine

We start off by reading a JSON (JavaScript Object Notation) file containing all the configurations we need for the current run and proceed to initialize loggers for indicators, prices and candles data as well as errors that might occur. The initial configuration file is checked for any possible conflicting settings and the necessary changes are made. Then, four manager objects that handle different topics during the run are initialized. We will go into them with more detail but, in summary, we have:

Broker Manager Responsible for handling the broker connection (be it OANDA or a backtesting channel) and is the sole segment of the program that has any knowledge whether it is running on live or past data, without having any say on decisions, thus completely avoiding look-ahead bias during backtesting.

Indicator Manager Sets up and updates the indicators currently in use, by instrument. It also provides the necessary indicator data to the strategy manager.

Strategy Manager Creates all the strategies, for each instrument, according to the initial configuration and assesses, according to each strategy, if a order should be placed each time a new candle is created for a given instrument. It also yields the orders to the broker manager so it can place them in the market.

Asset Manager Holds information regarding the assets available in the trading account, provided by the broker manager, and sizes the trades accordingly.

After this, we start iterating the data stream and act on it as follows:

Procedure Main routine

```
broker_manager.sync_time()
broker_manager.initialize_history()
broker_manager.initialize_ticks()
$n \leftarrow 0$
for each response in broker_manager.data_stream() do
  broker_manager.check_orders()
  asset_manager.update_balance(broker_manager.balance)
  if response.type is price then
    price_logger.log_price(response.price)
    $n \leftarrow n + 1$
  else if response.type is candle then
    indicator_manager.update(response)
    strategy_manager.assess(response)
    for each order in strategy_manager.orders do
      broker_manager.create_order(order)
    end for
    $n \leftarrow n + 1$
  end if
  if $n \geq$ maximum_requests then
    broker_manager.terminate_stream()
  end if
end for
broker_manager.liquidate_positions()
```

This is the core of what our software will do, recurring of course to other modules that aid the main routine. The graph below shows how the main modules communicate, what information they exchange and classes organization. Some of these items we have not discussed yet, but all will be detailed later.

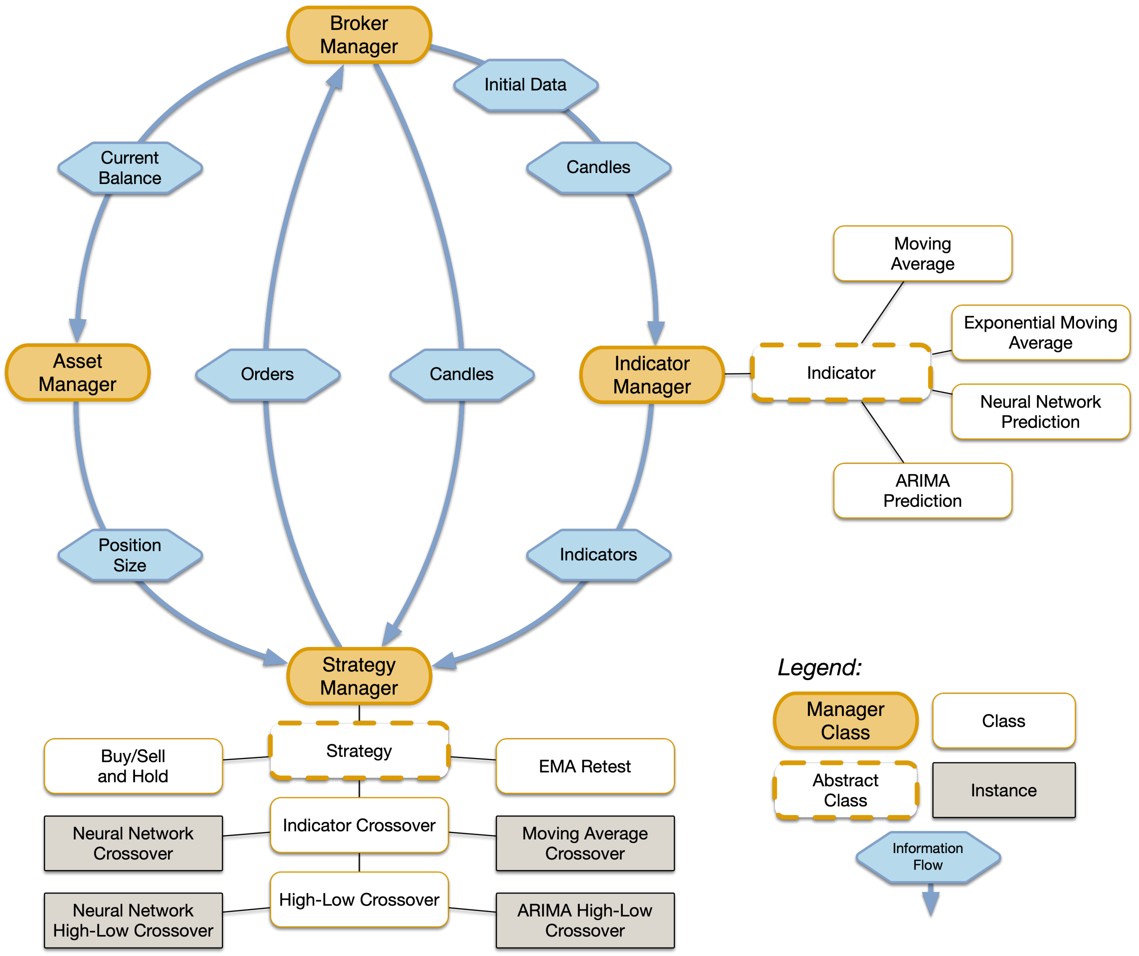


Figure 4.1: The main modules in the trading system and how they interact. All non-abstract classes have indistinct instances of themselves, if not indicated otherwise. The connections from the Indicator manager

The configuration file read at the start of the program allows us to set a wide array of values allowing to fine-tune how we want it to run. Specifically, for each run we can opt to change the following settings:

run_type This dictates whether the system will run in backtesting or live trading mode;

backtest_split The percentage of data to be used as train/past data in backtesting mode. The remaining data will be used to measure performance.

backtest_balance The starting balance in the backtesting mode.

backtest_data_paths A dictionary containing paths for the comma-separated values (CSV) files containing price data to use in the backtesting mode

oanda_token A personal token which allows us to use the OANDA API.

oanda_account_id The ID of the OANDA account we want to use.

instruments An array of currency pairs to be monitored in the form of strings, with the base currency and the quote currency being separated by an underscore (*e.g.* EUR_USD). This will be the notation throughout the program.

max_candles The maximum amount of candles to be kept in broker manager's history.

look_back An integer denoting the amount of candles to be retrieved in an initial request of recent history in order to initialize the indicators that are going to be used.

granularity The granularity of the candles to be requested in the initial data request described above.

candle_size Specifies the size of the candles the broker manager will create in time in seconds (*i.e.* if this is set to 60 for instance, then a new candle will be created when 60 seconds have passed since the last one).

indicators A dictionary that has in turn a dictionary mapped to each indicator name that is implemented, containing generic items needed for the initialization such as specifying whether the given indicator will be used in this run or what type of input it takes, as well as specific items for each indicator such as the window size for the moving average.

strategies Also a dictionary that contains the names of the strategies implemented, similarly mapped to a dictionary of generic and specific settings for each strategy.

max_records An integer specifying the maximum amount of responses we will receive in this run before terminating the stream.

trade_size_type Whether the size of the trade is computed by the asset manager based on a fixed percentage of the available amount of capital or it is computed by other means.

base_percentage_amount The percentage of our available capital to be used in each trade if the trade size type is set to "fixed".

flat_amount The flat amount to be used in each trade if the trade size type is set to "flat".

logging Contains a dictionary of settings specific for logging purposes, in particular sets the level of logging we want to display in the console during the run.

4.2 Broker Manager and Data Structures

This is the first manager we initialize and is the one that contains the bulk of data processing and logic. This is because it effectively acts as a broker to the rest of the program, being the single piece of code that knows whether we are running a test on historical data or live trading. When initializing the broker manager we first set the few attributes that are common between the two types of runs, like the run type, maximum amount of candles, the size of each candle, and the maximum number of records. We then proceed to initialize the attributes that are specific to the current run type. As with the attributes, the methods (although they are not distinct so the rest of the code sees no difference) have very different implementations between run types. For this reason, we will detail how the broker manages functions for each run type separately.

4.2.1 Live Trading

For a live run, the broker manager keeps track of items such as the connection token, the OANDA account ID, the current session and the pricing stream. It also holds a tick collector for each instrument and a list of the orders placed for each strategy and instrument. After initializing the broker manager, the main routine initializes the indicator manager, which in turn creates all the indicators used by the strategies we want to run. In order to estimate the initial values for these indicators, some recent past data is required, and it is the broker manager who provides it. During a live run, the past data is retrieved from OANDA. We request both ask and bid candles and aggregate this information in dataframes, one for each instrument, which are then passed to the indicator manager's constructor. Each response from these requests is a standardized structure, common for all candle requests [29].

Structure 1 (Candle response) *Being* **g** *the granularity and* **r** *the rate type from the configuration file and n the number of instruments in the configuration file (i.e. the length of the instruments array from the configuration and the number of elements in the initial* *data), for $i \in \{1,\ldots,n\}$ the ith candle response is structured as follows*

$$
\begin{aligned}
response_i = \{ &\text{“}instrument\text{”} : instrument_i, \\
&\text{“}granularity\text{”} : \boldsymbol{g}, \\
&\text{“}candles\text{”} : [\\
&\quad \{ \text{“}complete\text{”} : \boldsymbol{c}_1, \\
&\quad\ \text{“}volume\text{”} : \boldsymbol{v}_1, \\
&\quad\ \text{“}time\text{”} : \boldsymbol{t}_1, \\
&\quad\ \boldsymbol{r} : \{ \text{“}o\text{”} : \boldsymbol{open}_1, \text{“}h\text{”} : \boldsymbol{high}_1, \text{“}l\text{”} : \boldsymbol{low}_1, \text{“}c\text{”} : \boldsymbol{close}_1 \} \\
&\quad \}, \\
&\quad \ldots, \\
&\quad \{ \text{“}complete\text{”} : \boldsymbol{c}_m, \\
&\quad\ \text{“}volume\text{”} : \boldsymbol{v}_m, \\
&\quad\ \text{“}time\text{”} : \boldsymbol{t}_m, \\
&\quad\ \boldsymbol{r} : \{ \text{“}o\text{”} : \boldsymbol{open}_m, \text{“}h\text{”} : \boldsymbol{high}_m, \text{“}l\text{”} : \boldsymbol{low}_m, \text{“}c\text{”} : \boldsymbol{close}_m \} \\
&\quad \} \\
&\ ] \\
\}
\end{aligned}
$$

where m is the quantity of candles requested and, with $k \in \{1,\ldots,m\}$, c_k is a boolean value that states whether the candle k is complete or is ending in the future, v_k is the number of prices created during the time-range of the candle k, t_m is the start time and date of the candle k according to Greenwich mean time (GMT) and $open_m$,$high_m$,low_m and $close_m$ are their respective values for candle k.

Before starting to pull information from the data stream, the main routine calls three initial methods from the broker manager. Firstly, the sync time method makes sure we start receiving data at the start of the next time point matching the time frame of the candles we will use. It does this by simply waiting an amount of time, calculated as

$$wait\_time = candle\_size - (current\_seconds + current\_microseconds/1000000).$$

Then, we request the most recent candle for each instrument in the method that initializes the history and we do the same for the ticks, with the caveat that we might have to wait for a new tick if the latest one still belongs to the previous time frame (according to our candle size). While we have synchronized our time, this is still possible given that at most four price updates for a given instrument are provided per second and at least zero, therefore it happens that there are some seconds without price updates. The candles though, are always available after synchronizing since they close at a specific time.

After this is done, we will start iterating the responses from the *data_stream* generator. This method, in a live run, itself iterates the price stream from OANDA, assesses the quality of the responses and yields information accordingly. The format of a valid price response from OANDA follows below [33].

Structure 2 (Price stream response) *When the system is running, we are in constant connection with our broker's server and on the lookout for any new price updates. The data received in these updates is structured in JSON format, having the items on the right mapped to the names on the left.*

Table 4.1: Items contained in a pricing stream response.

Key	**Value**
time	The date and time when the response was created, according to GMT.
type	The type of response. If the type is "HEARTBEAT", no other item below this one will be included in the response and its purpose is solely to maintain the HTTP connection. Otherwise, the type will be "PRICE" and we will also have the items below.
instrument	The instrument to which the price refers to.
tradeable	A boolean indicating whether the price is tradeable or not.
bids	A list of JSON objects, one in our conditions, that have the price and liquidity available on the instrument's bid side.
asks	A list of JSON objects, one in our conditions, that have the price and liquidity available on the instrument's ask side.
closeoutBid	The closeout bid price. It is used when a bid price is required to closeout a position (margin closeout or manual).
closeoutAsk	The closeout ask price. It is used when an ask price is required to closeout a position (margin closeout or manual).

The responses from the price stream, before being passed to the main routine, are handled by the method *handle_broker_response*, which transforms the original response according to the following procedure.

Procedure Handling of broker response

```
if response.type is price then
  datapoint ← PRICE_STRUCTURE
  if response.time − tick_collector[response.instrument][0].time > candle_size then
    candle ← CANDLE_STRUCTURE
    candle.time ← now − [response.time − (response.time mod candle_size)] + candle_size
    datapoint.time ← response.time − (response.time mod candle_size)
    tick_collector[response.instrument] ← [datapoint]
    candle_history[response.instrument].append(candle)
    synthetic_response ← candle
  else
    tick_collector[response.instrument].append(datapoint)
    synthetic_response ← datapoint
  end if
else
  synthetic_response ← HEARTBEAT_STRUCTURE
end if
return synthetic_response
```

Each one of the three structures above is a different type of synthetic response created during this method, standardized for the rest of the program. These are different between themselves but all follow a generic structure, which we detail below.

Structure 3 (Broker manager response) *After receiving a response from the server, the broker manager manipulates this information and constructs a dictionary with different items depending on what type of synthetic response we want to output. The general format is the same though, with the items on the right column are mapped to the ones on the left.*

Table 4.2: Items contained in a response created by the broker manager.

Key	**Value**
time	The original response time if no candle is created, otherwise the time is set to the start of the latest candle's time frame, computed as in the procedure above, according to GMT. The format is *YYYY-MM-DDTHH:MM:SS_FFFFFF* (*e.g.* November 30th 2020 at 20 hours, 30 minutes, 15 seconds and 1864.53 milliseconds is represented by 2020-11-2020*T*20:30:15_1864.53).
type	The original response type is kept if no candle is created, otherwise the type is set to "CANDLE".
instrument *	The instrument to which the price or candle refers to.
bid*	If the type is "PRICE", this will contain the current bid price, as per the original response. Otherwise, it will contain a dictionary of the open, high, low and close bid prices of the latest candle's time frame.
ask*	If the type is "PRICE", this will contain the current ask price, as per the original response. Otherwise, it will contain a dictionary of the open, high, low and close ask prices of the latest candle's time frame.
n_ticks**	The number of ticks received during the latest candle's time frame.

*The items with * are available only in price and candle synthetic responses. The item with ** is only available for candles.*

After creating the synthetic response, it is passed to the main routine. Firstly, the *check_orders* method of the broker manager is called but since OANDA's servers handle all the work related to filling orders or canceling them, there is nothing for us to do during a live run, so there is no implementation of this method in this case.

The broker managed is then called again in case there are any orders to be created. If so, the method *create_order* is called for each one. Besides placing the order in the server, it also sets the **stop-loss** and **take-profit** prices for **market orders** according to the latest available prices and the margin allowed by the respective strategy, like so:

- **long orders**

$$stop\_loss = current\_bid - margin;$$

$$take\_profit = current\_bid + margin.$$

- **short orders**

$$stop\_loss = current\_ask + margin$$

$$take\_profit = current\_ask - margin$$

This is done so there is no way of triggering these sub-orders immediately following the execution of the parent order or, since OANDA checks for this themselves, we do it to avoid having orders being canceled by OANDA for the same reason. This is done solely for market orders since for this type of order we do not set a specific entry price, as is detailed below, for limit and stop orders the strategies set the stop-loss and take-profit prices themselves.

Definition 10 (Market, Limit and Stop orders) *A market order is an order that executed at the current market price, guaranteeing that the order will be filled if there is enough liquidity in the market [17]. As for limit and stop orders, they are used depending on whether we want to, in sum, wait for a price movement or react to it, respectively. This is because a limit order will be filled when the order price, or a more favorable one, is available [26]. Long limit orders are set below the market price while short limit orders are set above it. In turn, a stop order is filled as soon as possible when the order price is reached, that is, we will try go get into the market at any price, in the direction of the momentum, when the price threshold we set is crossed [37]. Long stop orders are placed above the market and short stop orders below.*

Definition 11 (Take-Profit and Stop-Loss orders) *Take-profit and stop-loss orders, to which we will also refer as boundary orders, are sub-types of limit and stop orders, respectively, that given a live trade, are placed with a symmetric amount of units of said trade. This is done in order to effectively unwind the first trade and get out of the market realizing the profit or loss, respectively, at a desired price point[32]. They are used concurrently to manage open positions: if the market price raises over the take-profit price, the take-profit order is filled and if it falls below the stop-loss point, the stop-loss order is executed instead. It is important to notice that these orders remain live only as long as the linked open position or unfilled order is live as well.*

Finally, when the maximum number of responses allowed by the initial configuration is reached, the *liquidate_positions* method is called, which requests a list of all currently open trades in our account and closes each one at market price, fully realizing the profit or loss for each strategy for the current run.

4.2.2 Backtesting

The purpose of the broker manager during backtests is to effectively simulate a broker for the rest of the code, so many implementations of the methods we have seen above are quite different. The overall logic behind the program does not change, only the way we obtain and create the data we have already seen does. While, for instance, during a live run the broker manager does not have to worry about order execution (since OANDA handles it and the orders always have self-triggering boundary orders so the whole trade settles and closes without any further interaction) here it must instead check whether the orders are filled according to current market prices and keep track of all pending orders. During backtesting mode, we focus on one single instrument in each run, since there we can always run the same data again with a different instrument to compare results. Thus, when being initialized, the broker manager reads from the initial configuration, among other settings, which instrument it will use and creates the backtesting channel from a CSV file, detailed below.

Structure 4 (Backtesting data) *The data we use for backtesting the system and the strategies implemented is obtained from Dukascopy, a Swiss banking group that, among other things, provides market information including access to historical price data [9]. The data we request from them is both bids and asks in 1 minute candles, which they provide for free, up to a 3-year time frame. What we get as output are two CSV files, one for the ask prices and another for the bids. Each of these files comprise over 1.1 million data points, ranging from October* 1*st* *2017 to October* 1*st* *2020. All flat days are filtered out, including weekends as well as other days where there was no price movement, since these do not provide any information. Each data point includes the following items:*

- ***Gmt time**, marking the time, according to GMT, when each candle was created;*
- ***Open**, the price at which the candle opened;*
- ***High**, the highest price achieved during the candle's time frame;*
- ***Low**, the lowest price achieved during the candle's time frame;*
- ***Close**, the price at which the candle closed;*
- ***Volume**, the amount of units traded during the candle's time frame, in millions.*

The start of each day is according to coordinated universal time and all price values have up to 6 significant digits. We take these two files for each currency and create one that combines the ask and the bid information, while dropping the volume. The volume is not considered since in the data from Dukascopy, it represents the amount of units traded during the respective minute, in millions, while in OANDA it is the number of times the price changed during the candle interval. Therefore, they are not comparable. This is the file we will use as input for performing backtests.

The open CSV file of historical data is then wrapped in a generator, yielding responses as requested by the main routine, mimicking the behavior of OANDA's pricing stream. This is the backtesting implementation of the *data_stream* method in the broker manager.

As for providing the initial data for the initialization of indicators, we simply take the first elements of the data stream and structure them in the same dataframe format as done when running in live trading mode.

Of the three first methods on the main routine, none of them has any implementation if we are running a backtest. This is because during backtesting we do not need to synchronize the time to create candles, since that is the format of our input already and since there are no ticks, there is no need to have initial values for them. As for initializing the history, we have all the data at our disposal, so there is no need to do anything else.

We therefore start immediately iterating the rows in the open CSV file, in exactly the same way we do for server responses from OANDA. The datapoints are also firstly processed by the handle_broker_response method, which in backtesting mode simply creates the same structure we have seen before for synthetic responses. The difference here is that every single response passed to the main routine are candles, since this is the format of our dataset.

The greatest difference comes when the main routine calls the *check_orders* method. While in live mode it does not have any implementation, this is how it operates during backtesting.

Procedure Checking orders

```
for each order in pending_boundary_orders do
  create_order(order)
end for
pending_boundary_orders.clear()
$i \leftarrow 0$
while i < pending_orders.length do
  order ← pending_order[i]
  if not try_fill(order) then
    if order contains life_time then
      order.life_time ← order.life_time −1
      if order.life_time is 0 then
        remove_order(order.id)
      else
        i ← i + 1
      end if
    end if
  else
    i ← i + 1
  end if
end while
```

First, we create any pending boundary orders resulting from resolving executions of parent orders that occurred during previous candles. They are kept in a separate list so they are not put together with the remaining orders waiting to be filled. This is done so it does not happen that we open a trade and close it at a loss of the bid-ask spread during the same candle time frame. The method that does this is the same that creates orders issued by strategies. It simply converts the “good until date” time parameter into a life-time integer indicating for how many candles the order will be available to be filled and gives it an ID.

