UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Prediction model for Forex based on neural networks

Bachelor Thesis

Degree in computer science

Specialization in Computing

Sergi Mestres Canales

Director: Enrique Romero Merino

GEP Tutor: Joan Sarda

26th April 2022

Resum

El mercat de divises o Forex és un dels mercats més grans del món, amb milions de transaccions per dia per valor de trilions de dollars. Poder anticipar els moviments del mercat pot ser una activitat molt lucrativa sempre que s'avaluï correctament el risc de cada transacció.

L'objectiu d'aquest treball és trobar una manera de predir el mercat de divises usant xarxes neuronals, ja que les xarxes neuronals han demostrat ser en múltiples ocasions una eina de predicció molt potent.

Així doncs, el treball consistirà a desenvolupar un sistema per experimentar amb models de predicció i avaluar les seves capacitats predictives.

Resumen

El mercado de divisas o Forex es uno de los mercados más grandes del mundo, con millones de transacciones por día por valor de trillones de dólares. Poder anticipar los movimientos del mercado puede ser una actividad muy lucrativa siempre que se evalúe correctamente el riesgo de cada transacción.

El objetivo de este trabajo es encontrar una forma de predecir el mercado de divisas usando redes neuronales, ya que las redes neuronales han demostrado ser en múltiples ocasiones una herramienta de predicción muy potente.

Así pues, el trabajo consistirá en desarrollar un sistema para experimentar con modelos de predicción y evaluar sus capacidades predictivas.

Abstract

The forex market is one of the largest markets in the world, with millions of transactions per day with trillions of dollars in value. Being able to anticipate market movements can be a very lucrative activity as long as the risk of each transaction is properly assessed.

The aim of this project is to find a way to predict the forex market using neural networks, as neural networks have repeatedly proved to be a very powerful prediction tool.

Thus, the project will consist of developing a system for experimenting with prediction models and evaluating their predictive capabilities.

Contents

List of Figures

List of Tables

List of Listings

1 | Context And Scope

1.1 Context

This is a Bachelor Thesis of the Computer Engineering Degree, specializing in Computing, done in the Facultat d'Informàtica de Barcelona of the Universitat Politècnica de Catalunya directed by Enrique Romero Merino.

1.1.1 Introduction

Neural Networks

Neural networks are computing systems inspired by the biological neural networks that constitute animal brains. Neural networks are able to learn from data in order to make predictions when new data is presented. Although neural networks can generate very good predictions, they can also perform very poorly. In order to be able to make good predictions it's important to be careful when performing the training, as you could end with a trained neural network that is very good at predicting the training data but very bad at predicting new data, that's what is called overfitting. It is also very important to have good quality data and formatted correctly in order to make it easy for the neural network to learn from it, neural networks are very powerful tools, but they are not magical, if you use bad data in they will struggle to make good predictions.

A neural network is constituted of one input layer, one or more hidden layers and one output layer, each layer has one or more neurons, you can see a representation of a neural network on figure 1.1. Each neuron can has many inputs and produces a single output that can be sent to many neurons. Neurons have an activation value which is the weighed sum of all the connections that input to it, neurons also have an activation function used to produce the output from the weighted sum. Each connection has a weight value that determines how much of the input it receives will be used.[2]

There are many different types of neural networks, and each one suits best different kinds of problems. Part of this project will be doing some