Then, we start iterating the pending orders and trying to fill them given the current market prices. This is done by the *try_fill* method which tests if the order could have been filled during the latest candle according to its type. As described below, we will assume a previous market order got filled as soon as the current candle opened at the open price. For limit (resp. stop) orders we will assume that, if the current candle opened

inside (resp. beyond) the price set by the order, it will be filled at the open price. If the price set by the order is reached during the candle's time frame, the order is instead filled at the order price. This ensures the limit and stop orders work as in live trading. Indeed, a limit order will only be filled if a better price or the one requested is available, and a stop order will only be filled if the indicated price is reached, being executed at that price or a worse one.
Note: We are assessing whether past orders were executed during the current candle's time frame, at the end of the current candle's time frame, so we are not choosing at which price the order will execute, but merely verifying at what price it did.

Procedure Testing order execution

```
filled $\leftarrow$ **false**
price $\leftarrow$ order.price
units $\leftarrow$ order.units
**if** order.type **is** market **then**
    **if** units $> 0$ **and** current_ask_open $\times$ units $\leq$ balance **then**
        filled $\leftarrow$ **true**
        fill_price $\leftarrow$ current_ask_open
    **else if** units $< 0$ **then**
        filled $\leftarrow$ **true**
        fill_price $\leftarrow$ current_bid_open
    **end if**
**else if** order.type **is** limit **or** take_profit **then**
    **if** (units $> 0$ **and** current_ask_open $\leq$ price **and** current_ask_open $\times$ units $\leq$ balance)
        **or** (units $<$ balance **and** current_bid_open $\geq$ price) **then**
        filled $\leftarrow$ **true**
        fill_price $\leftarrow$ current_ask_open **if** units $> 0$ **else** current_bid_open
    **else if** (units $> 0$ **and** current_ask_low $\leq$ price) **or** (units $< 0$ **and** current_bid_high $\geq$
            price) **then**
        filled $\leftarrow$ **true**
        fill_price $\leftarrow$ price
    **end if**
**else if** order.type **is** stop **or** stop_loss **then**
    **if** (units $>$ 0 **and** current_ask_open $\geq$ price **and** current_ask_open $\times$ units $\leq$ balance)
        **or** (units $< 0$ **and** current_bid_open $\leq$ price) **then**
        filled $\leftarrow$ **true**
        fill_price $\leftarrow$ current_ask_open **if** units $> 0$ **else** current_bid_open
    **else if** (units $> 0$ **and** price $\leq$ current_ask_high) **or** (units $< 0$ **and** current_bid_low $\leq$
            price) **then**
        filled $\leftarrow$ **true**
        fill_price $\leftarrow$ price
    **end if**
**end if**
**if** filled **then**
    apply_order_fill(order, fill_price)
**end if**
**return** filled
```

If any order is filled, then another method, *apply_order_fill*, is called which applies the resulting effects. Specifically, it alters the balance accordingly, creates any stop-loss or take-profit orders if applicable and removes the recently filled order from the pending orders stack. This last action is done based on the ID of the order. Since we are removing the orders from the list while we are iterating it by index, we do not need to increase the index in this case, since the current index will provide the next order if there is any, given that the current order was removed. Boundary orders are identified by their parent's ID, so if the stop-loss (resp. take-profit) is filled, the corresponding take-profit (resp. stop-loss), if it exists, is removed from this list as well. Note that this way we only ever disregard stop-loss (resp. take-profit) orders for which the corresponding take-profit (resp. stop-loss) has already been filled, so there is no unwanted skipping of orders even though we are altering the list while we are iterating it.

As in live runs, after the maximum amount of expected responses is reached, the broker manager liquidates the current positions, if there are any. In the backtesting mode, this is done simply by changing the balance according to the number of units currently held and the latest ask or bid price, whether we are short or long, respectively.

4.3 Indicator Manager

The Indicator manager is responsible for creating and handling the indicators themselves. It is initialized after the broker manager and takes the initial data requested by it as well some items from the configuration file. During its initialization procedure all indicators that are flagged in the configuration file are created for each instrument.

The data manager has two main purposes: to retrieve all the indicator data that is needed by the strategy manager, and to handle indicator updates. This last function is called whenever there is a candle response from the data stream. It is interesting to also detail how are the indicators themselves implemented. Since we do not want having to regularly check which indicator we are dealing with at the moment and have different ways to handling each one, an abstract indicator class was created with a set of common methods that each different type of indicator overrides if necessary. The purpose of these functions is to allow the indicator itself to handle the data update and to estimate its value when the higher-level code requires it, while on the surface we are apparently only dealing with a single seamless type of indicator.

It is also worth noting that there is no way to retrieve past indicator data (no history is kept during run-time), other than looking at what is being written in the logs. This is by design since we want to keep a minimum of memory usage and perform updates in the values quickly, rather than having to recompute the indicator's value from scratch every time we have a new candle. In order to be able to do this, we have to resort to single-pass algorithms to update core statistics for the indicators we use, such as the average or the standard deviation described below, which take as input solely the previous estimation and the new value.

4.3.1 Exponential moving average estimation

The exponential moving average is firstly estimated when the data manager is initialized with the candle history requested. If we represent the degree of weighting decrease configured for this indicator by $w \in]0, 1[$, the initial candle look back as n and $p_1, \dots, p_n$ the prices from the candle history, we estimate the current EMA for the given pair as

$$\widehat{EMA}_n = \frac{p_n + w^1 p_{n-1} + \cdots + w^{n-1} p_1}{1 + w + w^2 + \cdots + w_{n-1}}.$$

This is of course not equal to the actual EMA, since the weighting for an EMA decreases exponentially never actually reaching 0 and thus we would have an infinite number of terms. Since this is impossible to obtain in practice, and given that for n sufficiently large the weights will be negligible, we will drop the estimation notation. After creating a new candle and having a new price to update the indicator with, we want now to compute EMA_{n+1}. If we say that $W_{n+1} = 1 + w^1 + \cdots + w^n$, then

$$\begin{aligned}
EMA_{n+1} &= \frac{p_{n+1} + w^1 p_n + \cdots + w^n p_1}{1 + w + w^2 + \cdots + w_n} \\
&= \frac{1}{W_{n+1}} \left(p_{n+1} + \sum_{i=1}^{n} w^{n+1-i} p_i \right) \\
&= \frac{1}{W_{n+1}} \left(p_{n+1} + w \sum_{i=1}^{n} w^{n-i} p_i \right) \\
&= \frac{1}{W_{n+1}} \left(p_{n+1} + w W_n EMA_n \right) \\
&= \frac{1}{W_{n+1}} \left[p_{n+1} + (W_{n+1} - 1) EMA_n \right] \\
&= \frac{1}{W_{n+1}} p_{n+1} + \left(1 - \frac{1}{W_{n+1}} \right) EMA_n \\
&= EMA_n + \frac{1}{W_{n+1}} \left(p_{n+1} - EMA_n \right)
\end{aligned}$$

However, since $0 < w < 1$, we know that $\sum_{i=0}^{\infty} w^i = (1 - w)^{-1}$. So, if we start with n large enough, we can use the following approximation

$$\frac{1}{W_{n+1}} \approx (1 - w)$$

which simplifies our strategy of obtaining an online method to update the indicator since we do not need to also keep track of the total sum of weights. In fact, every time we get a new price update, the EMA indicator updates its price according to

$$EMA_{current} = EMA_{previous} + w \left(p_{current} - EMA_{previous} \right)$$

thus this is the only computation the indicator does when asked to update its value and it only needs to save the output.

4.3.2 Standard deviation estimation

For the sample standard deviation we have the same issue, we want to keep track of as little information as possible. Firstly, when initializing this statistic, we compute the sample standard deviation from the candle history request. As new prices arrive, we need to update this value but we would prefer to do so without having to recompute the standard deviation of the growing sample every time, and thus having to keep it in memory. Fortunately, back in 1962 B. P. Welford discovered a method to do exactly what we need [42].

Algorithm 6 Welford's method for sample variance

```
Require: stream of values $p_k$, with $k \in \mathbb{N}$
  append $p_k$ to sample
  $n \leftarrow$ sample.size
  if $n = 1$ then
    $M_1 \leftarrow p_1$
    $S_1 \leftarrow 0$
  else
    $M_n \leftarrow M_{n-1} + (p_n - M_{n-1})/n$
    $S_n \leftarrow S_{n-1} + (p_n - M_{n-1})(p_n - M_n)$
    yield $S_n/(n-1)$
  end if
```

We then take the square root of $S_n/(n-1)$ to obtain the standard deviation of the current sample of n candles, while having only to save the current value of the statistic, the current average and the sample size.

4.4 Strategy Manager

Responsible for handling strategies' workflows, providing each strategy with updated data as well as collecting their orders, the strategy manager is initiated after the indicator manager, once we have the indicators created and ready to be used. When initializing the strategy manager, we simply take the strategies dictionary from the configuration file and iterate it to create each strategy accordingly, using the set of information contained in this dictionary that is mapped to the relevant strategy.

Similarly to how the indicators are organized, each strategy is its own class that inherits some functions from an abstract parent class that are overriden in each implementation to suit the specific strategy needs, allowing the higher level code to treat them indistinctly. Collectively, strategies must keep track of whether they are ready to order long or short positions, as well as a list of their pending orders, for each instrument. They must also be able to assess if a new order should be placed given the state of the market and its own algorithm, and if so, create them since the order's details also depend on the strategy. The orders created by the strategies are in the following format, in compliance with what OANDA's server expects [30]:

Structure 5 (Order request) *Created by each individual strategy and passed on (done by the strategy manager, when looking for strategies that are ready to order) to the broker manager, which in turn relays the request to the server, the order request is organized as a dictionary linking the following values to the respective keys.*

Table 4.3: Items contained in an order request.

Key	**Value**
type	The type of order we want to create (market, limit or stop).
units	The amount of units to be filled in this order. A positive quantity represents a long order and a negative amount will result in a short position.
instrument	The instrument that we intend to trade.
stopLossOnFill	A JSON object specifying the details of a stop-loss order to be placed when the main order is filled.
takeProfitOnFill	A JSON object specifying the details of a take-profit order to be placed when the main order is filled.
price*	The price threshold that will trigger the order. Limit orders will be filled at this price or a better one, while stop orders will be filled at this price or one that is worse.
timeInForce*	The type of time limitation imposed on how long the order should be kept pending before being cancelled. For limit or stop orders (that are not take-profit or stop-loss) we will use "good until date", which will cancel the order at the provided time. Market orders are "filled or killed" by default.
gtdTime*	The time at which an order set with timeInForce as "good until date" will be cancelled.

*The items marked with * are not used for market orders.*

Other than creating the strategies themselves, the strategy manager also has three other important purposes. The first is iterating all strategies upon the creation of a new candle for a given instrument and requesting each one to assess the new price and create any orders if applicable. If indeed it is the case that we should create new orders, the other method in the strategy manager builds a generator that is iterated in the main routine, yielding all pending orders for the given instrument so they can be created by the broker manager.

Chapter 5

Statistical Models in Time-Series Forecasting

Having a software that is ready to test any strategies we might think of and to actively trade them, it is worth to think about how we can get an hedge on the market and time our entries and exits accurately enough to obtain a good performance. Of course, if we knew in advance exactly what would be the next price, it would be trivial to beat the market's performance by a wide margin. That is impossible, but having this in mind, is it possible, having a reasonably good knowledge of the market behavior, to satisfactorily forecast the upcoming price, enough so to outperform a buy-and-hold strategy? This is in essence what the common strategies we saw before try to do: recurring to technical indicators to study and trace the market's behavior so far, and using that knowledge to predict how it is going to act next. The way the market behaves is in itself just a collection of millions of participants looking to get their own interests fulfilled. Then, it is interesting to think that, given any strategy that grows in popularity to be widespread and common enough, traded in sufficiently large amounts to move the market, it might become a self-fulfilling prophecy. Indeed, it makes sense that researchers at Credit Suisse [38] compare technical analysis to and associate it with behavioral finance.

There is, though, at least one other way to gain knowledge on the way the market acts, while accepting the inherent randomness that results from the constant clash of so many different strategies, thoughts and interests of all market players, as well as other external factors and events. This entails creating a mathematical model, with a stochastic component, built to fit a time-series of prices for a given instrument we are interested in trading. According to Paul Wilmott [43], we are now entering a different domain of financial analysis, called quantitative (or mathematical) finance.

The most basic family of models we have at our disposal for this purpose are the autoregressive (AR), the moving average (MA) and their hybrid, the **autoregressive moving average model** (ARMA) [24]. Most of these models require the underlying time-series process to be at least **weakly stationary**.

Note: A time-series process, $\{X_t : t \in T\}$ with $T \subseteq \mathbb{Z}$, is a stochastic process indexed by integers, hereafter referred to simply as time-series. Going forward, only this particular type of stochastic process will be considered, therefore the index set T will always be a subset of $\mathbb{Z}$. We will also call the observation of a time-series process, $\{x_t : t \in T\}$, by time-series.

Definition 12 (ARMA Process) *A process $\{Y_t : t \in T\}$ is said to be an ARMA(p,q) process if $\{Y_t\}$ is weakly stationary and*

$$Y_t = \mu + \varphi_1 Y_{t-1} + \cdots + \varphi_p Y_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}, \ \forall t \in T.$$

where μ is a constant, $\varphi_1, \ldots, \varphi_p, \theta_1, \ldots, \theta_q$ are parameters of the model and $\varepsilon_{t-q}, \ldots, \varepsilon_t \sim i.i.d.(0, \sigma^2)$ [1]. The general expression for the underlying AR (without trend) and MA processes can be obtained from this one by setting p and q equal to 0 *[11], respectively:*

$$ARMA(p,0) \equiv AR(p), \quad ARMA(0,q) \equiv MA(q).$$

Definition 13 (Weak Stationarity) *A time-series $\{X_t : t \in \mathbb{Z}\}$ is said to be weakly (wide-sense, second order or covariance) stationary if:*

- *the first order moment of x_t does not depend on time, i.e. $E[X_t] = \mu$, $\forall t \in \mathbb{Z}$;*
- *the second order moment of x_t is finite at all times, i.e. $E[X_t^2] < \infty$, $\forall t \in \mathbb{Z}$;*
- *the autocovariance of the process does not depend on time, i.e. $\forall t_1, t_2, k \in \mathbb{Z}$, $Cov(X_{t_1}, X_{t_2}) = Cov(X_{t_1+k}, X_{t_2+k})$. [1]*

There is also the strong (or strict-sense) stationarity, which assumes that the joint distribution of $(X_{t_1}, X_{t_2} \ldots, X_{t_s})$ is the same as the joint distribution of $(X_{t_1+k}, X_{t_2+k} \ldots, X_{t_s+k})$ $\forall t_1, t_2, \ldots, t_s, k \in \mathbb{Z}$ [1]. However, according to [13], this is rarely required in practical applications since most weakly stationary time-series in the real world will also be strictly stationary, so we will only refer to stationarity in the wide-sense.

5.1 Non-Stationarity

There are some different types of non-stationarity, and we will list below some examples that illustrate the ones that might be more relevant to our problem [10]:

- **Random walk**, $Y_t = Y_{t-1} + \varepsilon_t$, a discrete counterpart to the Brownian motion [3], where $\varepsilon_t \sim i.i.d.(0, \sigma^2)$ is white noise, has $Cov(Y_t, Y_{t+k}) = Cov(Y_t, Y_t + \sum_{i=1}^{k} \varepsilon_{t+i}) = Cov(Y_t, Y_t) + Cov(Y_t, \sum_{i=1}^{k} \varepsilon_{t+i})) = Var(Y_t) = Var(Y_0) + \sum_{i=1}^{t} Var(\varepsilon_i) = Var(Y_0) + t\sigma^2$, therefore the covariance is time dependent;

- **Random walk with drift**, $Y_t = \alpha + Y_{t-1} + \varepsilon_t$, a slight modification that adds a drift component on every iteration, having $E[Y_t] = \alpha t + E[Y_0]$, *i.e.* the mean dependent on time;

- **Random walk with time trend**, $Y_t = \beta t + Y_{t-1} + \varepsilon_t$, a variant that considers a drift that scales alongside time, has $E[Y_t] = \sum_{i=1}^{t} \beta i + E[Y_0] = \beta t(t+1)/2 + E[Y_0]$, a mean also depending on time;

- **Random walk with drift and time trend**, $Y_t = \alpha + \beta t + Y_{t-1} + \varepsilon_t$, assumes there is a presence of both components above, which has of course a time dependent mean as well: $E[Y_t] = \alpha t + \sum_{i=1}^{t} \beta i + E[Y_0] = \alpha t + \beta t(t+1)/2 + E[Y_0]$.

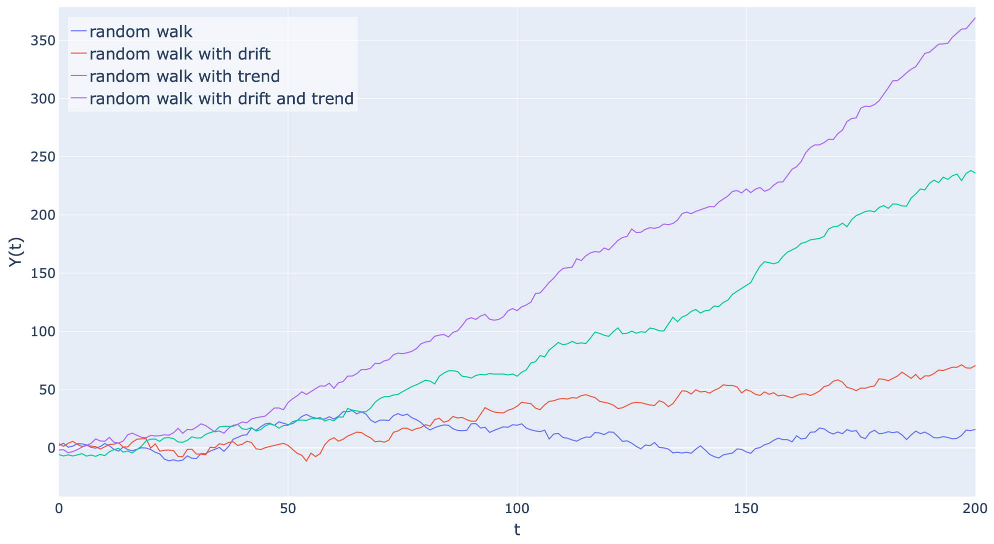


Figure 5.1: 4 random walks of the different configurations above, each one having its own sample of white noise where $\varepsilon_t \sim N(0, 2.5^2), \forall t \in \{0, 200\}$, with drift $\alpha = 0.5$ and trend $\beta = 0.01$.

For a class of non-stationary processes, ones that have stationary increments, non-stationarity is caused by the presence of a unit root, *i.e.* the process's characteristic equation having 1 as a root [24]. A time-series being unit root non-stationary is equivalently defined as the process $Y_t - Y_{t-1}$ being weakly stationary. We can generalize this definition for unit roots of multiplicity d, where it is said that the series is integrated of order d if $(1-L)^d Y_t$ is a stationary process, where L is the lag operator, *i.e.* $L^k Y_t = Y_{t-k}, \forall k \in \mathbb{Z}$.

We can see for example that the presence of a unit root in particular zero-mean first order AR processes causes the process to not validate stationarity in the mean. This process is defined by $Y_t = c + \varphi Y_{t-1} + \varepsilon_t$, where c is the drift constant, φ is a parameter of the model and $\varepsilon_t \sim i.i.d.(0, \sigma^2)$. The respective characteristic equation is $r - \varphi = 0$ therefore φ is the root. If this process has the parameter φ equal to 1, then it has an unitary root (in fact, this particular first-order AR process is a random walk with drift). In this case, by repeatedly substituting Y_h by $c + Y_{h-1} + \epsilon_h$ for every $h \in \{1, \ldots, t-1\}$, we can verify that

$$E[Y_t] = c + E[Y_{t-1}] = \sum_{i=1}^{t} c + E[Y_0] = t\,c + E[Y_0]$$

which means that the mean is affected by time. We can also verify that having a unit root causes stochastic shocks to have long-lasting effects in the variance:

$$Var(Y_t) = Var(Y_{t-1}) + \sigma^2 = Var(Y_0) + \sum_{i=1}^{t} \sigma^2 = Var(Y_0) + t\,\sigma^2$$

causing the variance to diverge to infinity as t increases in the same manner. This is because having $\varphi = 1$ (or $|\varphi| \geq 1$ for that matter) propagates the initial bias or stochastic shocks rather than nullifying them as t increases.

5.2 Checking for Stationarity

If we want to model the price time-series we must then verify if they are stationary, and we will do this by checking if there is a presence of an unit root. There are many tests that allow us to confirm this and one of the most widely referred is the augmented Dickey-Fuller test [11, 13, 24]. This test firstly fits an autoregressive model of order p, abbreviated $AR(p)$, to our data, which has the following form

$$Y_t = \alpha + \beta t + \varphi_1 Y_{t-1} + \varphi_2 Y_{t-2} + \cdots + \varphi_{p-1} Y_{t-(p-1)} + \varepsilon_t,$$

where α is the model's drift, β is the model's time trend coefficient and $\varphi_1, \ldots, \varphi_{p-1}$ are the model's parameters. Using the difference operator Δ (*i.e.* $\Delta Y_t = Y_t - Y_{t-1}\ \forall\, t \in T$) and by adding and subtracting $p-2$ terms of the form $\left(\sum_{i=j}^{p-2} \varphi_{i+1}\right) Y_{t-j}$, $\forall j \in \{1, \ldots, p-2\}$, this expression can be rewritten in the form it is commonly seen in the literature

$$\begin{aligned}
Y_t &= \alpha + \beta t + \varphi_1 Y_{t-1} + \left[\left(\sum_{i=1}^{p-2} \varphi_{i+1}\right) Y_{t-1} - \left(\sum_{i=1}^{p-2} \varphi_{i+1}\right) Y_{t-1}\right] + \varphi_2 Y_{t-2} \\
&\quad + \left[\left(\sum_{i=2}^{p-2} \varphi_{i+1}\right) Y_{t-2} - \left(\sum_{i=2}^{p-2} \varphi_{i+1}\right) Y_{t-2}\right] + \varphi_3 Y_{t-3} \\
&\quad + \cdots + (\varphi_{p-1} Y_{t-(p-2)} - \varphi_{p-1} Y_{t-(p-2)}) + \varphi_{p-1} Y_{t-(p-1)} + \varepsilon_t \\
&= \alpha + \beta t + \left(\varphi_1 + \sum_{i=1}^{p-2} \varphi_{i+1}\right) Y_{t-1} - \left(\sum_{i=1}^{p-2} \varphi_{i+1}\right) Y_{t-1} + \left(\varphi_2 + \sum_{i=2}^{p-2} \varphi_{i+1}\right) Y_{t-2} \\
&\quad - \left(\sum_{i=2}^{p-2} \varphi_{i+1}\right) Y_{t-2} + \left(\varphi_3 + \sum_{i=3}^{p-2} \varphi_{i+1}\right) Y_{t-3} \\
&\quad + \cdots - \varphi_{p-1} Y_{t-(p-2)} + \varphi_{p-1} Y_{t-(p-1)} + \varepsilon_t \\
&= \alpha + \beta t + \left(\sum_{i=1}^{p-1} \varphi_i\right) Y_{t-1} + \left(\sum_{i=2}^{p-1} \varphi_i\right) (-Y_{t-1} + Y_{t-2}) \\
&\quad + \cdots + \varphi_{p-1} \left(-Y_{t-(p-1)} + Y_{t-(p-2)}\right) + \varepsilon_t \\
&= \alpha + \beta t + \left(\sum_{i=1}^{p-1} \varphi_i\right) Y_{t-1} - \left(\sum_{i=2}^{p-1} \varphi_i\right) \Delta Y_{t-1} - \cdots - \varphi_{p-1} \Delta Y_{t-(p-1)} + \varepsilon_t \\
\Leftrightarrow \Delta Y_t &= \alpha + \beta t + (\varphi - 1) Y_{t-1} + \delta_1 \Delta Y_{t-1} + \cdots + \delta_{p-1} \Delta Y_{t-(p-1)} + \varepsilon_t
\end{aligned}$$

where $\varphi = \sum_{i=1}^{p-1} \varphi_i$ and $\delta_j = -\sum_{i=j}^{p-1} \varphi_i$, $\forall\, j \in \{2, \dots, p-1\}$.

It then constructs the test statistic, which is a standard t-statistic, where the parameter we are trying to estimate is φ [24]. This is because the characteristic equation of the AR(p) process has an unit root if and only if $\varphi = 1$. Therefore, we will have this as the null hypothesis ($H_0 : \varphi = 1$), while the alternate hypothesis is that there is stationarity ($H_1 : |\varphi| < 1$). We can then define the test statistic as

$$ADF_t = \frac{\hat{\varphi} - 1}{\text{std. error}(\hat{\varphi})}.$$

The augmented Dickey-Fuller test has some variants depending on whether we want to test if α or β are not null, using additional F-statistics to test joint hypothesis (*e.g.* $\alpha = \varphi - 1 = 0$) [11], so it makes sense to first assess if our data contains any non-zero drift or time trend. Having this in mind, if we plot our whole dataset for EUR/USD, we can say with confidence that at least one of these variables must be not null, as clearly the data is not centered around 0 and there is an apparent trend.

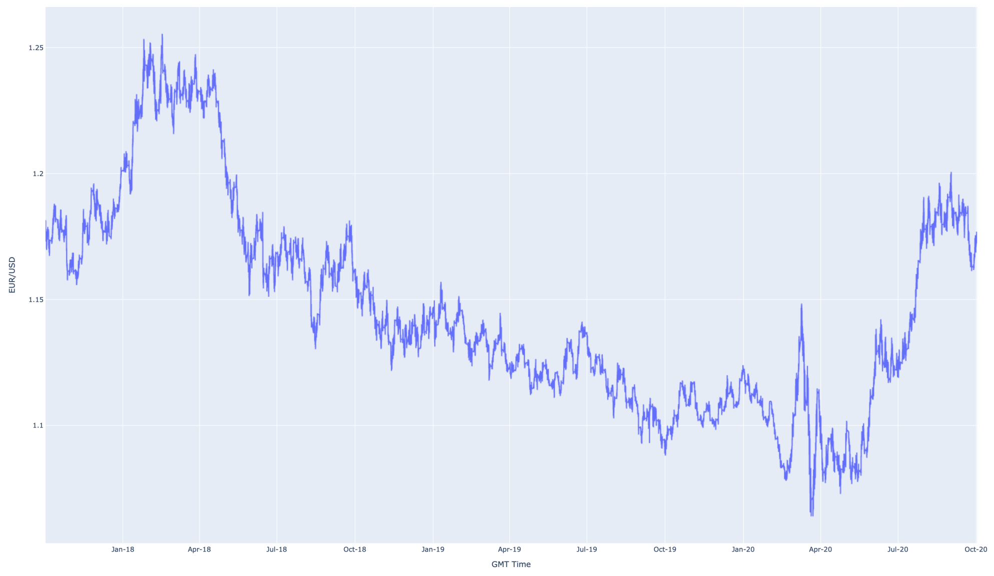


Figure 5.2: The open price of 1-minute ask candles for EUR/USD, from October 30^{th} 2017 to October 30^{th} 2020

We will therefore run the version of the augmented Dickey-Fuller test which does not apply restrictions on the drift or the time trend, but by looking at the plot we can already say that this series does not seem to be stationary as it quite resembles a random walk with drift and/or trend.

The *statsmodels* package for Python has an implementation for this test, the *adfuller* function, which we will use with the configuration for both drift and trend, and also letting the number of lags to use in the autoregressive model to be determined by the Akaike information criterion. Due to a memory limitation, we are not able to run the test for the full data set (1.1 million data points) so it was performed for the first 900.000 points of EUR/USD open prices of 1-minute ask candles instead. The results were the following:

Test Statistic	-2.871343
p-value	0.171975
Number of Lags Used	117
Number of Observations Used	899882
Critical Value (1%)	-3.958780
Critical Value (5%)	-3.410495
Critical Value (10%)	-3.127053

The high p-value leads us to not reject the null hypothesis, thus we can conclude that there is an unitary root present in the model and therefore the series is not stationary. This means that a stationary model is not the right fit for our data.

5.3 ARIMA Models

Even though we are working with a non-stationary time-series, there are ways to still construct a statistical model for it through the means of applying mathematical transformations on our data in order to obtain a stationary series which we can easily model [24]. We then reverse model's predictions by overturning the transformations previously applied and obtain a prediction for the original data. One approach that is quite common, which we have seen in the generalized definition of unit root non-stationarity, is by successively differencing the original data until we obtain a stationary series. This type of transformation is especially effective for non-stationary processes with stationary increments. Indeed, if we differentiate the same data set once and plot the results, we verify that it now appears to be a stationary time-series.

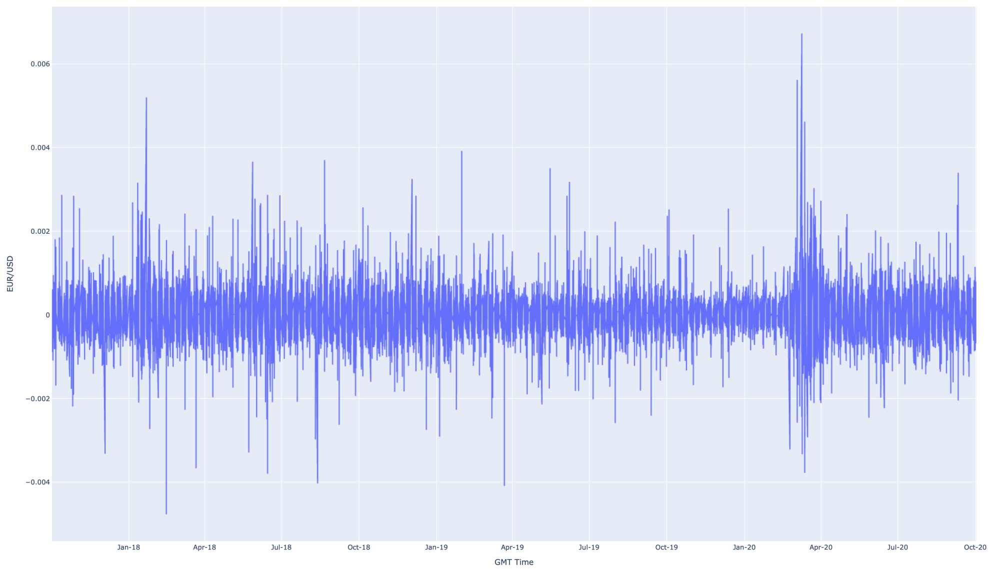


Figure 5.3: The increments of open price of 1-minute ask candles for EUR/USD, from October 30^{th} 2017 to October 30^{th} 2020

It looks like this series has neither a trend nor a drift, so the most appropriate version of the augmented Dickey-Fuller test is the one that assumes $\alpha = \beta = 0$. Without any other change to the settings of the test, performing it on the differentiated data yields the following results:

Test Statistic	-88.713398
p-value	0.000000
Number of Lags Used	117
Number of Observations Used	899881
Critical Value (1%)	-2.565742
Critical Value (5%)	-1.941000
Critical Value (10%)	-1.616820

Which heavily suggests that we should reject the null hypothesis of the presence of an unit root, therefore we can say that there is strong statistical evidence that the first order difference of our original time-series is stationary. This means that we can use one of the models mentioned below that require a stationary process for the increments in our data and then add the predicted increments to obtain predictions to the original price series. There is a specific model that generalizes the ARMA for non-stationary series that after a finite number d of differentiations are stationary (*i.e.* series integrated of order d). Since our data set is integrated of order 1, this model, named **autoregressive integrated moving average** (ARIMA), is the one we will apply in our case.

Definition 14 (ARIMA Process) *A process $\{Y_t : t \in T\}$ is said to be an ARIMA(p,d,q) process if $(1-L)^d Y_t$ is an ARMA(p,q) process. The full expression for an ARIMA(p,d,q) model is*

$$(1-L)^d Y_t = \mu + \varphi_1 (1-L)^d Y_{t-1} + \cdots + \varphi_p (1-L)^d Y_{t-p} + \varepsilon_t - \theta_1 \varepsilon_{t-1} - \cdots - \theta_q \varepsilon_{t-q}, \ \forall t \in T$$

where μ is a constant, $\varphi_1, \ldots, \varphi_p, \theta_1, \ldots, \theta_q$ are parameters of the model and $\varepsilon_{t-q}, \ldots, \varepsilon_t \sim$ i.i.d.$(0,\sigma^2)$ [1].

Fitting these models to our data entails estimating the parameters $\varphi_1, \ldots, \varphi_p, \theta_1, \ldots, \theta_q$, μ and σ^2, which can be achieved through maximum likelihood estimation [1]. Given an observation $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, the likelihood function is $L(\boldsymbol{\theta}|\mathbf{x}) = f(x_1, x_2, \ldots, x_n|\boldsymbol{\theta})$, where f is the joint probability density function of the sample and $\boldsymbol{\theta}$ is the vector of parameters we want to estimate. Therefore, the likelihood function is the joint density in function of the parameters $\boldsymbol{\theta}$. We can rewrite the likelihood as

$$f(x_1, x_2, \ldots, x_n|\boldsymbol{\theta}) = f(x_1|\boldsymbol{\theta}) f(x_2|\boldsymbol{\theta}) \ldots f(x_n|\boldsymbol{\theta}) = \prod_{i=1}^{n} f(x_i|\boldsymbol{\theta}).$$

The estimation of the parameters that maximize this function are the ones used to build the model, *i.e.*, the parameters used are

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta} \in \Theta} L(\boldsymbol{\theta}|\mathbf{x}),$$

where Θ is the parameter space. Maximizing $l(\boldsymbol{\theta}|\mathbf{x}) = L(\boldsymbol{\theta}|\mathbf{x})$ is equivalent to maximizing $\log L(\boldsymbol{\theta}|\mathbf{x})$, since the logarithm is an monotonic increasing function. This transformation is used as it makes computing the estimations easier:

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta} \in \Theta} L(\boldsymbol{\theta}|\mathbf{x}) = \arg\max_{\boldsymbol{\theta} \in \Theta} l(\boldsymbol{\theta}|\mathbf{x}) = \log \prod_{i=1}^{n} f(x_i|\boldsymbol{\theta}) = \sum_{i=1}^{n} \log f(x_i|\boldsymbol{\theta}).$$

Taking the derivative of this function in respect to $\boldsymbol{\theta}$ and setting it to zero will yield us the values that maximize the likelihood. Thus, the equations we need to solve are

$$\frac{\partial l(\boldsymbol{\theta}|\mathbf{x})}{\partial \boldsymbol{\theta}} = 0.$$

which have as solution the maximum likelihood estimations for the model parameters that best fit the sample data.

Using the *ARIMA* class in the *statsmodels* package, we can easily fit an ARIMA model to our dataset. Doing so for different orders of autoregression and moving average shows that this type of model is well-suited for our purposes. In particular, an ARIMA$(3,1,3)$ model fit to the first 5 days of our 1-minute EUR/USD data set, produced the following results:

Number of Observations Used	7200
Log Likelihood	54424.211
Akaike Information Criterion	-108834.421
Coefficients p-value	all ≈ 0
Coefficients Std. Error	all $\leq 4.87 \times 10^{-19}$

We will therefore also include this model when benchmarking the neural network model, having the role of a quantitative counterpart, in addition to the regular technical strategies.

CHAPTER 6

Neural Networks

6.1 Introduction

In 1997, Tom Mitchel succinctly defined Machine Learning (ML) as such: "*A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E*" [27].

Giving a bit more detail, machine learning is a field of computer science, while being closely related to mathematics, that focus on the autonomous improvement of programs by providing a large set of examples, with or without a specified correct output, as input to a software and allowing it to learn, or adapt, from the examples it is given [14, 34]. This is a specially interesting tool to use in the study of problems which may not be easily converted into a traditional task-specific program simply because we may not know how the task is done in our brains (e.g. recognizing objects or patterns) or there may be a very large amount of constantly changing weak rules for categorizing a given input (e.g. speech recognition or credit card fraud) [34].

There are three main ways of computationally acquiring knowledge from experience that are tackled by machine learning, differing in how is the input structured, how it is fed to the software and how should the program react to it [14]:

- **Supervised Learning**, In this type of learning, the data is fed in input/target pairs, letting the software know the target it should produce for the given input. We then allow it to modify some internal state so it can obtain a better fit to the target variable, given the example. This type of learning is the one used for regression and classification problems, where the target is a class or numerical label, respectively;

- **Unsupervised Learning**, Contrary to supervised learning, the target output is not provided in the input data, so objective is that the software tries to create a representation of the input, or find relationships or patterns in the data;

- **Reinforcement Learning**, Here the program will have no input data but is instead meant to operate in an environment where it is given tasks to carry out as well as positive or negative feedback depending on how it performs.

In turn, supervised algorithms can be further divided into two types: regression and classification [14]. As said above, regression algorithms are used when the target output is quantitative and classification algorithms have the task of correctly labeling the given input. As such, since our problem involves predicting numerical quotes for currency pairs, regression algorithms are the ones that interest us the most and will be the ones we will consider hereafter.

There are many types of models that implement the generic idea behind regression-based machine learning. Some of the most discussed ones are decision trees, support vector machines and artificial neural networks [14, 15, 34]. The latter will be the subject of this text.

How to model a neural network (NN) is the first issue we will discuss. Since a neural network is a population of **neurons** [34], we will start by defining those.

Definition 15 (Neuron) *A neuron has a set of n inputs x_i, where $i \in \{1,...,n\}$, and one output Y. Each input is associated with a weight w_i that calibrates the impact of the corresponding input in the output according to some formula for Y. In general, we can express Y as*

$$Y(X;W) = f\left(b + \sum_{i=1}^{n} w_i x_i\right)$$

where b is a bias scalar and f is a function [12].

The function that defines Y is called an activation function and it is supposed to represent how each neuron behaves, so naturally, there are many different models for it. Some of the most popular are presented below [12, 16, 34].

- **Linear**, The produced output by each neuron will simply be the product of the inputs and the weights and adding the bias, without any transformation:

$$f : \mathbb{R} \to \mathbb{R}, \quad f(x) = x;$$

- **Binary Threshold**, These emit a fixed spike of activity when the threshold $h \in \mathbb{R}$ is reached, so each neuron is either turned on or off:

$$f : \mathbb{R} \to \{0,1\}, \quad f(x) = \begin{cases} 1 & \text{if} \quad x \geq h \\ 0 & \text{otherwise} \end{cases} \quad ;$$

- **Sigmoid**, The output in this case has the nice properties of being smooth and bounded. The logistic function or the hyperbolic tangent are common examples:

$$f:\mathbb{R}\to]0,1[,\quad f(x)=\frac{1}{1+e^{-x}};\qquad f:\mathbb{R}\to]-1,1[,\quad f(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}};$$

- **Stochastic Binary**, Using the same functions as the sigmoid neurons, they treat the respective outcome as a probability of giving out a spike of activity:

$$f:\mathbb{R}\to\{0,1\},\quad f(x)=X \text{ random variable where } P(X=1)=\frac{1}{1+e^{-x}}$$

Note that the linear neuron by itself is effectively an AR(n) model without the white noise component, where the weights $w_1,\dots,w_n$ are the parameters we try to fit to the data. The motivation behind a neural network is that by joining many of these structures and letting them communicate, or trade information, and influence each other, we might produce a more accurate outcome [34]. Having the building blocks of a neural network described, we can formally define the most simple type of artificial neural network.

Definition 16 (Feedforward Neural Network) *Motivated by [34], we can define a feedforward neural network through a directed graph, $G=(N,E)$, where the nodes $n\in N$ symbolize neurons and the weighted edges $e\in E$, represented by the pair of nodes they link, indicate how the neurons share information. The function $w:E\to\mathbb{R}$ provides the weight for each edge and the function $y:N\to\mathbb{R}$ provides the output of each neuron. Neural networks are usually organized in disjoint subsets of nodes $L_1,\dots L_q\subset N$, called layers, where $L_i=\{n_{i,1},\dots,n_{i,k_i}\}$ and $\bigcup_{i=1}^{q}L_i=N$. The first layer, L_1 is called the input layer, layers $L_2,\dots,L_{q-1}$ are usually called hidden layers and the final layer, L_q, is the output layer. The network is organized in a way where each node from a given layer connects to all the nodes in the following layer, i.e. $(n_{i,j},n_{i+1,j'})\in E\ \forall n_{i,j}\in L_i,\forall n_{i+1,j'}\in L_{i+1},\forall i\in\{1,\dots,q-1\}$. The input of the neurons in the input layer are the samples in the data set $X_{m\times k_1}$ used to train the model, that is, for each row of the input data set, the input to neuron $n_{1,j}$ is $x_j,\forall j\in\{1,\dots,k_1\}$. The input for a neuron $n_{i,j}$ with $i\in\{2,\dots,q\}$ is computed as*

$$\sum_{j'=1}^{k_{i-1}} w[(n_{i-1,j'},n_{i,j})]\times y(n_{i-1,j'})$$

where k_{i-1} is the cardinality of L_{i-1}. The output of the neural network is the output of the output layer. If the target contains p values, then the last layer must have p neurons.

We will hereafter use the term neural network to mean feedforward neural networks, unless stated in contrary. The data input to a neural network is generally in the form of a matrix $X_{m\times n}$, where m is the amount of individual samples and n is the amount of features in the model. Each individual feature can be thought of as a characteristic of the data that we believe to have some influence in the target output [14]. They are homologous to the independent (or explanatory) variables of statistical modeling, while the output of the network corresponds to the dependent variable.