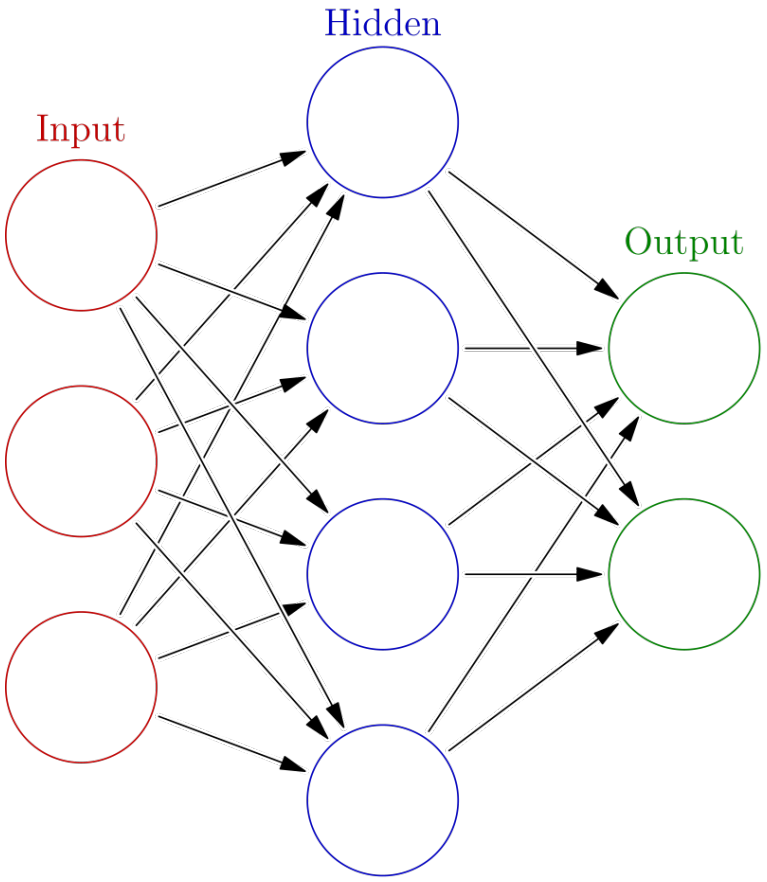


Figure 1.1: Representation of neural network basic structure. [2]

research on neural networks to determine which may work best for this problem.

FOREX

The Foreign Exchange Market (FOREX) is one of the most important and liquid markets, as It's where currencies are traded. In 2019 the EUR currency had an average daily turnover of over 2 trillion dollars [5], that is, if we added the value of all the transactions that involved EUR in a single day, we would get that value. Inside this market the most traded pair is the EUR/USD, as it involves the two major currencies in the world. As EUR/USD it's the most traded pair it will be the main focus on this project.

Market Data

In order to understand the markets we first need to understand how the data is formatted, so I will explain how to interpret market data. The most common way to represent market data is by using candlesticks, this candlesticks give price action information about a certain period of time,

for example, I we looked at a chart of EUR/USD with M5 period each candlestick would give us information about a period of 5 minutes, on figure 1.2 is an example of a EUR/USD chart on M5 period. The most

Figure 1.2: EUR/USD chart on M5 period. [Own screenshot]

common periods to use are M1, M5 and M15 for short-term trading also known as daytrading, M30, H1 and H2 for mid-term trading also known as swing trading and finally H4, D1 and W1 for long-term trading also known as investing. M stand for Minute, H for Hour, D for Day and W for Week. On this project I will use data from M1 period, as my goal is two generate predictions for daytrading.

Japanese Candlestick

The japanese candlestick, candlestick or for short, is the main form of representing data in the markets, it gives us information about what happened in a certain period of time. The candlestick is very useful as it gives us the High, Low, Open and Close price, all this in a visual form as you can see on figure 1.3. High and Low are the maximum and minimum prices reached during that period, Open and Close are the prices at the beginning and end of the period. Normally a green candlestick represents a closing price higher than an opening price and a red candlestick represents the opposite. There are a lot of theories about candlestick patterns, but that it's not in the scope of this project.

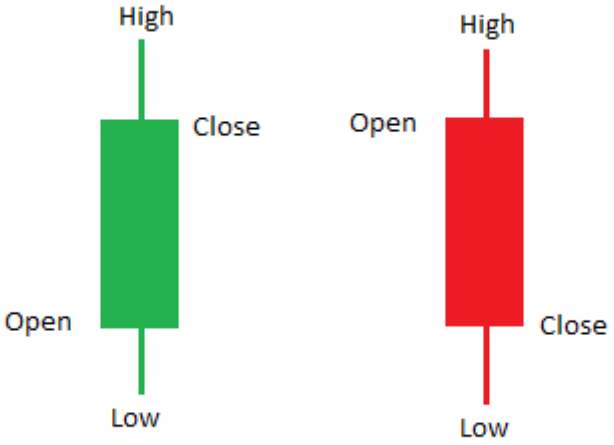


Figure 1.3: Candlestick format with legend. [3]

Indicators

Indicators are a way to represent a certain characteristic of the market easily, this indicators are plotted on the charts to help visualize and understand what is going on. The most used and simplest indicator is the Moving Average, it is basically plotting a point for each candlestick that represents the average of past close prices within a certain period, then all points are connected to form a line representing how the average is moving, you can see and example of a two moving averages on figure 1.4. There are many different indicators, some may be useful, other may be completely useless, many people believe that indicators serve no use as they are all lagging, meaning they only give useful information once the event has finished. For this project I will be using only the Moving Average indicator.