6.2 Proof of Concept

As a proof of concept, we will start by having one single set of inputs, $X_{1\times n}$ (or individual sample), and trying to obtain the desired correct output T respective to this particular set of inputs. For now, we will consider the simple linear neurons, that is, the neurons we will use will produce an output $Y = b + \sum_{i=1}^{n} w_i x_i$. We could start by taking a random guess at the set of weights that correctly describe the relation of our inputs to the target output T. Taking that first set of weights, W_n^0, would yield us a first set of results that could be, as far as we know, equally amazingly accurate or disastrously off. In any case, it would make sense to measure how far is our first output, Y^0, from actually reaching T. When using regression-based models to estimate a value, it makes sense to consider the problem of minimizing the resulting mean squared error (mse) in order to tune the parameters. In this particular case, the error associated with the first set of weights is $E^0 = Y^0 - T$ and thus, the mse is simply

$$(E^0)^2/1 = \left[Y^0(X_{1\times n}; W_n^0) - T\right]^2 = \left(b + \sum_{i=1}^{n} w_i^0 x_i - T\right)^2 = \left(b + X_{1\times n} W_n^0 - T\right)^2.$$

We could use another function to measure the error but as we want to find a minimum for it, it is useful that the function chosen is differentiable and decreases to zero as the error decreases to zero, which is the case with the mean squared error. Now, to find a minimum for the mse, as we want to obtain an output Y as close to the respective target T for any set of inputs we might encounter, we will try to find the direction where the slope of the mse is the steepest, with respect to each weight. Being $k \in \{1,2,...,n\}$,

$$\begin{aligned}\frac{\partial}{\partial w_k^0}\left[Y(X_{1\times n}; W_n) - T\right]^2 &= \frac{\partial}{\partial w_k^0}\left(b + \sum_{i=1}^{n} w_i^0 x_i - T\right)^2 \\ &= 2\left(b + \sum_{i=1}^{n} w_i^0 x_i - T\right) x_k \\ &= 2E^0 x_k\end{aligned}$$

We thus obtain $\nabla mse = \sum_{i=1}^{n} 2E^0 x_i = 2E^0 X_{1\times n}$.

Following the gradient descent iterative method of finding a local minimum, we should correct the weights in the opposite direction of the gradient of the mean squared error by a “small” amount in each iteration [34]. This means that we should try to improve our initial guess by constructing $W_n^1 = W_n^0 - \eta \nabla mse$, where $\eta \in \mathbb{R}^+$ is small enough for the method to converge, and repeating the process until we obtain a satisfactory solution. Thus, we should correct the weights iteratively as such:

$$W_n^{k+1} = W_n^k - \eta \nabla mse\left(W_n^k\right), \quad k \in \mathbb{N}_0.$$

6.2.1 Cashier Problem

Using the logic above, we will try to tackle the problem of solving a simple problem by using a neural network consisting of a single neuron besides the input layer. This problem consists in correctly obtaining the price for each food item given the total price of the meal (consisting of 3 items: fries, hamburgers and beverages) and the portions of each item. If we manage to observe someone ordering food and how much they paid for it, we have a training case. Lets suppose in this example that someone paid 14€ for two orders of fries, three hamburgers and a bottle of water. Here, (lightening the notation) $X = [2,3,1]$ and $T = 14$. Lets start with an initial guess of 1€ for each item, i.e. $W^0 = [1,1,1]^T$. For simplicity, let us assume that the bias b is null. Below follows a python script to apply the method described above.

```python
import numpy as np

W=np.array([1,1,1])
T=14
inputs=np.array([2,3,1])
weights=[W]
outputs=[]
learning_rate=0.01
iterations=1000

def Y(X,W):
    return np.dot(X,W)

def gradMSE(X,W):
    return np.dot(2*(Y(X,W)-T), X)

for i in range(0, iterations):
    #Compute the output
    outputs.append(Y(inputs, weights[i]))
    #Correct the weights opposite to the gradient, by a percentage (learning_rate) of the gradient
    weights.append(weights[i]-learning_rate*gradMSE(inputs, weights[i]))
    print(weights[i])
```

After 1000 iterations, the adjusted weights we obtain are such that

```python
weights[iterations][0] * inputs[0] + weights[iterations][1] * inputs[1] + weights[iterations][2] * inputs[2]
```

```
13.999999999999998
```

which is a somewhat valid answer for the price of each item, since multiplying each inferred price by the respective amounts of each item ordered, yields aproximately 14€ which we know was the total amount paid. The final guess we obtained for the prices was:

$$W^{1000} \approx [2.14285714, 2.71428571, 1.57142857]^T$$

Now lets try with a different initial guess, for instance $W^0 = [2,2,1]^T$. After 1000 iterations, we obtain a similar output ($Y^{1000} = 13.999999999999998$) but a different set of weights:

$$W^{1000} \approx [2.42857143, 2.64285714, 1.21428571]^T.$$

This makes sense since the problem we are trying to solve is obtaining w_1, w_2 and w_3 such that

$$x_1w_1 + x_2w_2 + x_3w_3 = T \Leftrightarrow 2w_1 + 3w_2 + w_3 = 14.$$

Therefore, any solution we obtain has 2 degrees of freedom and without any other types of restrictions, it is possible to find an infinite number of solutions. Thus, we should try to provide at least two other examples so the model is be able to get a consistent solution that does not change based on the initial guess. So, observing two other purchases of 6€

(one pack of chips, one hamburger and one beverage) and 7€ (two hamburgers and one beverage), we get the following set of inputs and respective targets:

$$X_{3\times 3} = \begin{bmatrix} 2 & 3 & 1 \\ 1 & 1 & 1 \\ 0 & 2 & 1 \end{bmatrix}, \quad T_3 = \begin{bmatrix} 14 \\ 6 \\ 7 \end{bmatrix}$$

Now, we have more than one input so the mean squared error is a bit different. If we say that we have an amount m of inputs, then the error associated with input i is $x_{i1}w_1 + x_{i2}w_2 + ... + x_{in}w_n - y_i$, thus the mse is given by

$$\frac{1}{m}\sum_{i=1}^{m}(x_{i1}w_1 + x_{i2}w_2 + ... + x_{in}w_n - y_i)^2 .$$

For each weight w_k, the partial derivative of mse with respect to w_k is

$$\frac{\partial mse}{\partial w_k} = \frac{2}{m}\sum_{i=1}^{m} x_{ik}\left(x_{i1}w_1 + x_{i2}w_2 + ... + x_{in}w_n - y_i\right) = \frac{2}{m}\sum_{i=1}^{m} x_{ik}e_i$$

which is the line k of the matrix $\frac{2}{m}X^T E$. Thus, the gradient of the mse is

$$\nabla mse = \begin{bmatrix} \dfrac{\partial mse}{\partial w_1} \\ \dfrac{\partial mse}{\partial w_2} \\ \vdots \\ \dfrac{\partial mse}{\partial w_m} \end{bmatrix} = \begin{bmatrix} \dfrac{2}{m}\sum_{i=1}^{m} x_{i1}e_i \\ \dfrac{2}{m}\sum_{i=1}^{m} x_{i2}e_i \\ \vdots \\ \dfrac{2}{m}\sum_{i=1}^{m} x_{in}e_i \end{bmatrix} = \frac{2}{m}X^T E.$$

Going back to our example, we are trying to solve the following system

$$X \times W = T \Leftrightarrow \begin{cases} 2w_1 + 3w_2 + w_3 = 14 \\ w_1 + w_2 + w_3 = 6 \\ 2w_2 + w_3 = 7 \end{cases}$$

and since the determinant of X is -3 (non-zero), the system has one and only one solution. Also, it means that X is invertible so we can analytically obtain W:

$$W = X^{-1}T = \frac{1}{3}\begin{bmatrix} 1 & 1 & -2 \\ 1 & -2 & 1 \\ -4 & 4 & 1 \end{bmatrix}\begin{bmatrix} 14 \\ 6 \\ 7 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$

So, the solution we are looking for is $[2, 3, 1]^T$. If we update our script to take two additional examples and take into account the generic form of the gradient of the mean squared error, we have

```
import numpy as np

W=np.array([[1],[1],[1]])
T=np.array([[14],[6],[7]])
inputs=np.array([[2,3,1],[1,1,1],[0,2,1]])
weights=[W]
outputs=[]
learning_rate=0.1
iterations=1000

def Y(X,W):
    return np.dot(X,W)

def gradMSE(X,W):
    inputs_count = np.shape(X)[0]
    return 2/inputs_count*np.dot(np.transpose(X), Y(X,W)-T)

for i in range(0, iterations):
    #Compute the output
    outputs.append(Y(inputs, weights[i]))
    #Correct the weights opposite to the gradient, by a percentage (learning_rate) of the gradient
    weights.append(weights[i]-learning_rate*gradMSE(inputs, weights[i]))
    print(weights[i])
```

Now, after 1000 iterations, we obtain the following set of weights

```
weights[iterations]
```

```
array([[2.],
       [3.],
       [1.]])
```

which is exactly what we obtained by solving the system analytically.

In order to greatly improve the efficiency, readability and the ease of experimenting with different aspects of neural networks, we should start using a proper library for machine learning. In this text, we will use Google's tensorflow as backend [39]. Converting the solution to the cashier problem we had before to use tensorflow, we simply have:

```
import tensorflow as tf
import numpy as np

x = tf.constant([[2, 3, 1],[1, 1, 1],[0, 2, 1]], dtype=tf.float64)
t = tf.constant([14, 6, 7], shape=(3, 1), dtype=tf.float64)

w = tf.Variable(tf.random.normal(shape=(3, 1), dtype=tf.float64), trainable=True)

learning_rate = 0.1
max_iterations = 1000

def output(x, w):
    return tf.matmul(x, w)

def mse():
    return tf.reduce_mean(tf.square(tf.subtract(output(x, w), t)))

error_tolerance = tf.constant([1e-10, 1e-10, 1e-10], shape=(3, 1), dtype=tf.float64)
error = tf.constant([np.inf, np.inf, np.inf], shape=(3, 1), dtype=tf.float64)
i = 0

while (error.numpy() > error_tolerance.numpy()).any() and max_iterations > i:
    i += 1
    tf.optimizers.SGD(learning_rate).minimize(mse, var_list=[w])
    error = tf.abs(t.numpy() - output(x, w))
```

This includes two changes as compared to the last script. Firstly, we are now allowing the starting weights to be random which makes sense since we do not have any other initial information, and we want to solve the problem regardless of our initial guess. The other change is measuring the error along the way and stopping the process if we are satisfactorily close to our solution. The result, as expected, is the same. After 1000 iterations, we obtain

```
w.numpy()
```

```
array([[2.],
       [3.],
       [1.]])
```

Note that with this implementation, we do not have to compute the gradient of the mean squared error function ourselves. Tensorflow does that automatically in its stochastic gradient descent method (SGD) and computes the next set of weights according to the given learning rate[40]. This will allow us to easily experiment different activation functions and depths as computing the resulting *mse* will not be a problem.

6.2.2 Activation Functions

Having the basis of the problem defined and a neural network to solve it, we will start experimenting different ways to build the network and we will start with activation functions. We already stated a few types of possible activation functions: the linear that we have used so far, the binaries which are more fit for classification problems and the sigmoid.

We could try to solve our problem using the sigmoid function or similarly the hyperbolic tangent function which has a similar behavior. The problem is that the range of these type of functions do not allow us to obtain the target value we need. If we were to try to solve the cashier problem with them, we would just obtain the weights that maximize these functions in order for them to output 1, the closest value of any of our targets. As a whole, the best we could hope for would be $mse = [(14-1)^2 + (6-1)^2 + (7-1)^2]/3 \approx 76.67$ which is quite poor. So these types of functions with limited ranges are not the best for producing an output for which we do not want to put any boundaries on.

We will instead try to solve our problem with another class of functions such as the rectified linear unit ($y(x) = \max(0, x)$) or a soft approximation of it, the softplus ($y(x) = \ln(1 + e^x)$) [14]. The range for both these functions is $\mathbb{R}^+$ which fits our needs. To do this, we will simply change the output function defined above to return one of these transformations of $X \times W$.

6.2.3 Adding Depth

The neural network we have built so far is correctly solving the problem we defined but struggling to consistently solve all problems under the assumptions below, within 1000 iterations.

- Initial weights W_3 that contain values sampled from a $N(0,1)$ distribution;
- Integer inputs $X_{3\times3}$ taking values in $\{0,1,2,3\}$ that have a non-zero determinant, in order to obtain a solvable system;
- Integer solutions Y_3 consisting of values in $\{1,2,3\}$.

In order to make this engine more robust and have consistent results we will allow it to have more complexity and freedom in obtaining a solution. For this we will add more neurons organized in a series of layers.

For these intermediary neurons it makes sense to have an activation function that is limited in range (as we only need to produce an output close to the desired target in the final layer) but that still has a continuous domain in $\mathbb{R}$ [14]. As such, we will model each intermediary neuron with an hyperbolic tangent activation function for now. As for the final layer, we will use the rectified linear unit as the activation function in order to have the desired range of $[0, +\infty[$ for the final output. As a first approach, we will build a network with two hidden layers with 50 neurons each. The input will still be a matrix $X_{3\times3}$, with each sample being a vector in $\mathscr{M}_{1\times3}$, but after the input layer it is multiplied by a vector $W^1 \in \mathscr{M}_{3\times50}$. Therefore, the first hidden layer will receive as input a vector $I^1 \in \mathscr{M}_{1\times50}$ for each sample. The following weight structure is a 50×50 matrix, since there are 50 nodes in the first hidden layer and each one connects to the following 50 nodes in the second hidden layer. The output of the second layer is finally multiplied by a weight vector with 50 rows and the final layer has only one weight value since there is only one node in the output layer.

Out of the same 100 problems for both approaches (but with different initial weights as the dimensions are different), neither obtained a solution within the error tolerance defined in 32 cases, within 1000 iterations. Of the remaining problems, both approaches solved them, but a solution was found with the deep network model in fewer iterations than the homologous simple network problem in 42 cases. In other words, out of all problems that were solved by either method, the deep network model solved it in fewer iterations 67.6% of the time. This seems to suggest we should focus on the deep network model, try to apply it to a case more familiar with our topic of interest and further improve it.

6.3 Applying Deep Networks

Now that we have decided the basis of the model we will use for a candle predicting engine, the next step is to try to build one to predict data similar to the one we will use in our algorithmic trading bot. For this we will use the same dataset that is used during backtesting, described in 4.2.2.

In order to simplify building and using a model, we will use from now on the Keras package as a frontend interface while still using tensorflow as backend for the model [20]. The Keras Sequential model allows us to easily construct different types of networks by plugging in various layers, while specifying their output and activation functions, on the fly with a single command.

We should leave a note on the importance of scaling the input data, since most machine learning algorithms do not perform well when the input data has very different scales [14]. One common way of doing this is to standardize the dataset if it follows a normal distribution, that is, we subtract and then divide the dataset by its mean and standard deviation, respectively. It is important to note that the mean and standard deviation should be computed using only the training set since we do not want our model to have any information other than what is in the training data. Another way of obtaining scaled input variables is to normalize them. This is done by subtracting the minimum value from the dataset and then dividing the result by the difference of the maximum and the minimum values. This ensures that the input is within $[0, 1]$. Since our dataset is already in the same scale, has small increments and is nicely bounded, we would not get much benefit from scaling. In fact, using scaled inputs yielded significantly worse results when comparing the loss on the training data.

As a proof of concept, we will try to predict the next candle's open ask price. We fed 3 days worth of 1 minute candle data to a model consisting of one hidden layer of 50 nodes, besides the input layer, outputting a single value using a linear activation function. As for each hidden neuron, the activation function used will now be the reduced linear unit, since testing with the hyperbolic tangent suggested that the model would not adapt well to the volatility in the data. The optimizer used was the stochastic gradient descent and the loss function was the mean-squared error as before and any bias was disregarded as well. The loss obtained was in the order of 10^{-11} on the training data. Below is a plot of the real values during the following 3 days of candles versus our model's prediction.

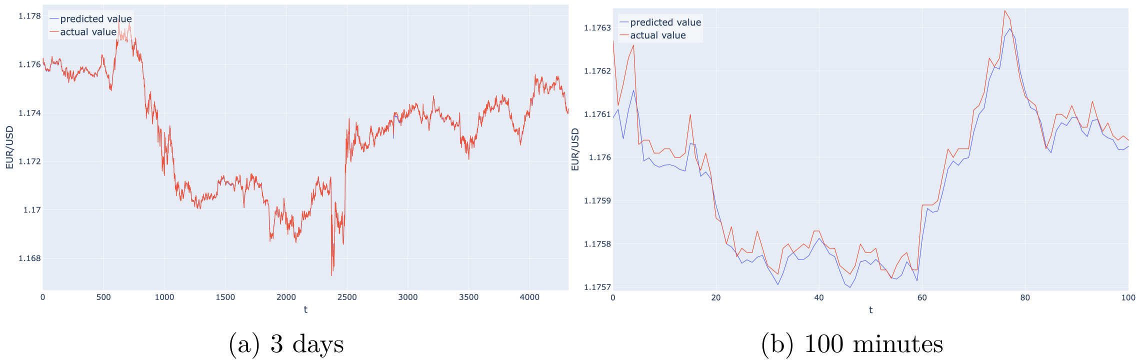


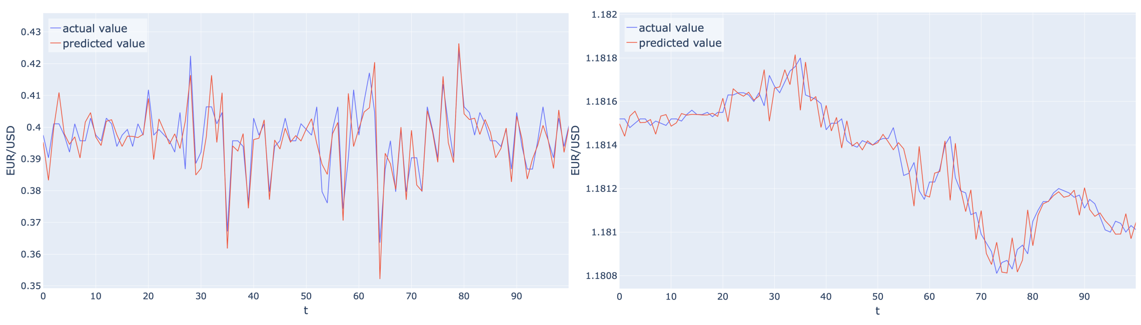
Figure 6.1: Model with two hidden layers vs target on validation data. Prediction of EUR/USD open ask prices for the following 3 days and 100 minutes.

This is apparently a pretty good result, having the predicted values closely tracing the actual open values. Even so, we cannot hope to have found the optimal specification for our neural network by selecting a few arbitrary configuration elements, and it is clear that the result is not perfect. Therefore, our objective now is to test a large number of configurations for the network, comparing them, and obtain a combination of characteristics that will yield us the best performance.

For this purpose, we built a neural network creation tool using Keras, that allows us to easily experiment with, compare and save models. This script makes use of the same neural network class we developed for the algorithmic trading system. It was designed to take many parameters that tune different aspects of a network. We will discuss below some of them and see how they could affect our model.

6.3.1 Increments

Similarly to the mathematical models that require the underlying time-series to be stationary, it might make sense to build the model using a stationary series of price increments instead of the prices themselves. If this option is turned on, the input is transformed in order to obtain the first differences before being fed to the model. In this case, we must normalize the input data after differencing, or else the optimizer will not be able to converge since the data is too fine, in the order of 10^{-4} or smaller. After a prediction is done, the outcome is reversed using the saved minimum and maximum values of the original training set. Using this idea, feeding the model 3 days of differenced data, the prediction outcome on the following 100 minutes is plotted below.



(a) Increments (b) Prices

Figure 6.2: Model with two hidden layers vs target on validation data. Prediction of EUR/USD open ask increments for the following 100 minutes, and the transformation into price predictions.

While it does not show necessarily better results, we will also experiment with this idea.

6.3.2 Hidden Layer Types

There are three types of layers we implemented for the hidden section of our neural network, leading to three different types of models that we will briefly detail. The simplest is the one we have already defined in detail and explored throughout this chapter, it is the feedforward neural network. Here, each neuron in each hidden layer connects to all neurons in the next layer. These are referred to as a *dense* layers.

A different class of neural networks, the convolutional neural network (CNN), is a staple in image recognition [14]. This is usually done recurring to two-dimensional convolutional layers, for the processing of 2D images. For our problem type, we will use the 1D variant, in which the one dimension is time. The main difference to the feedforward network that entices experimenting with them, is that the input to these networks is different from what we have seen until now, taking multiple time steps in the past as opposed to a single one, processed in bulk. To create a CNN, we simply add a 1D convolutional layer as the first hidden layer.

We also considered another class of neural networks that is derived from the feedforward. Similarly to the convolutional network, the recurrent neural network (RNN) takes as input multiple time steps [14]. In this type of network though, the input is iterated step-by-step. In each iteration, the internal state of the neurons in the layers that characterize a RNN are updated in relation to a new time step, as well as the information they created in the previous step, since each neuron also has an edge to itself, besides the usual edges to neurons in the next layer. The type of layer of this class we will use is called long short-term memory (LSTM), which similarly to the CNN, we will add as the first hidden layer of the model when we want to create a RNN.

6.3.3 Loss Functions

The loss function we have used so far has been the mean squared error. Another another loss function that is implemented by Keras for regression problems is the logarithm of the hyperbolic cosine (logcosh) [22]. Given a target output t and a prediction y, the logcosh loss function is defined as

$$\sum_{i=1}^{n} \log[\cosh(y_i - t_i)].$$

This function is interesting to consider since it is less punitive in larger errors, as we can see in the plot below. It might be a good fit in our dataset given the high volatility in our fine-grained data.