1.1.2 Problem to be solved

One of the most difficult problems but at the same time most rewarding in monetary terms is being able to predict market movements. If you are able to make better than random predictions in a high liquidity market, the potential for profit is very high. So this projects aims to find a way to make reliable and precise predictions for the Foreign Exchange Market, and in order to make that predictions we are going to employ Neural Networks as they have the ability to learn from data and adapt.

1.1.3 Stakeholders

This project could be of use to the following statements:

- **Individuals:** Any individual that has any interest in the markets and making predictions could find this work useful.

Figure 1.4: EUR/USD chart on M5 period with two Moving Average indicators, red for period 20 and orange for period 50. [Own screenshot]

- **Companies:** Companies interested in developing systems for automated trading could find this work really useful.
- **Banks:** Banks use their capital to generate more capital and often they do so using the market, this tool could complement their systems for prediction and risk management.
- **Hedge Funds:** Hedge Funds are alternative instruments of investing, most of them base their growth on predicting the market, so they would find this work really useful.
- **Myself:** I'm really interested in this project as it's my final project and I will graduate after it's successful completion. But also, I'm interested in using the results of this project to create an automated system capable of trading on it's own, as this was the reason behind choosing this project.

1.2 Justification

1.2.1 Previous Studies

Trying to make predictions about the market is not a new thing, many individuals and companies are trying it every day. Using neural networks to make predictions about the market isn't either a new thing, thou it's more limited to big companies with huge budgets for research. As only big companies tend to develop this prediction models based on Neural Networks, it all remains very occult. The companies that succeed in developing such a prediction model don't share their research with the public.

1.2.2 Justification

I developed an interest for markets over the last years. I find them fascinating and puzzling, and also see them as a challenge. My first attempt at making some predictions was feeding a neural network with a lot of data from the market to predict the next close price, it was a failure. At that moment I didn't have any profound knowledge or understanding of the markets, so I decided to study them and return when I had gained enough knowledge.

Over the last year I've developed a lot of automatic systems based on what I learned so far. Many of these systems performed very poorly, others did very good under some circumstances but failed in the long run. But recently I've found a system that could take advantage of neural networks, as I've been using it for some time and has proven quite successful. The problem of coding this system is that it needs to adapt to the market, and I found that very hard with traditional coding approach. I'm convinced that if done right, a neural network would be very well suited for this task.

This project, if successful, will generate a trained model, ready to make predictions. These predictions will be very fast to make compared to doing extensive calculations of probabilities on the market data, meaning that I could try to find parameters in my strategies efficiently as each run would be very fast, that it's not possible if for each candlestick you have to performs calculations that last more than milliseconds. Also there is a huge reduction in the energy waste for each prediction, meaning that using this would have a lot less impact on the environment.

Also, making predictions using a neural network is very powerful, as the neural network has the capacity to learn complex behaviours that would be very hard to model mathematically and would be even harder to discover. The neural network has the potential to learn something from the market that no one has realised or only few people know.

1.3 Scope

1.3.1 Objectives and sub-objectives

As said before, the objective of this project is to develop a prediction model for the EUR/USD pair using Neural Networks. In order to do so I divided the project in sub-objectives:

- **Raw data:** The first sub-objective is to obtain raw data, needed for the training, although it has to be pre-processed first.
- **Pre-processed data:** The next step is obtaining pre-processed data ready for training, I will need to program a pre-processor in order to do so.
- **Working neural network:** Then, I need to have a working neural network, once I finish coding it, I can begin using the pre-processed data for training.
- **Trained model:** After all the necessary training has been done I finally have a trained model, ready to make predictions.
- **Evaluation:** With the trained model, I can finally do and evaluation and determine whether it is really useful or not.

1.3.2 Requirements

This are the requirements to ensure the quality of this project:

- **Good data:** It's a must too have good data in order to achieve a model capable of making good predictions.
- **Good Programming practices:** In order to develop and maintain the code it's crucial to adhere to good programming practices.
- **Validation:** In order to prevent over-fitting it's necessary to use a model validation technique such as cross-validation.