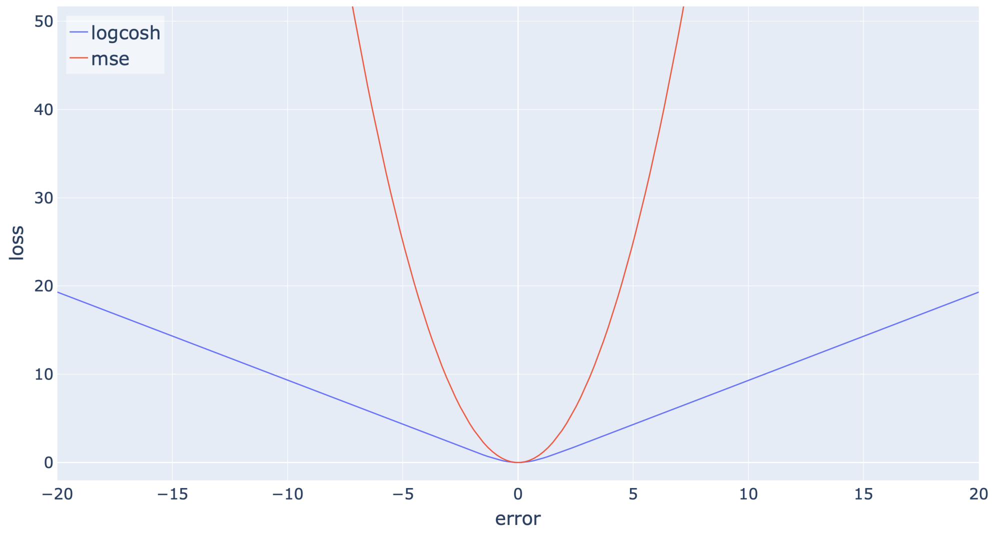


Figure 6.3: The loss value for $x \in [-20, 20]$, for each loss function.

6.3.4 Optimizer Algorithms

Until now, we have always used the gradient descent algorithm to find a local minimum for the loss function. One other optimization algorithm, published in 2015, that has been largely referenced in regard to training neural networks, is the adaptive moment estimation (Adam) [23]. It is described as being similar to the SGD with momentum, with faster performance [14]. While it might not necessarily find better results than the SGD, it may be useful when training large networks with computationally heavy components, like the RNN.

6.3.5 History Sizes

The amount of time steps we provide as individual input samples to the model has been unitary thus far, but as seen in the network types above, this will also be a parameter we want to configure. Not only for the convolutional and recurrent networks, but also for the usual feedforward network. Keras provides a special type of layer that allows the feedforward to take an input with multiple time steps [21]. This layer is called flatten, and it does exactly that: it takes an input with multiple rows and transforms them into features. To use it we need to firstly change the shape of the input layer to allow for additional rows in each individual sample, and then set the following layer to the flatten.

6.3.6 Batch Sizes

The batch size is a parameter that defines whether the network will learn incrementally as it is fed new samples, or only update its internal state when it has processed a given amount of samples [14]. What we have used earlier as default was batch size equal to 1. This is usually called online learning, as the model corrects its weights after each sample. Opposed to this approach, we have what is called batch learning, where the batch size is the dimension of the whole dataset, in which case updates to the model are only performed after going through all the samples. Finally, we can also set the batch size to a value between 1 and the total amount of samples, where updates are performed at the end of each batch. This is known as mini-batch learning. One advantage in using a larger batch size is that the the step taken in the optimization algorithm takes more information, therefore it can possibly find a better direction to minimize the loss. This of course takes much more time for the same amount of dataset iterations, as more samples must be processed before taking each step. We will experiment with different batch sizes, with larger batches making more sense when we intend to train a network, save it and use it later on. Smaller batches however, are more fit for cases where we want to quickly train a model on recent data to use it right away.

We can also configure other items in our implementation of a neural network, that cover topics we have already discussed or are simply settings to make the model better fit our needs. They are:

- **load**, whether we want to load a previous model or create a new one based on these configurations;
- **train_dataset_size**, a setting that is exclusive for the algorithmic trading software. It indicates the amount of past data from the initial request to use if we want to create and train a model during initialization;

- **input_type**, also common to other indicators, it states the price type the network will take as input. We can set it to take both ask and bid candles, only one of them or a specific item from one of the sides (*e.g.* ask open);
- **output_type**, the price type the network will output. Can be set to the same values as above;
- **n_hidden_layers**, the number of hidden layers to add to the model;
- **n_hidden_nodes**, the number of hidden nodes in each hidden layer;
- **activation**, the activation function hidden nodes should use;
- **bias**, the bias to add in each layer;
- **train_epochs** the number of iterations, also called epochs, we will use to train the model. One iteration is a complete loop through the entire dataset;
- **seed**, if we want to use a seed to allow us to reproduce random components.

With this infrastructure set, we can start experimenting with different aspects of a neural network to find which configurations better suit our needs. This will be done resorting to backtesting, and then translated to live trading. In both cases, its predictive capability and performance when applied in a strategy will be benchmarked against the models and strategies we have discussed before.

Chapter 7

Results

In this final chapter we will detail our findings when applying various strategies which we will describe below, also showing how we tuned each one in the backtesting section. While true for all strategies, particularity for the models, we will experiment thoroughly with them and try to find optimal configurations.

7.1 Benchmarking Strategies

Buy (Sell) and Hold

As described in Chapter 3, this simply entails buying the instrument in the beginning of the trading session. We do this so we can measure the performance of the pair. We will also do the same for the sell side: selling at the start of the session and buying in the end. Each side of this strategy will place a single order during its lifetime, which is a simple market order without any other configuration.

Table 7.1: Orders created by the hold strategy, where u is the total amount of units to trade.

Position	**Type**	**Units**	**Price**	**Stop Loss**	**Take Profit**	**Lifetime**
Long	Market	u	—	—	—	—
Short	Market	u	—	—	—	—

Technical Indicators

Exponential Moving Average Retest

Aside from what we have already related, its important to note that the strategy actually contains two indicators. Since we are searching for long and short entries, the test done to the exponential moving average must be considered from both sides, evaluating the ask and the bid, respectively, so we must have an exponential moving average indicator for each one.

The orders this strategy creates are split in two, dividing the total units we intend to trade in two orders with different risk profiles.

Table 7.2: Orders created by the exponential moving average retest strategy, where c denotes the candle item (here being the close), t is the time of the current candle (just now opened) and risk is the difference between the price and the stop loss.

Position	Type	Units	Price	Stop Loss	Take Profit	Lifetime
Long	Limit	$u/2$	$\text{ask}^c_{t-1} + 2$ pips	$\text{ask}^c_{t-2} - 2$ pips	**price** + risk	1 candle
Long	Limit	$u/2$	$\text{ask}^c_{t-1} + 2$ pips	$\text{ask}^c_{t-2} - 2$ pips	**price** +2× risk	1 candle
Short	Limit	$u/2$	$\text{bid}^c_{t-1} - 2$ pips	$\text{bid}^c_{t-2} + 2$ pips	**price** + risk	1 candle
Short	Limit	$u/2$	$\text{bid}^c_{t-1} - 2$ pips	$\text{bid}^c_{t-2} + 2$ pips	**price** +2× risk	1 candle

Moving Average Crossover

The moving average is a simple indicator to flag entry points by smoothing the price series, trying to extract fundamental information by removing the random noise to some degree [6]. The basic idea in a crossover strategy is that when the price crosses the indicator, it is signaling a change in the trend. In sum, we should buy when the moving average is crossed from below and sell when it is crossed from above.

It is usually applied in conjunction with different moving average windows, showing how is the price trend in the shorter and longer terms [8]. The same logic applies, with the order creating a buy signal when the short term moving average crosses the long term moving average from below, and a sell signal if the opposite occurs.

Algorithm 7 Moving average (MA) crossover for long positions

```
while true do
  if previous_ma.short_term $\leq$ $(1 - \text{margin}) \times$ previous_ma.long_term and
  current_ma.short_term $\geq$ $(1 + \text{margin}) \times$ current_ma.long_term then
    create_order()
  end if
end while
```

The margin in the algorithm is there to prevent false triggers, where the moving averages just barely cross each other. This way, only significant crosses signal an entry.

The orders placed by this strategy are simple, like the hold strategy, but here contain boundary orders so that they close automatically when our risk-defined limit for loss or profit is reached.

Table 7.3: Orders created by the moving average crossing strategy, where o denotes the candle item (here being the open).

Position	**Type**	**Units**	**Price**	**Stop Loss**	**Take Profit**	**Lifetime**
Long	Market	u	$\text{ask}_t^o + 2$ pips	$\text{ask}_t^o - 2$ pips	**price**$+$risk	—
Short	Market	u	$\text{bid}_t^o - 2$ pips	$\text{bid}_t^o + 2$ pips	**price**$+2\times$risk	—

Quantitative Indicators

Neural Network Smoothing Crossover

Similarly to the moving average crossover, we will create indicators of the price trend, by allowing the neural network taking in a large amount of past data to perform the next forecast. Just like the moving average counterpart, we will have one indicator that reacts faster to price changes and one more slowly. The entry signals will be created once the fast-reacting indicator crosses the slower one. If, for ask indicators, the faster indicator crosses the slow one from below, it signals a buy. Reversely, for bid indicators, if the fast one crosses the slow from above, a sell signal is emitted.

The order placed by this strategy are identical to the ones placed by the moving average crossover.

Prediction High-Low Crossover

We will use predictions from both ARIMA and neural networks to predict the next candle's high or low and act accordingly. We will also compare the accuracy of neural networks' predictions to the homologous ARIMA predictions. Both models seem to predict fairly well the ask high during downtrends and the bid low during uptrends. However, in the opposite direction, the ask high prices when the price is rising or the bid lows when it is decreasing, are apparently hard to predict. This strategy is based in this idea, meaning that when the prediction error starts to increase, we should trigger a buy or sell signal. Since the prediction has an inherent lag, we can simply construct this strategy to act similarly to a crossover, while incorporating a direction confirmation mechanism as in the EMA retest so it does not trigger too easily. The implementation of this idea is presented underneath.

Algorithm 8 Prediction high-low crossover for long positions

```
while true do
  if flagged then
    if current_candle.ask.high $>$ previous_candle.ask.high then
      create_order()
    end if
    flagged $\leftarrow$ false
  end if
  if previous_prediction $\geq$ previous_candle.ask.high and
  current_prediction $\times$ (1 + error_margin) $\leq$ current_candle.ask.high then
    flagged $\leftarrow$ true
  end if
end while
```

The order placed by this strategy are identical to the ones placed by the moving average and the neural network smoothing crossovers.

7.2 Backtesting Results

The data we will use during backtesting and to train the models was already described in chapter 4. We will use this data structure for the EUR/USD pair, which will be the subject of our study during backtest.

7.2.1 Preliminary Assessment on Neural Networks

Having so many possible settings combinations for neural networks makes it difficult to narrow in the optimal type of network to use and which configuration. One way this could be done would be simply training all possible configurations that we can think of on a small data set, and then compare how their predictions perform out of sample. Unfortunately, there is not enough time for that. As described in the previous chapter, we have around 13 parameters to configure, and each one can take many different values. Even if we limit the number of possible values for each one as to average a total of three per parameter, we would have to test 3^{13} combinations. Limiting the amount of training epochs to 5, given that most configurations converge quite quickly, and averaging 42s for each (training the network and predicting out of sample data), it would still take over 127 years. So we limited heavily what types of configurations we wanted to assess, based on our exploratory testing, and trimmed the number of possible configurations to bulk test, as a first step, down to 5184. This way, we can see results in little over 16 hours. The possible parameters we tested follow below.

Table 7.4: The parameters considered for bulk testing different neural networks' configurations on the left, and the values they can take on the right.

Parameter	**Values**
Batch size	1, 32
Bias	true, false
History block size	1, 3
Increments	true, false
Input type	ask/bid candles, ask candles, same as output
Loss	MSE, logcosh
# Hidden layers	1, 4, 25
# Hidden nodes	10, 50, 250
Hidden layer type	dense, convolutional 1D, long short-term memory
Activation	reduced linear unit (for feedforward NN), tanh (for CNN and RNN)
Optimizer	SGD (for feedforward NN), Adam (for CNN and RNN)
Output type	ask open, ask close
Training epochs	5
Seed	7

Each configuration is comparable since we fixed the seed each time we ran a new one. We trained each model with the same data set (1 day, 1440 1-minute datapoints) and used it to predict the following 1440 1-minute ask open or close prices, depending on the configuration. We then measured the standard deviation (STD), mean absolute error (MAE) and the root mean squared error (RMSE) on the out of sample prediction. The results can be seen in the plot below.

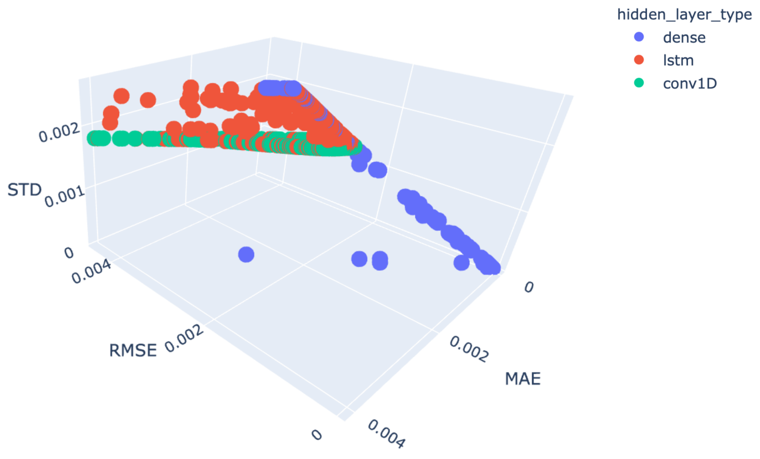


Figure 7.1: The STD, MAE and RMSE for each configuration's set of predictions against the targets, highlighting the different layer types.

It is clear that the dense layers give us the best results over a small set of iterations and limited dataset, and it makes sense that for instance the RNN benefits from taking a greater history size. Indeed, zooming in the under-performing types, we see that better results were obtained with the LSTM layer using a higher history count and amount of hidden layers, which also take longer to train.

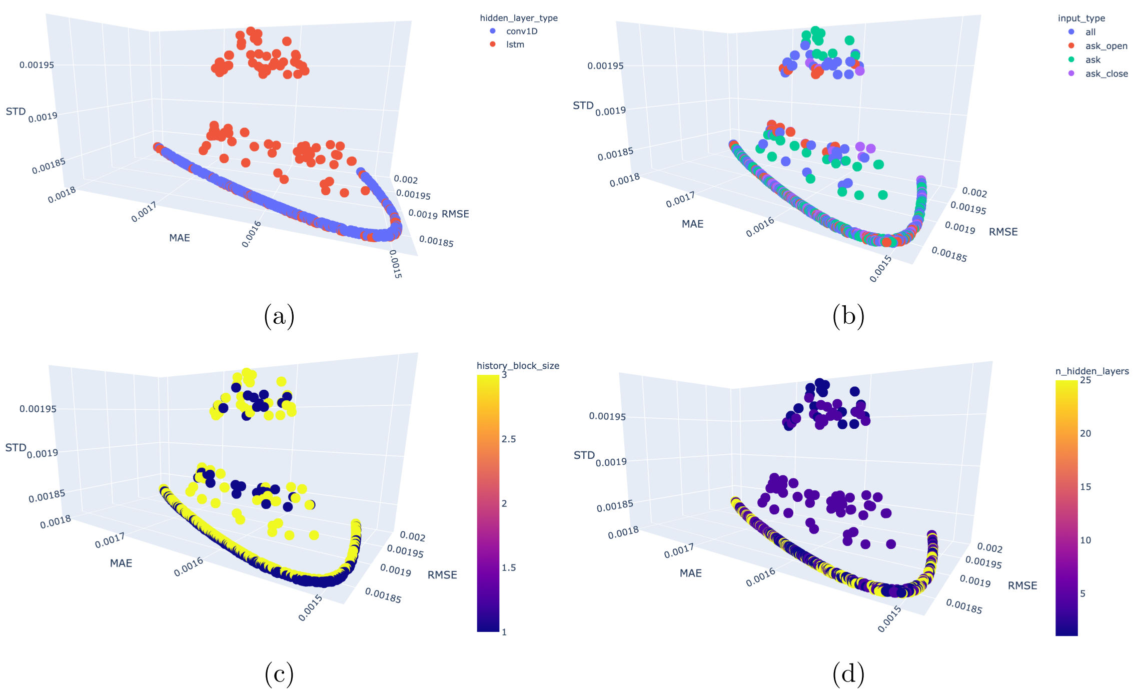


Figure 7.2: The STD, MAE and RMSE for each configuration's set of predictions against the targets, highlighting (a) the hidden layer type, (b) the input type, (c) the history block size and (d) the number of hidden layers.

What we take from this initial test is what kind of configurations minimize these metrics for each network type. For instance, the results when predicting increments were almost always strictly worse than directly predicting the price so we will not consider this option.

Based on this initial assessment, the configurations we will further explore will be the ones in the following table.

Table 7.5: Combinations of parameters we will consider going forward for each type of neural network.

Hid. Layer Type	**Input Type**	**# Hid. Layers**	**# Hid. Nodes**	**Bias**	**History Size**
dense	ask	1	10	False	1
dense	all	4	50	False	1
dense	ask	25	50	False	1
convolutional 1D	all	4	50	False	≥ 3
convolutional 1D	output	1	50	False	≥ 3
convolutional 1D	all	25	50	True	≥ 3
lstm	all	25	10	True	≥ 3
lstm	ask	25	10	False	≥ 3
lstm	ask	25	250	True	≥ 3

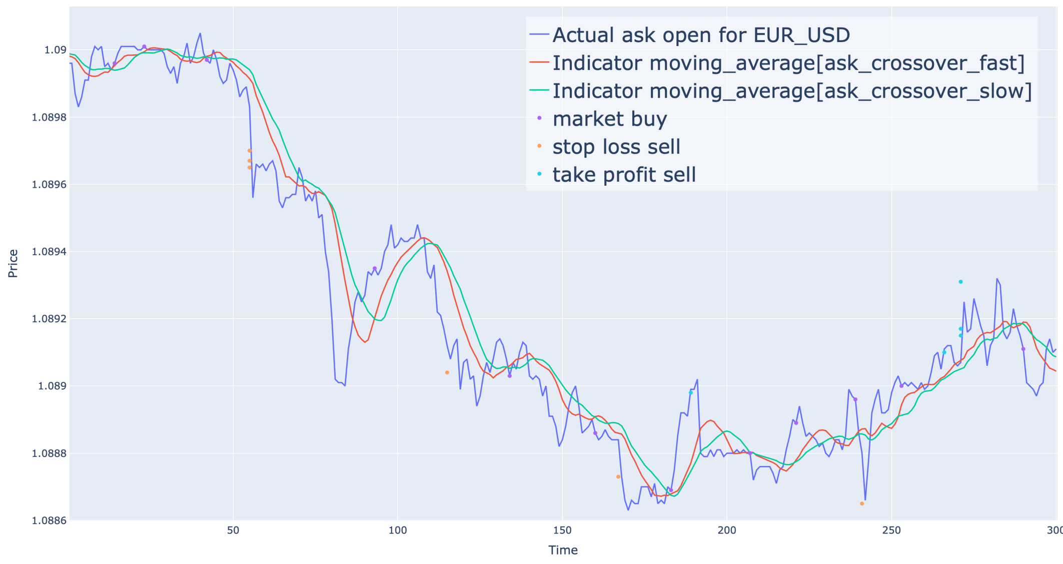
Before creating benchmarks, we must to decide what part of our dataset will be the out-of-sample data to run the backtests in and which will be our "past" data. Since our broker allows for a maximum of 5000 candles of recent past data on data requests, this is what we will use during backtesting, and we will consider the last year of our dataset as out-of-sample. Therefore, the start date of our backtesting period will be October 1st 2019 for the technical indicator-based strategies, and will end on the same day in 2020. For the strategies based on models' predictions, we will test here only the first out-of-sample day.

7.2.2 Benchmark Strategies

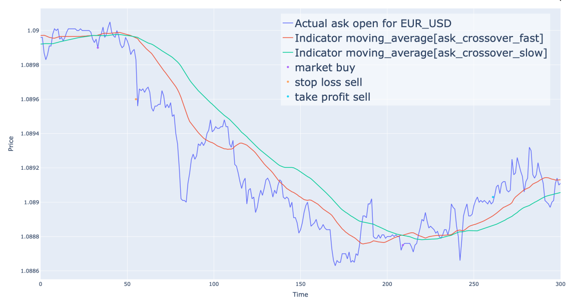
For each benchmark strategy, we considered three possible variations of parameters and compared the results to assess which configuration we would use to live trade. The buy/sell and hold are extremely simple so they do not have any possible configurations, so they will be shown here only as control, showing how the market performed from October 2019 to October 2020. For the remaining ones, we tested three different settings with each. As to lighten the load of graphics in this section, we will display only the graphics for the buy side, but the configurations for the sell side are analogous and the results similar since we have the same approach for both sides. The amount of units bought or sold in each trade are computed as 15% of the total current balance, which is 100.000€ at the start. As control, the buy and hold strategy gained 7.75% during the same period, while the sell and hold lost slightly more than that.

Moving Average Crossover

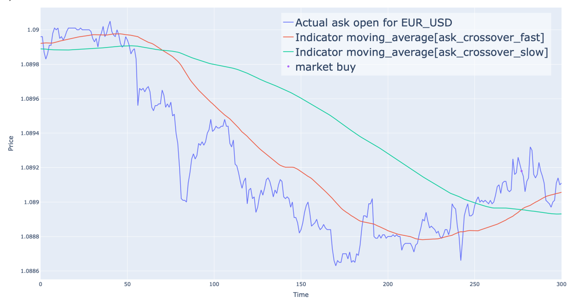
For the moving average crossover, we contemplated three window speeds. One reacts quite fast to the underlying price movements having a short-term MA of 10 candles and a long-term with 15. Another is the opposite, reacting very slowly to price changes, where the short-term MA is 60 minutes and the long-term is 180 minutes. The third one is a middle ground between the first two: has a short-term window of 30 minutes and a long-term of 60. Over the first 300 minutes of the out-of-sample dataset, the results obtained can be seen below for the long strategies.



(a)



(b)



(c)

Figure 7.3: The first 300 ask open prices of the out-of-sample dataset along with the short- and long-term moving averages for (a) fast, (b) medium and (c) slow configurations. The dots are the trades performed by the strategy, being represented in the vertical axis by the respective fill price.

Below, we can see the performance of each configuration, including both the long and short strategies, represented by the 15-day average of the percentage change in balance.

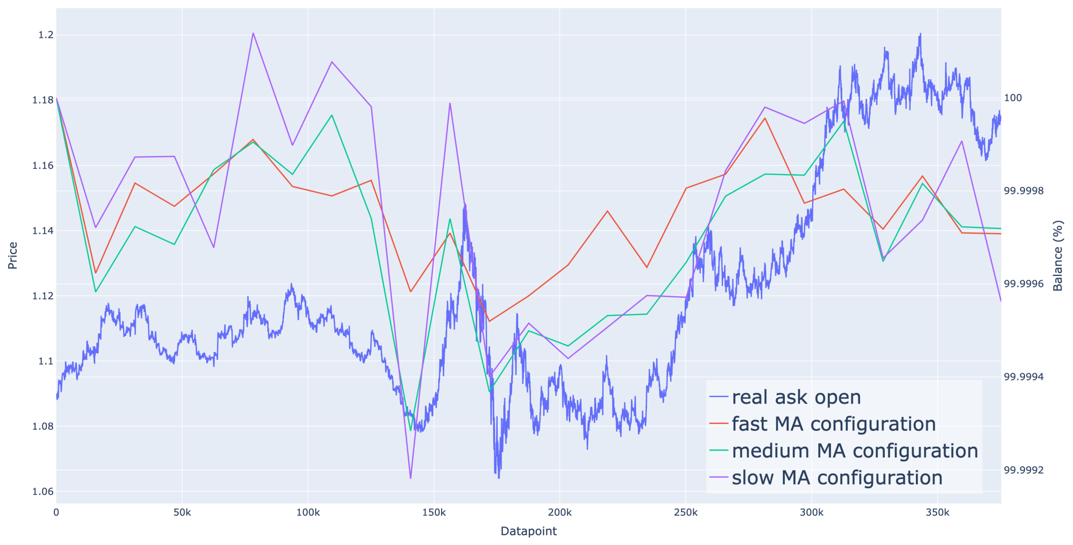


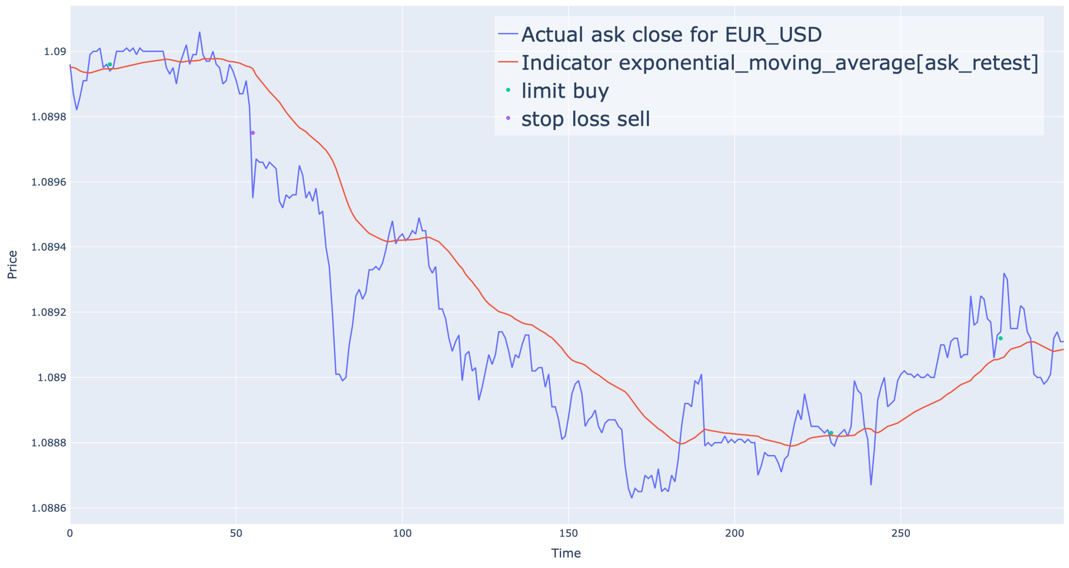
Figure 7.4: The average results of each moving average configuration during the past 15 days, over the performance of the underlying currency pair.

Despite the fast, medium and slow configurations having performed 36, 12.5 and 7.8 thousand trades over the course of the year, respectively, and the size of the trades being quite large, the performance was not impressive on either of them. It is clear that this strategy does not perform better than a strategy that is right 50% of the time. Only the slow configuration was positive at some point, and by a very small margin, but it also showed the greatest losses.

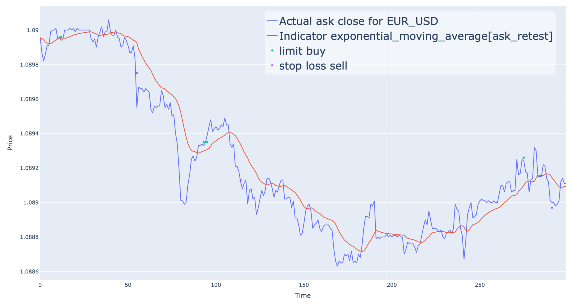
This type of strategy seems to have better results when the market is slowly moving in one direction while performing worse when there are swift up or down movements. Nevertheless, the fast configuration was the one that showed best resilience to market volatility so it is the one we will consider.

Exponential Moving Average Retest

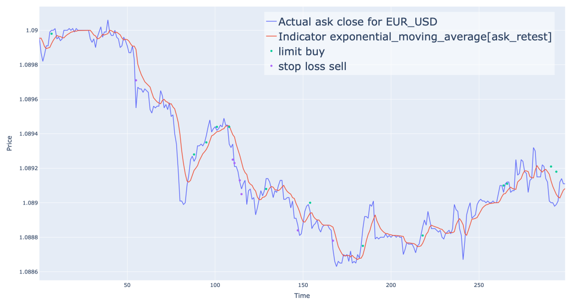
Same as the one before, this strategy has only one parameter that we will explore, which is the rate of exponential decay. We will let this parameter take values of 5%, 10% and 25%, allowing the EMA to be more or less sensible to price movements. Below, we show the first 300 minutes of the out-of-sample dataset alongside the indicator behavior for each decay.



(a)



(b)



(c)

Figure 7.5: The first 300 ask open prices of the out-of-sample dataset along with the exponential moving averages with a decay of (a) 5%, (b) 10% and (c) 25%. The dots are the trades performed by the strategy, being represented in the vertical axis by the respective fill price.

Over the course of the out-of-sample year, the average performance by 15-day period for each configuration, including both long and short strategies, follows bellow.

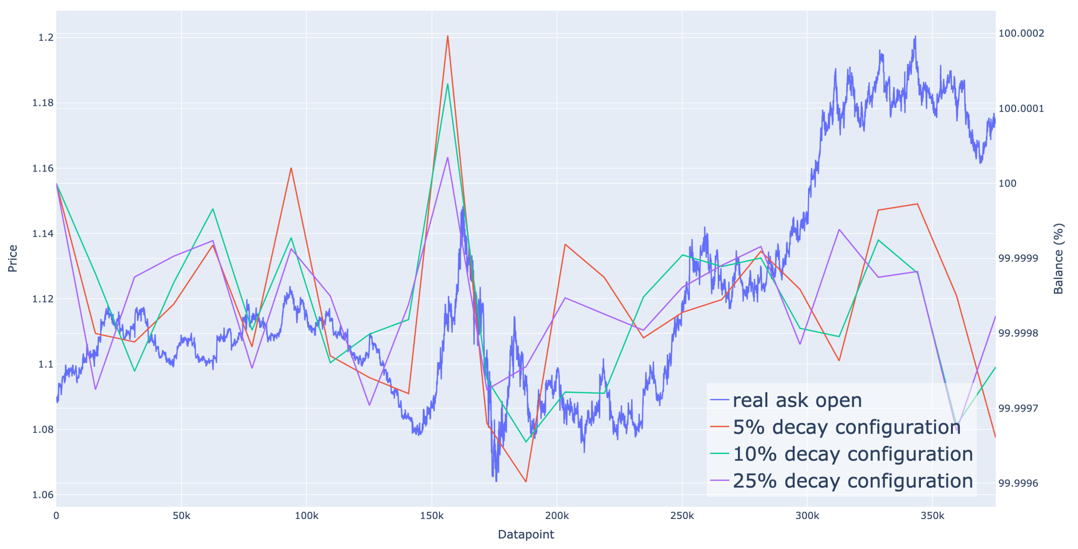


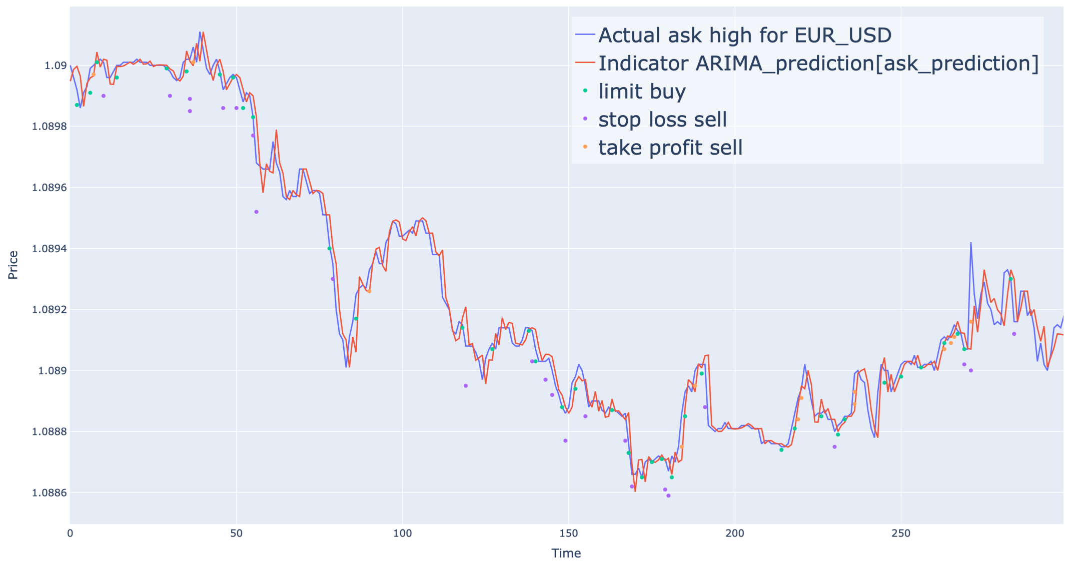
Figure 7.6: The average results of each exponential moving average configuration over the past 15 days, alongside the performance of the underlying currency pair.

During this time frame, the 5% decay configuration performed 42.2k trades, the 10% one did 63.5k and the 25% decay configuration traded 112.5k times. This strategy similarly does not have a good performance, staying below the 100% balance line most of the time. All the configurations seem to have quite similar performance, but the slowest decay of 5% has the advantage of performing the least amount of trades, which inherently makes us lose the bid-ask at best, if we do not turn a profit. Despite this, the configuration of 10% decay achieved similar tops and reasonable lows, while having also a moderately amount of trades, so we will consider it from now on.

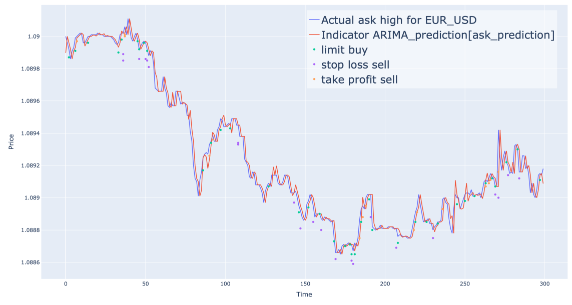
ARIMA High-Low Crossover

The ARIMA high-low crossover requires, as all strategies, having a new indicator value in each assessment. For ARIMA in particular however, this means that we have to fit the model on each new datapoint, since it quickly converges to the mean of the model. Because of this, obtaining a new prediction is very computationally expensive, with a single time step prediction (including fitting the model to the new window of data) takes around 80ms on the lower order models up to 230ms on the higher one. Therefore, we will only perform a backtest on three different orders (3, 5 and 10) and use the same for the autoregressive and the moving average (that is, we will test ARIMA models of orders (3,1,3), (5,1,5) and (10,1,10)), always fitting the last 30 relevant datapoints. Also, the backtest will be limited to the first day of trading (1440 datapoints) in the out-of-sample dataset. We will compare the outcome to the performance of other strategies during the first day as well.

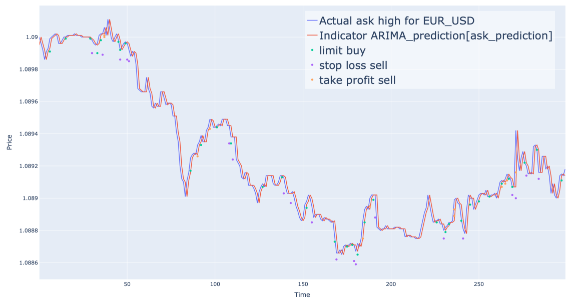
Below we display the behavior of the ARIMA prediction for the next candle's high, and the actual realized value, for the first 300 minutes of out-of-sample data.



(a)



(b)



(c)

Figure 7.7: The first 300 ask high prices of the out-of-sample dataset alongside the prediction for the same time step from the ARIMA model of order (a) (3,1,3), (b) (5,1,5) and (c) (10,1,10). The dots are the trades performed by the strategy, being represented in the vertical axis by the respective fill price.

Running this strategy in the backtesting engine for the course of one day, and plotting the 30-minute average percent change in balance, we obtain the graph below.

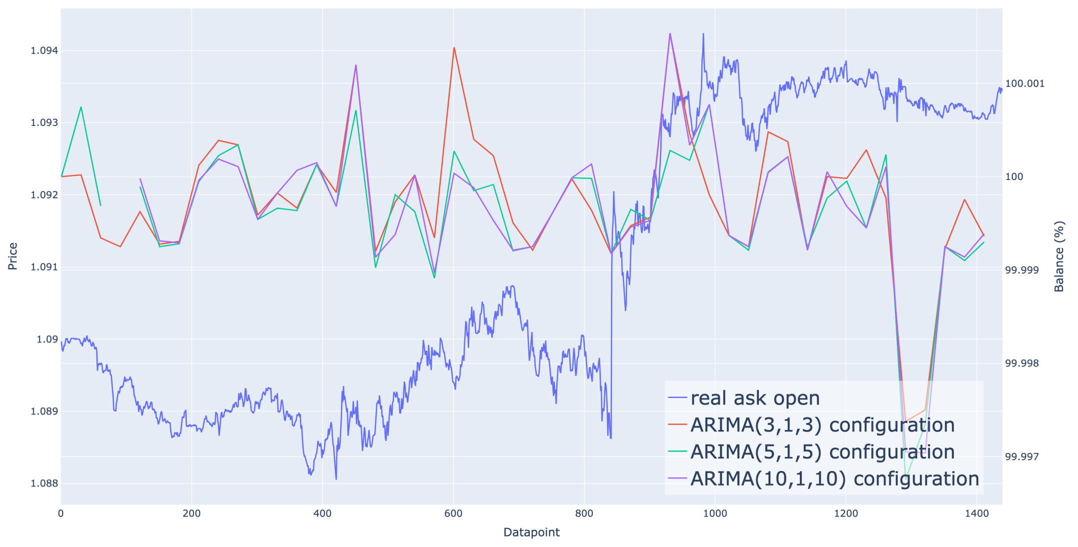


Figure 7.8: The average change in balance for each ARIMA order over the past 30 minutes and the performance of the underlying currency pair.

Note that there are some points missing for the models of higher orders in the plot above. This is because during that 30-minute time frame the strategy placed no trades. Interestingly enough, despite this, during the first day of out-of-sample data, the three models of different orders placed between 356 and 360 trades. This means that the models with a higher order concentrated the trades in certain time frames. Indeed, creating an histogram of the amount of trades closed during each period of 30 minutes, we verify that the ARIMA(3,1,3) distributed better the trades it placed. For this reason, and because it also performed slightly better that its counterparts, we will consider this order for ARIMA.

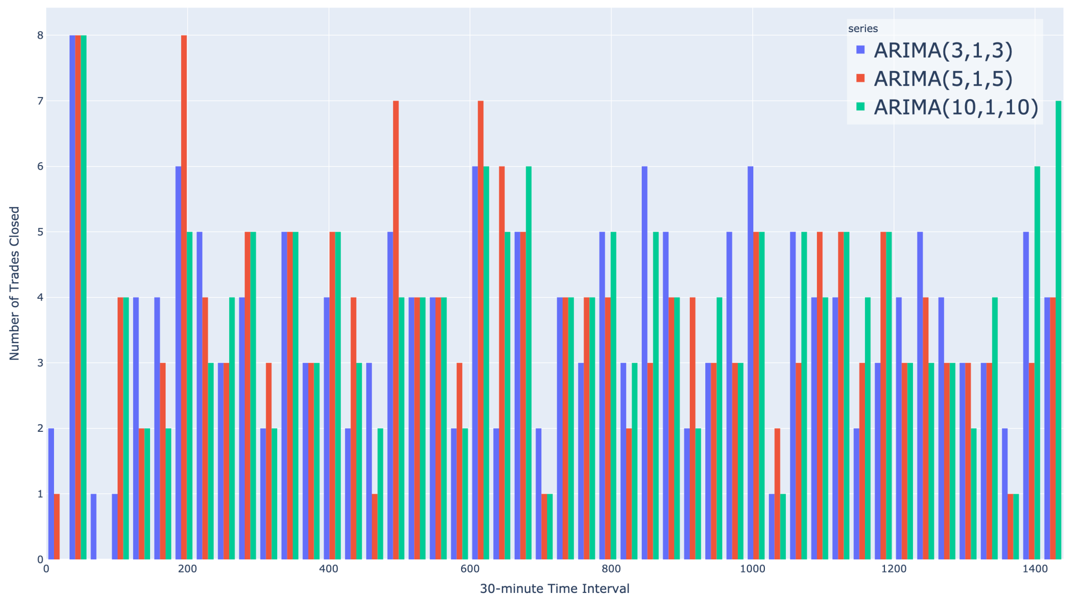


Figure 7.9: The amount of trades closed during each 30-minute time frame, by ARIMA model.

To allow comparing this performance with the other strategies, the table below shows how each one performed during the first day of out-of-sample testing.

Table 7.6: The performance of the technical strategies and the ARIMA-based strategy during the first out-of-sample 1440 minutes.

Strategy	**Configuration**	**Profit or Loss (in %)**	**# Trades Placed**
Buy and Hold	—	+0.318%	2
Sell and Hold	—	-0.322%	2
MA Crossover	fast	+0.0036%	136
MA Crossover	medium	-0.0011%	38
MA Crossover	slow	+0.0044%	22
EMA Retest	5% decay	-0.0004%	188
EMA Retest	10% decay	+0.0002%	284
EMA Retest	25% decay	-0.0005%	420
ARIMA	order 3	-0.0007%	358
ARIMA	order 5	-0.0008%	360
ARIMA	order 10	-0.0007%	356

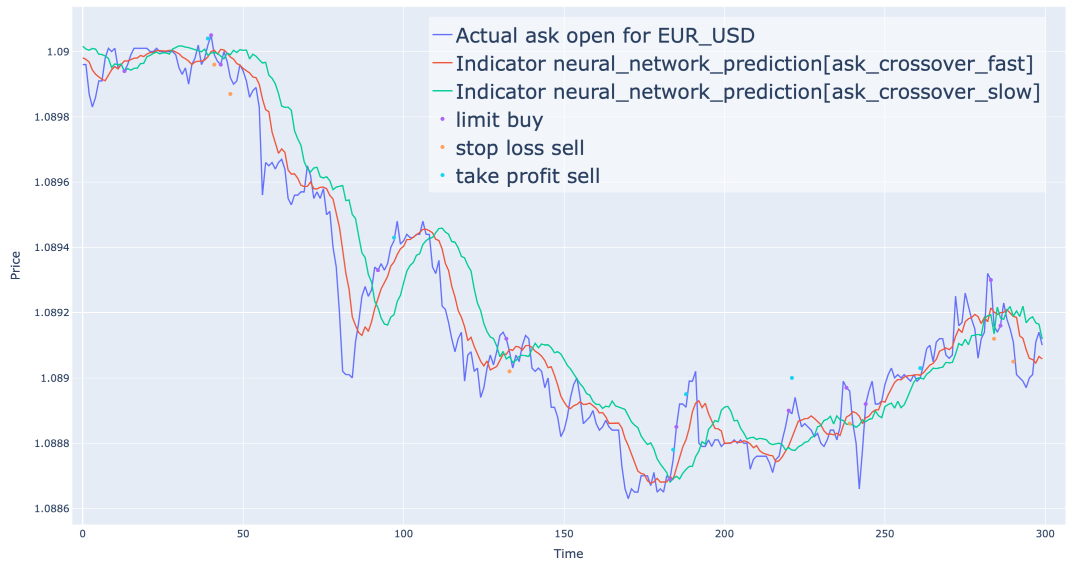
7.2.3 Neural Networks' Strategies

Neural Network Smoothing Crossover

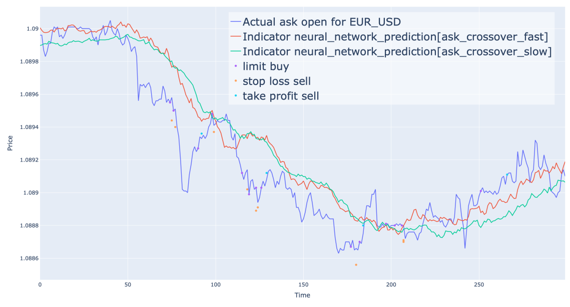
In a similar approach to the moving average crossover, we will attempt to use a neural network to create a smoothing of the data, and use the crossing of the different-period smoothing to signal entries in the market. Out of the configurations mentioned in table 7.6, the one that fitted best the purpose of extracting short-term and longer-term trends from the price values was the one with 4 dense layers with 50 neurons each. Like the MA crossover, we will test fast, medium and slow configurations, with the same window size for each, meaning that the neural networks we will use here will take an history block size equal to the respective MA window in each case.