1.3.3 Potential obstacles and risks

- **Deadline:** There is a deadline to be met for this project, this may result in drastic changes to the project to met this deadline. This would potentially generate bad results and not of the quality I expected.
- **Bugs in libraries:** Some of the software I decide to use may have some bugs that generate errors on the code and hinder the progress of the project.

- **Bad training data:** It may happen that the data chosen its not of the quality I expected, meaning it has a lot of gaps or bad information.
- **Very long training times:** Sometimes training neural networks takes a long time, depending on the number of hidden layers and neurons per layer it may go from minutes to days.
- **Computer break:** It is very improbable, but it may happen that the computer I use for this project breaks due to internal failure or external damage. This would hinder the project as there would be some time I would be unable to advance.

1.4 Methodology and Rigor

1.4.1 Methodology

I've decided to use the Kanban methodology, this methodology consists of cards that represent a task, this cards are then assigned to different state columns. This are the columns I will be using:

- **To do**
- **In progress**
- **Testing**
- **Completed**

I really like this methodology because I find it very easy and intuitive to use. Also, it's very visual, so you get an idea of how the project is going in a second.

For this project I will be using Trello, since it has a very accessible web platform, and I'm already familiar with it. On Trello you generate columns and then assign cards to this columns, representing their state. When the state of some task changes, you simply drag it from a column to another to represent that change. Also, you can add notes and images to each card to better understand the task. On figure 1.5 there is an example of how a Trello board would look like.

1.4.2 Validation

In order to keep track of the project and have a way to revert changes if something goes wrong, I will be using Git. Finished and working code will be on the *master* branch, while code being developed with be in the *dev* branch. The git repository will be private and shared with my director Enrique Romero Merino.

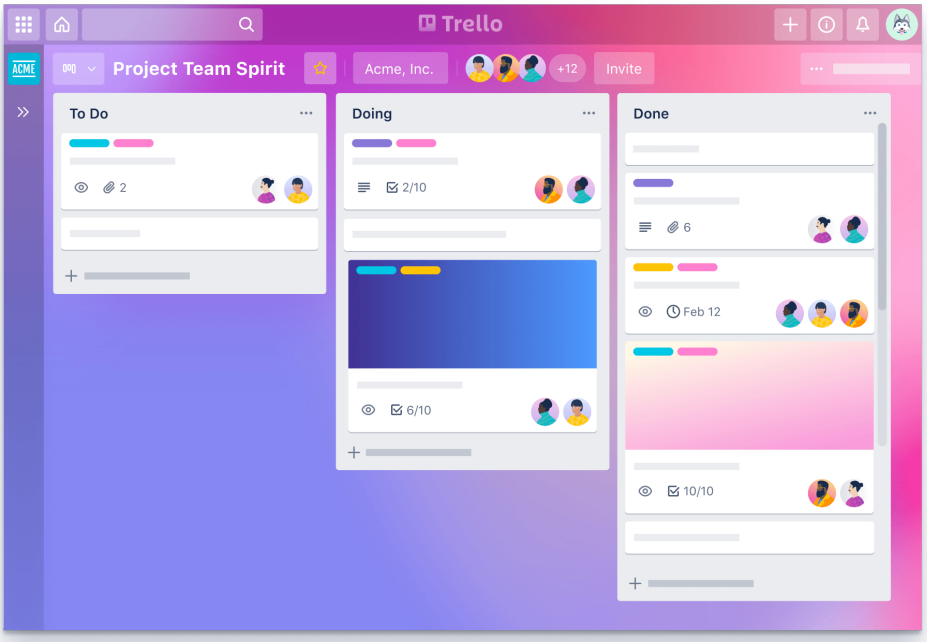


Figure 1.5: Trello board with three columns and some cards for each column. [18]

2 | Project Planning

This project has a workload of 540 hours, distributed from September 20th 2021 to January 24th 2022, assuming that the next available oral defense date is chosen. There are a total of 126 days between those two dates, that means an average dedication of ∼4 hours per day.

2.1 Task definition

For the sake of clarity I've divided the tasks into five categories:

2.1.1 Project management

First of all, there is there project management, necessary for the correct development of the project.