Maintaining an indicator based on a neural network model is also computationally expensive and it takes some time, at around 25ms, to obtain a prediction. For the same reason as the ARIMA-based strategy, we will make our assessments based on the first day of out-of-sample data.

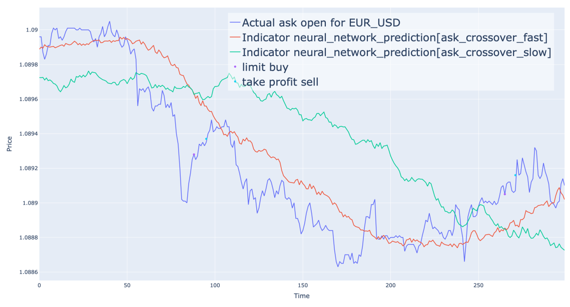
As before, we will first plot the different behavior of each configuration, the trades performed during the first 300 minutes and the relevant actual price.



(a)



(b)



(c)

Figure 7.10: The initial 300 ask open prices of the out-of-sample and the trend predicted by the neural network model with a (a) fast, (b) medium and (c) slow configuration. The dots are the trades performed by the strategy, being represented in the vertical axis by the respective fill price.

Allowing the strategy to run over the first day, we obtain the profitability results below.

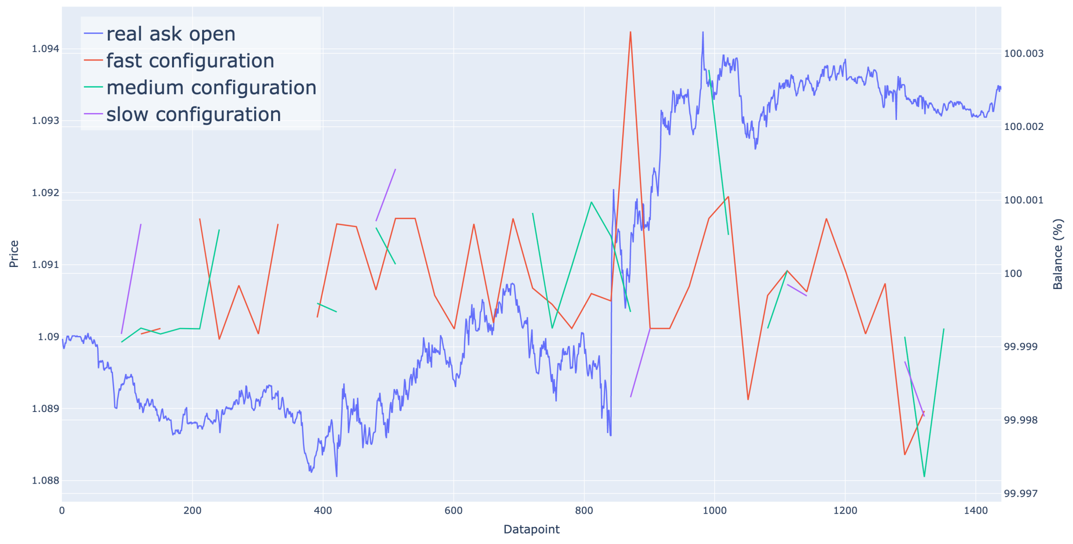
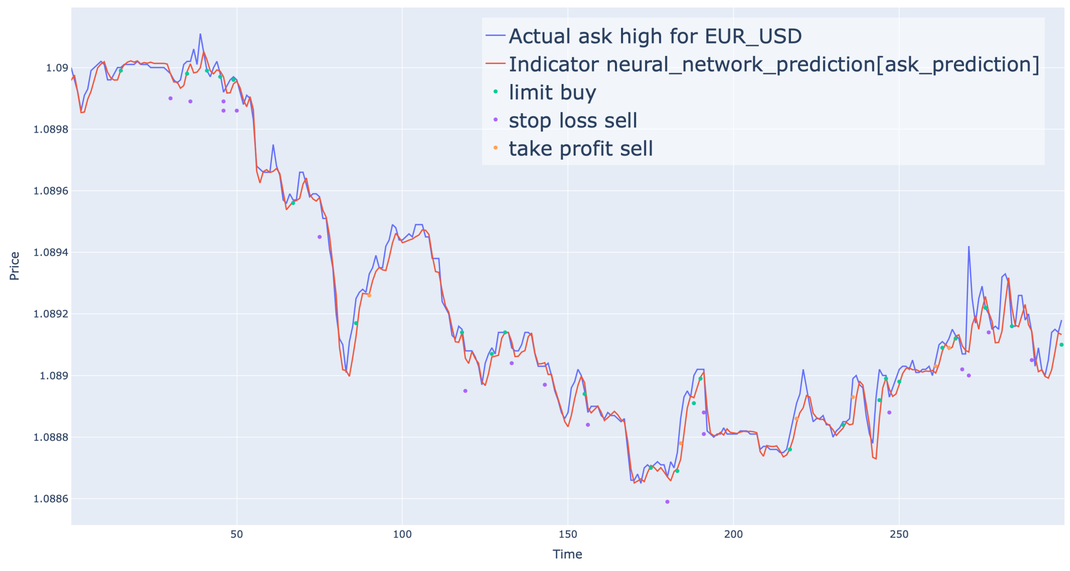


Figure 7.11: The average balance change for each 30-minute time frame, by smoothing configuration and the real open prices of the day-one ask candles.

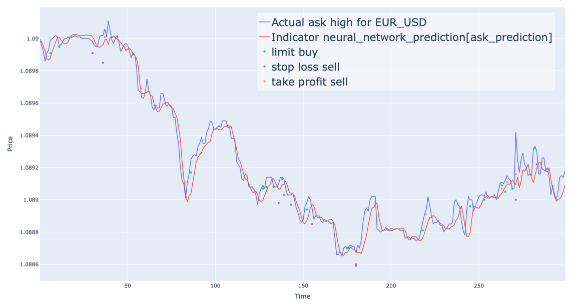
As is clear from the plot above, the two slower configurations are not fit to be used, barely acting on significant price moves. This is observed from the lack of trades closed during most of the 30-minute periods. The fast configuration does not show great performance either since it mostly follows a up and down pattern that hardly relates to the price trend, picking up too much information from small changes. Thus, we should discard it, but we will use it simply to later on compare it to the live performance of the equivalent moving average crossover.

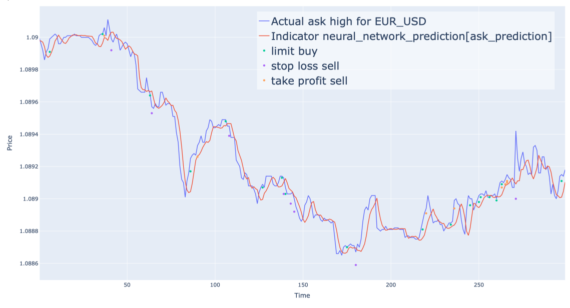
Neural Network High-Low Crossover

Just like the ARIMA prediction-based crossover, we will do the same here but obtaining the high and low predictions from a neural network. Starting from the same network of 4 dense layers with 50 nodes each, which again proved to be the most effective, we will consider three different history block sizes for the network, similar to what we have done in the analogous ARIMA strategy. Here however, we will drop the 10 history size option in favor for a configuration that takes solely one candle as input. This is because this strategy seems to work better when the highs during a down trend (resp. up trend for short positions) are closely predicted, and we verified that the bigger the history size, the worse the high prediction during these trends. This behavior, for each history size, is displayed below.



(a)





(b) (c)

Figure 7.12: The fist 300 ask high prices of the out-of-sample and the respective prediction obtained from a neural network using (a) 1, (b) 3 and (c) 5 past prices. The dots are the trades performed by the strategy, being represented in the vertical axis by the respective fill price.

Indeed, the prediction quality of the highs during a down trend is considerably increased if we use lesser history. Taking these networks and assessing their performance in the first out-of-sample day, we construct the following plot.

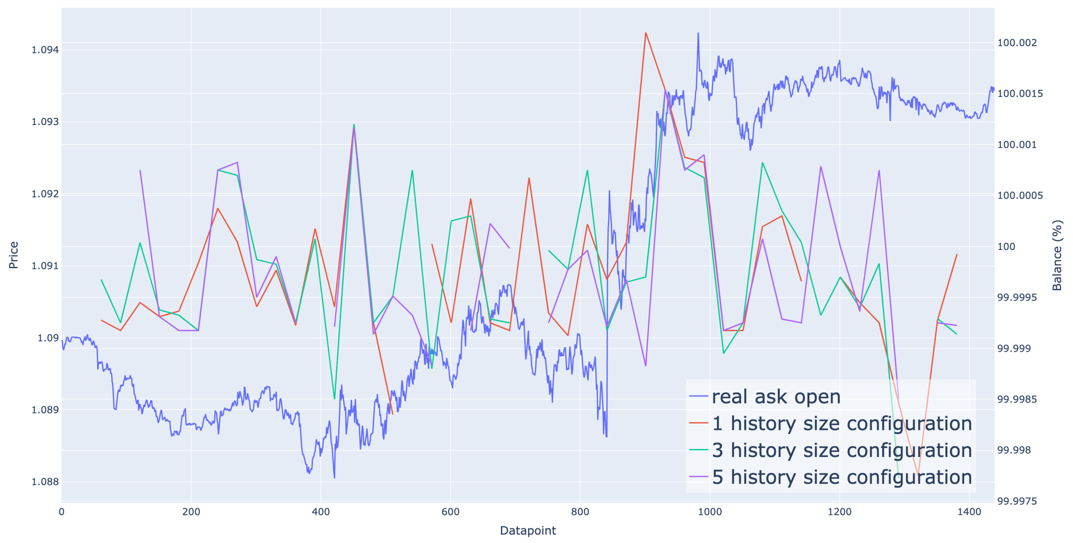


Figure 7.13: The average balance change for each 30-minute time frame, by smoothing configuration and the real open prices of the day-one ask candles.

The performance of the network taking a single candle is preferred here, as it is able to identify profitable entry points that the ones with a higher history size cannot, as can be seen, for instance, in datapoints 390 and 720 above. It also achieved greater highs while having comparably moderate lows. This is then the configuration we will use for this strategy, which will be compared to the performance of ARIMA(3,1,3).

Backtesting allowed us to make these assessments and see what might or might not work in live trading. For instance, the neural network smoothing appears to be a poor strategy and, if not for comparing it during live trading for the sake of completeness, we would disregard it based on the assessment we made above. It also permitted us to check the concentration of trades in higher order ARIMA models, and also verify that the single input neural network performed quite well in the high-low crossover despite this not being the case for the ARIMA model of order 1.

We will now take the configurations for each strategy that we selected, and run them concurrently in live trading for a few separate periods, and compare the results in the next section.

7.3 Live Trading Results

During a period of five days, November 23^{rd} and 27^{th} 2020, between 13h and 16h GMT, and we put these strategies to the test in the live market. We did not make trades actual real trades, but used a paper trading account instead. It differs only in the sense that we are not actually using money to fund the account: it uses virtual money. Using paper trading accounts is said to be another (final) step in backtesting, allowing for true out-of-sample testing [5], so for our purposes it will be a great fit. It will perfectly replicate how we would lose or gain money with our strategies. The currency pairs we traded were the three most traded by volume, that have USD as quote currency. They are the following:

Table 7.7: Currency pairs used and the respective daily average percentage of total value for all executed transactions during April 2019 [18].

Pair	**Percentage of all Trades**
EUR/USD	24.0%
GBP/USD	13.2%
AUD/USD	5.4%

For each pair, we ran all strategies. In particular, we should note the performance of the control strategies, the buy/sell and hold:

Table 7.8: The profit or loss of the control strategies for each day during the live trading period, in pips per unit traded.

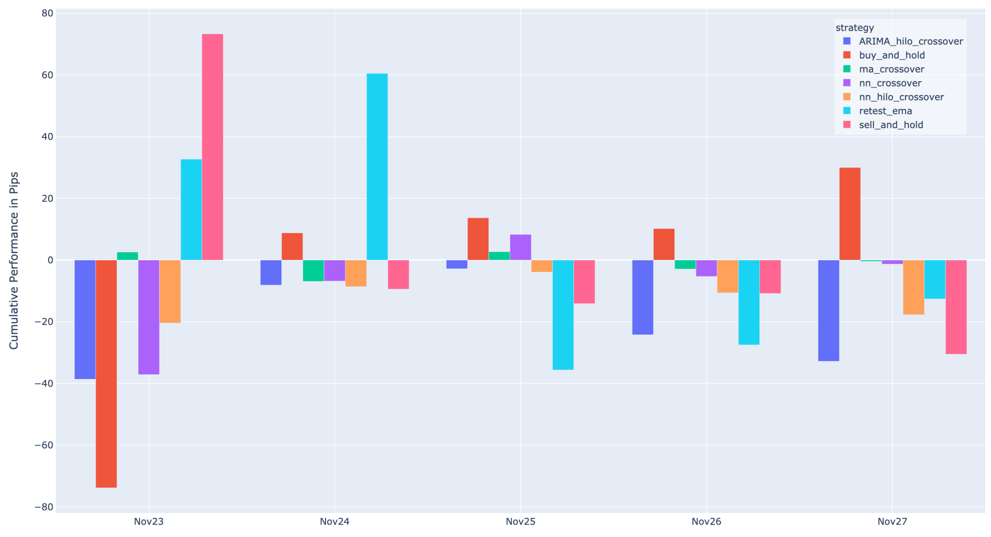
Pair	**Strategy**	**Nov. 23rd**	**Nov. 24th**	**Nov. 25th**	**Nov. 26th**	**Nov. 27th**
EUR/USD	Buy	-73.8	8.9	13.7	10.2	29.8
EUR/USD	Sell	73.3	-9.4	-14.1	-10.8	-30.5
GBP/USD	Buy	-92.8	23.6	22.2	6.7	-20.3
GBP/USD	Sell	91.1	-25.3	-24.0	-8.9	18.5
AUD/USD	Buy	-53.7	13.2	6.7	-0.2	11.5
AUD/USD	Sell	51.6	-15.2	-8.6	-1.7	-13.6

Note that they are almost symmetrical, the difference being attributed to the bid-ask spread. Had we joined the buy and the sell, the result would naturally be always negative. To help give a better picture of how the market behaved during these disjoint periods, the volatility measured by the standard deviation of the prices in pips, can be seen below.

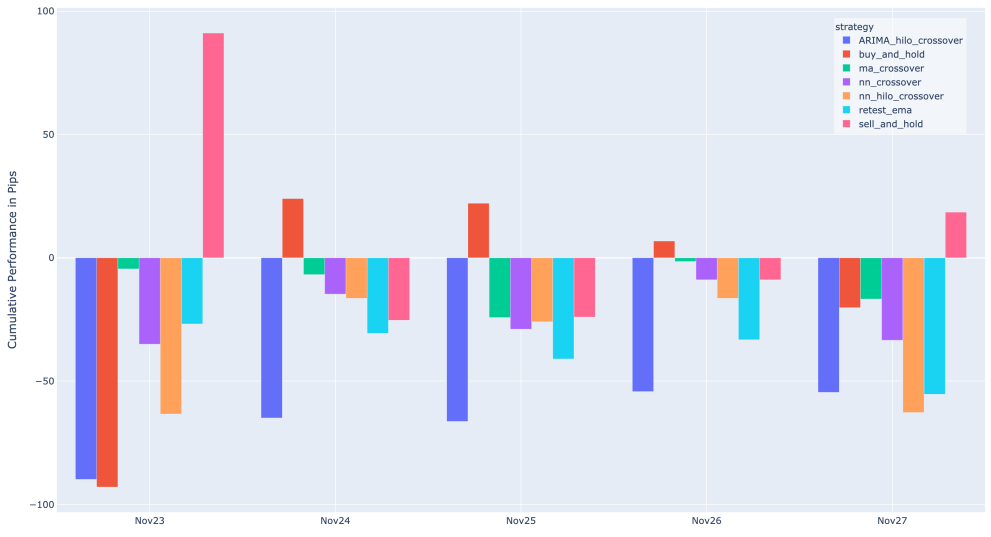
Table 7.9: The estimated sample volatility, in pips, averaged from all candle items (which show naturally very small differences), for each day and currency.

Pair	**Nov. 23rd**	**Nov. 24th**	**Nov. 25th**	**Nov. 26th**	**Nov. 27th**
EUR/USD	41.0	17.5	18.4	8.3	14.6
GBP/USD	36.2	10.6	6.5	4.0	9.7
AUD/USD	21.8	10.2	6.8	2.0	8.5

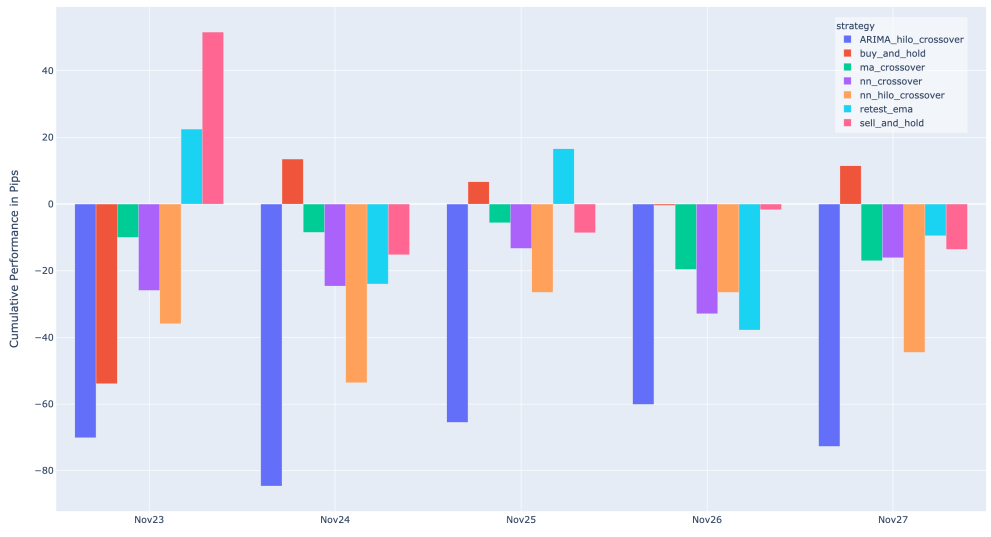
Following this, we will present the performance of each strategy in each currency, in pips profited or loss per unit traded in each day. In order to better understand the evolution of the performance of each strategy during the week, the respective cumulative yield in pips, per day in each currency, follows after that.



(a)

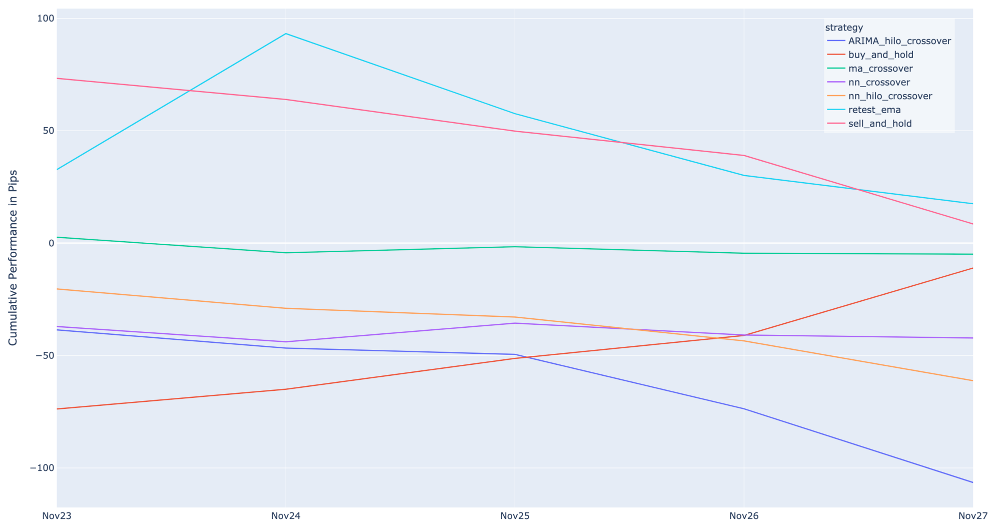


(b)

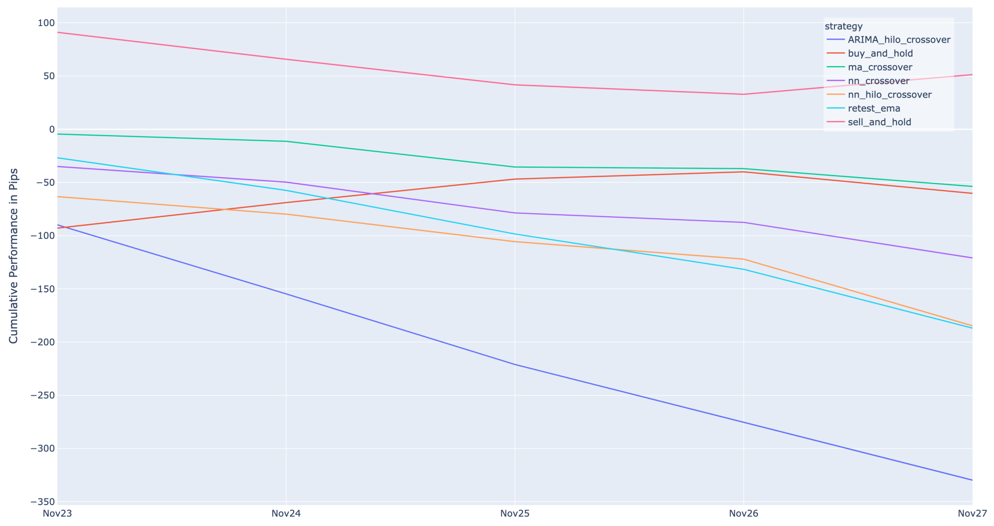


(c)

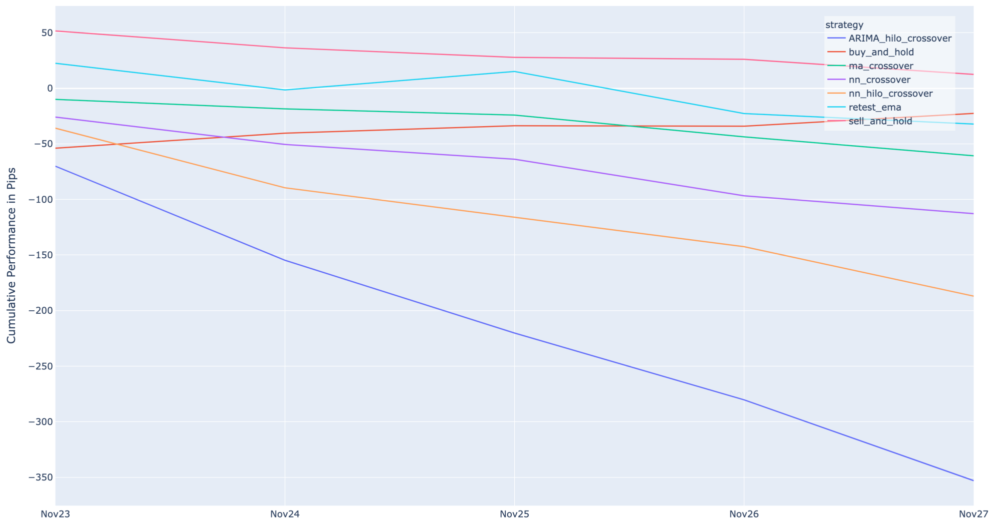
Figure 7.14: The performance of each strategy per day, for (a) EUR/USD, (b) GBP/USD and (c) AUD/USD.



(a)



(b)



(c)

Figure 7.15: The cumulative performance of each strategy throughout the week, for (a) EUR/USD, (b) GBP/USD and (c) AUD/USD.

One thing we can obverse right away, is that it is quite hard for any strategy we have implemented to beat the marked. Indeed, the only one that did was the EMA retest and not by a great amount. However, this strategy is the one that shows the highest volatility by far, as can be in the graph below, for EUR/USD. For the other two currencies this was similarly observed.

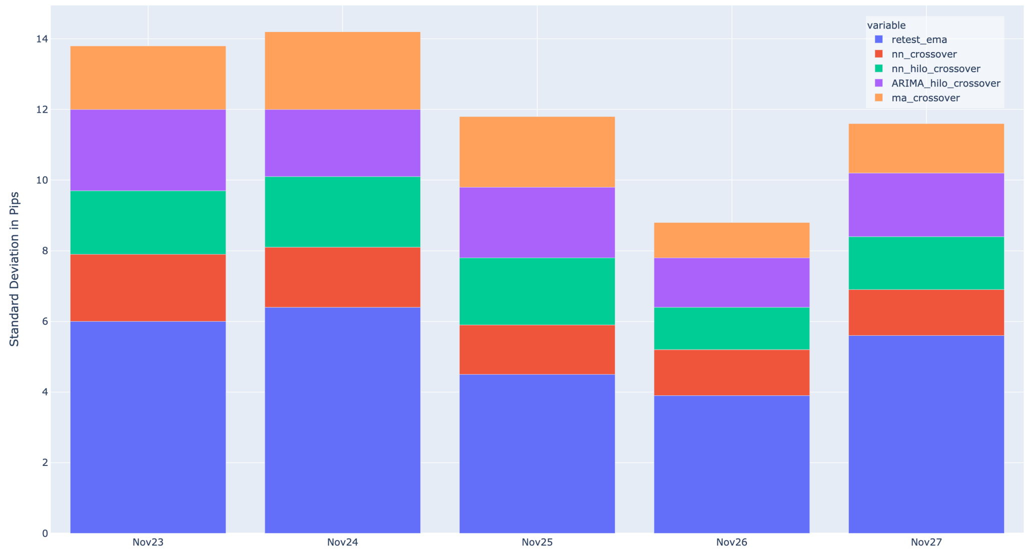


Figure 7.16: The estimated sample standard deviation for each strategy during the week, for EUR/USD.

The moving averages seemed to have performed better overall, losing the least amount of money, both having akin cumulative performance curves. Despite this, the moving average smoothing remained closer to being positive, which can be attributed to performing significantly less trades, as can see in the graph below.

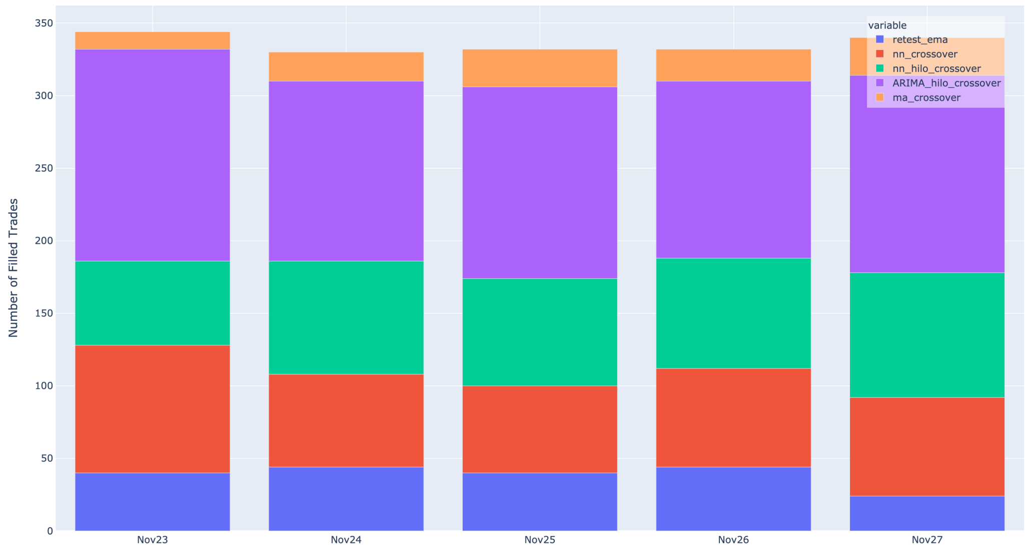


Figure 7.17: The amount of order placed that were filled for each strategy during the week, for EUR/USD.

This graph is for EUR/USD, but as the volatility, the amount of trades placed in other currencies have a similar relationship between strategies. The difference in the amount if trades performed by the MA and the NN crossovers is somewhat expected, as the moving average does a much better job at smoothing the price over time, given that the neural network does not put an equal weight to all history points. Therefore, its curve will have a more jagged appearance, similarly to the price itself, which will of course lead to more entry and exit signals.

Also interesting to note from the graph above, is the fact that the exponential moving average retest, despite showing the highest volatility, is a close second to the strategy with the least amount of trades done. This means that the volatility is not the cause for this, but rather a particular setting in the orders it places. This was the only strategy to have a larger take profit, specifically 2 times the amount in pips we are willing to risk in each trade, *i.e.* the difference between the entry price and the stop loss. All other strategies had their take profit equal to the amount risked.

Finally, regarding the strategy we devised for predicting the highs and lows, we can see that it performed quite poorly, especially for the ARIMA model. We can say that at least the neural network based strategy commonly incurred in lesser losses than the EMA retest, although this could be expected given the latter's higher risk tolerance. The neural network model also did almost always strictly better than its counterpart. During the last hour of the first trading session, the indicators created for both high-low strategies are displayed below, for EUR/USD.

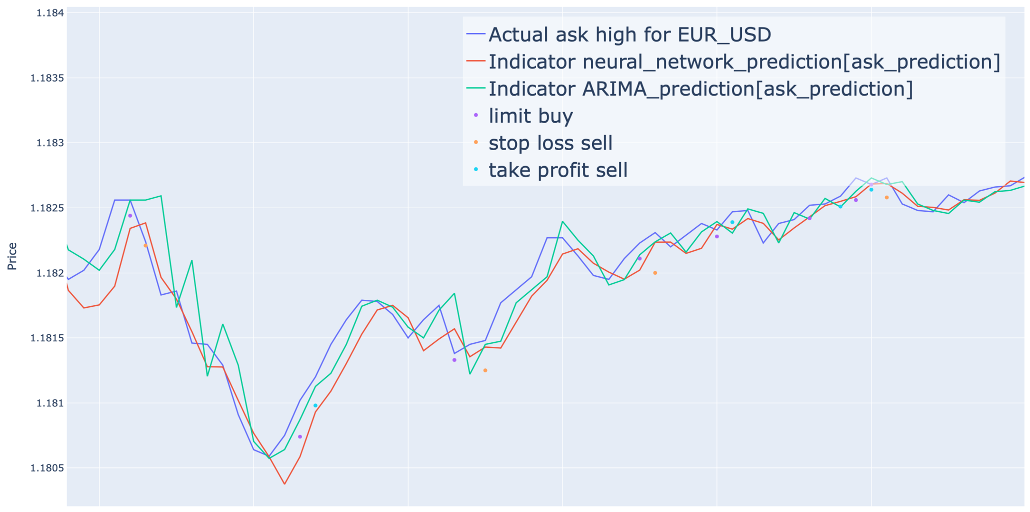


Figure 7.18: The EUR/USD high prices during November 23rd, 16h to 17h GMT and the predictions created by each model, and the trades placed by the ARIMA based model.

It seems like the spiked approach of the ARIMA model might be triggering too many trades, including false entries we would not want to flag. This is confirmed by the previous graph where we see that the ARIMA high-low crossover has done the highest amount of trades out of all strategies, by far. Despite this, we can see in the table below the predictions accuracy, measured in pips, which show good results for both models.

Table 7.10: The estimated sample standard deviation, mean absolute error and root mean squared error, in pips, of the price highs predictions during the week, for both models.

Pair	**Model**	**Std. Deviation**	**Mean Abs. Err.**	**Root Mean Sq. Err.**
EUR/USD	ARIMA	0.93	0.45	0.93
EUR/USD	Neural Network	0.95	0.79	1.15
GBP/USD	ARIMA	1.68	0.87	1.70
GBP/USD	Neural Network	1.12	0.90	1.37
AUD/USD	ARIMA	0.89	0.46	0.89
AUD/USD	Neural Network	0.64	0.55	0.80

Interestingly, although the ARIMA model shows consistently better results than the neural network according to absolute error, it presents greater error deviation than its counterpart. This is in agreement to what we observed above, with the predictions of the neural network model better tracing the general price behavior. Moreover, the NN was able to obtain similar results while being less computationally expensive since we do not need to retrain it in each time step. This saves a few precious milliseconds whenever we need to perform a new prediction, that quickly add-up given the amount of indicators we are estimating with each new candle. This would significantly decrease the backtesting duration, which would allow us to obtain results quicker and perform more tests. Most importantly, in live trading it could make a notable difference if we decide to start managing more currencies concurrently or working in smaller time frames.

Chapter 8

Conclusion and Next Steps

A fully fledged automated trading system is incredibly complex. However, the platform we built is quite capable, including in-depth backtesting with advanced features like execution verification and active order management with support for most order types. For our purposes, it completely simulated a broker aside from one small detail, slippage. Slippage is the difference between the expected price our order will be filled according to the most current data available to us, and the price at which it executes. However, this is a quite small factor in the foreign exchange market, usually ranging from 1 to 3 pips in the major pairs [8].

There are also many things we implemented but hardly messed with. Indeed, while an algorithmic trading system can do the trading part of the work for us, researching and backtesting different configurations and strategies is a full-time job [4]. For instance, our program fully supports candles of any size whereas we solely considered 60 seconds for this parameter. Also, sizing the trades could have a chapter just for it [8] and we merely used a fixed percentage of our total balance, despite having support to easily expand this topic, without having to change any existing code. The strategies themselves have settings that we didn't explore, such as the take-profit or stop-loss level, and error margins for crossovers. In sum, this means that we have constructed an algorithmic trading system that besides taking already into account many important characteristics of automated trading, with which we can further experiment with, was also designed with extensibility in mind so we can easily perform further research.

One limitation we faced was the amount of currency pairs being traded concurrently. Because the prediction models are comparably slower to compute a value for their indicator, if we ran our strategies on too many currencies at the same time, we would start to experience a delay on the pricing stream. This is because our program would take some time computing all the indicators when creating each candle, only reaching the step of reading the next price point (and possibly creating a candle for another currency pair) a few seconds later. One second or two did not prove to be a problem (which occurred when trading three pairs simultaneously), but when crossing this limit we would start to experience serious slippage issues or even failing to fill orders due to automatically triggering stop-losses or take-profits. One way we could curb this in the future, is to allow multi-threading in our program. This would permit one thread to deal with creating indicators while another is already reading the next price in the stream. This new thread would stand-by its execution if it retrieves a price for the currency pair for which the indicators are being created, in which case an other different thread would start retrieving prices. Although this adds another degree of complexity, it would greatly improve the capability of our system to manage trades in multiple pairs at the same time.

Regarding the results we have obtained, the lack of good performance by the strategies we implemented is in great part due to the fact that we have not thoroughly optimized each one. Also, and maybe most importantly, they are quite elementary. Of course, creating and optimizing strategies is the job of a technical or quantitative trader, depending on the type of strategies we would aim to use. Creating a consistent market-beating strategy could entail a work of the same magnitude as this one, and this was not our objective.

We have wholly developed a strategy from scratch that employs a machine learning prediction engine that, while not crossing the positive threshold, managed to beat the market performance of an equivalently defined ARIMA-based strategy. Furthermore, we have verified that, in comparison to ARIMA models, neural networks have the valuable characteristic of being much less computationally intensive in the long run: we can train a model taking a very large amount of historical data and then reliably use it to make as many predictions as needed without having to retrain it. We have also detailed the ML model used for this purpose from the most simple components to sophisticated types of networks. This was our objective, performing research in a completely different field, computer science, although closely related to ours. We have consolidated a great deal of knowledge regarding neural networks, and presented how they can be applied in a financial market. We touched all three corners of mathematical finance by comparing the neural network model to an established mathematical model for time-series forecasting, laying out the underlying theory as well. Thus, we succeeded in the work we set out to do, paving the way for the great deal of exploring and *learning* that there is still to do.

Bibliography

[1] P. J. Brockwell and R. A. Davis. *Time Series: Theory and Methods.* Springer, 1991.

[2] *Buying And Selling Currency Pairs.* URL: www.babypips.com/learn/forex/buying-selling-currency-pairs (visited on 11/27/2020).

[3] M. Capinski and T. Zastawniak. "Mathematics for Finance: An Introduction to Financial Engineering." In: (2004). ISSN: 00029890.

[4] E. P. Chan. *Quantitative Trading: How to Build Your Own Algorithmic Trading Business.* John Wiley & Sons, Inc., 2009.

[5] E. P. Chan. *Algorithmic Trading: Winning Strategies and Their Rationale.* John Wiley & Sons, Inc., 2013.

[6] A. Cofnas. *The Forex Trading Course: A Self-Study Guide to Becoming a Successful Currency Trader.* John Wiley & Sons, Inc., 2015.

[7] M. Davis and A. Etheridge. *Louis Bachelier's Theory of Speculation: The Origins of Modern Finance.* Princeton University Press, 2011.

[8] B. Donnelly. *The Art of Currency Trading: A Professional's Guide to the Foreign Exchange Market.* John Wiley & Sons, Inc., 2019.

[9] *Dukascopy Historical Data Feed.* URL: www.dukascopy.com/swiss/english/marketwatch/historical/ (visited on 11/27/2020).

[10] S. N. Durlauf and P. C. B. Phillips. "Trends versus Random Walks in Time Series Analysis." In: *Econometrica* 56.6 (1988), pp. 1333–1354. ISSN: 00129682. DOI: 10.2307/1913101. URL: http://www.jstor.org/stable/1913101.

[11] W. Enders. *Applied Econometric Time Series.* John Wiley & Sons, Inc., 2015.

[12] A. I. Galushkin. *Neural network theory.* Springer, 2007.

[13] W. H. Greene. *Econometric Analysis.* Prentice Hall, 2003.

[14] A. Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems.* O'Reilly Media, 2017.

[15] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Predictiont.* Springer, 2009.

[16] G. Hinton. *Coursera Lectures on Neural Networks, University of Toronto.* URL: www.cs.toronto.edu/~hinton/coursera_lectures.html (visited on 11/27/2020).

[17] J. Hull. *Options, Futures and Other Derivatives.* Pearson, 2015.

[18] B. for International Settlements. *Triennial Central Bank Survey - Foreign exchange turnover in April 2019.* Tech. rep. BIS, 2019. URL: www.bis.org/statistics/rpfx19.htm.

[19] B. Johnson. *Algorithmic Trading and DMA.* Springer, 2010.

[20] *Keras.* URL: www.keras.io (visited on 11/27/2020).

[21] *Keras Flatten Layer.* URL: www.keras.io/api/layers/reshaping_layers/flatten/ (visited on 11/27/2020).

[22] *Keras Regression Losses.* URL: www.keras.io/api/losses/regression_losses/ (visited on 11/27/2020).

[23] D. P. Kingma and J. L. Ba. "Adam: A method for stochastic optimization." In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings.* International Conference on Learning Representations, ICLR, 2015. arXiv: 1412.6980. URL: https://arxiv.org/abs/1412.6980v9.

[24] T. L. Lai and H. Xing. *Statistical Models and Methods for Financial Markets.* Springer, 2008.

[25] M. Lewis. *Flash Boys: A Wall Street Revolt.* W. W. Norton & Company, 2014.

[26] *Limit Orders.* URL: www.babypips.com/forexpedia/limit-order (visited on 11/27/2020).

[27] T. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[28] *OANDA.* URL: www.oanda.com/rw-en/ (visited on 11/27/2020).

[29] *OANDA Candle Response.* URL: developer.oanda.com/rest-live-v20/instrument-df/ (visited on 11/27/2020).

[30] *OANDA Candle Response.* URL: developer.oanda.com/rest-live-v20/order-df/ (visited on 11/27/2020).

[31] *OANDA Fees.* URL: www.oanda.com/uk-en/trading/our-charges/ (visited on 11/27/2020).

[32] *OANDA Orders.* URL: www1.oanda.com/forex-trading/learn/getting-started/order-types (visited on 11/27/2020).

[33] *OANDA Price Response.* URL: developer.oanda.com/rest-live-v20/pricing-df/ (visited on 11/27/2020).

[34] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms.* Vol. 9781107057135. Cambridge University Press, 2013. DOI: 10.1017/CBO9781107298019.

[35] C. Smith. *How to Make a Living Trading Foreign Exchange A Guaranteed Income for Life.* John Wiley & Sons, Inc., 2010.

[36] I. O. for Standardization. *ISO - Currency codes*. 2020. URL: www.iso.org/iso-4217-currency-codes.html (visited on 11/27/2020).

[37] *Stop Orders*. URL: www.babypips.com/forexpedia/stop-order (visited on 11/27/2020).

[38] C. Suisse. *Technical Analysis - Explained*. 2010. URL: www.credit-suisse.com/pwp/pb/pb_research/technical_tutorial_de.pdf (visited on 11/27/2020).

[39] *Tensorflow*. URL: www.tensorflow.org (visited on 11/27/2020).

[40] *Tensorflow Optimizer (Minimize)*. URL: www.tensorflow.org/api_docs/python/tf/keras/optimizers/Optimizer (visited on 11/27/2020).

[41] P. Wang. *The Economics of Foreign Exchange and Global Finance*. Springer, 2005. DOI: 10.1007/3-540-28524-5.

[42] B. P. Welford. "Note on a Method for Calculating Corrected Sums of Squares and Products." In: *Technometrics* (1962). ISSN: 00401706. DOI: 10.2307/1266577.

[43] P. Wilmott. *Paul Wilmott Introduces Quantitative Finance*. John Wiley & Sons, Inc., 2007.