- **Context and scope:** Give a general idea about the project, define the problem to be solved, justify why its a good idea to solve this project using this alternative and expose the sub-objectives.
- **Project planning:** Plan the project taking account of the time available and the tasks that need to be performed.
- **Budget and sustainability:** Give and approximation of the cost and environmental impact of this project.
- **Final project definition** Compile a final document with all previous three parts, correcting and improving based on the tutor feedback.
- **Meetings:** Regular meetings with the director to discuss the progress of the project, solve any issue it may arise and decide how to proceed.

2.1.2 Research

After the project management part has been completed, except for the meetings, I begin with the research. It's the first part of the actual project as it's needed for everything else.

- **Software:** Analyse the available software solutions to build neural networks, this includes the language and the libraries.
- **Neural Network:** Research neural networks to gain enough knowledge to work with them and find the one(s) that best suit this project.
- **Language:** If a unknown language it's chosen it should be studied.
- **Library:** Study how the chosen library works and how to use it to build the desired neural network.
- **Data source:** Investigate the different data sources available and decide which is the best in terms of quality and quantity.
- **Market:** It is necessary to have a robust knowledge of how the markets work and the usual ways to represent data. I've already done that so I can skip this task, but it is nonetheless a primordial task.

2.1.3 Development

Once the research has been done and there is a clear understanding of everything needed for the project I begin with the development of the code needed for the project, I need to develop a pre-processor and a neural network to be able to experiment.

- **Pre-processor:** Develop the needed code to perform pre-procsesing on the data and prepare it for the neural network.
- **Neural Network:** Code the Neural Network according to the previous research done.
- **Testing and debugging:** Test the code to see if it runs without errors, debug it and get it ready for experimenting.

2.1.4 Experiment and analysis

Once I have the pre-processor and neural network ready I can begin experimenting.

- **Collect data:** Retrieve the necessary raw data from the decided data source.

- **Experiment:** Use the data and the pre-processors to experiment with the neural network, adjusting the different parameters. Neural networks have a strong trial and error factor.
- **Analysis:** Once satisfied with the neural network performance, analyse it's results, determining the viability to use the generated model in the real market.

2.1.5 Document and Present

Once I finish experimenting, analysing and I am satisfied with the results, I need to document everything. To do so, I first collect all the information, and then I write the documentation.

And finally I have to prepare for the oral presentation, so I should craft a slide presentation and prepare for the possible questions the the tribunal may ask.

2.2 Resources

All projects need enough resources in order to progress adequately and produce good results. I will state all the resources needed divided into four categories:

2.2.1 Human Resources

This project has three human resources involved, this are:

- **Researcher/Developer:** Responsible of doing the research and carry the development of the project.
- **Director:** The director has the role of guiding the student in charge of the project, helping him solve any problem that may arise.
- **GEP Tutor:** The GEP tutor is in charge of providing feedback and guide the student on the project management done at the beginning, before starting the actual research and development, ensuring that the project is clearly defined and contextualized.

2.2.2 Hardware Resources

For this project I will need one computer, in this case I will be using a Laptop with the following specifications:

- **Model:** Lenovo ideapad 330
- **CPU:** i7-8750H CPU @ 2.20GHz

- **Ram:** 16GB DDR4 DIMM
- **GPU:** GTX 1050
- **Storage:** 256GB SSD / 1TB HDD

This project will make heavy use of the CPU and GPU, primarily the latter, as it will be used to perform the training of the neural network. Most libraries support training using the GPU and it's usually a lot more efficient than the CPU.

2.2.3 Software Resources

A lot of software is involved in this project, each software performs a different specific task. This is the software I will need:

- **Overleaf:** For generating all the documentation using latex.
- **Gmail:** For communicating via email to the Tutor and Director when needed.
- **Google Hangouts:** Used to carry out the meetings.
- **Github:** Used to store the project.
- **Programming language:** I will need a programming language to develop the Neural Network such a Python.
- **Code library:** The libraries I will be using to create the Neural Network such as TensorFlow.
- **Visual Studio Code:** I will also need an IDE to carry out the coding. For this, I will be using Visual Studio Code, as is the IDE I use the most.
- **The internet:** Internet access will be very import as it's going be the base for almost everything else.

2.2.4 Material Resources

Very little material resource will be needed, only a pencil, an eraser, a notebook and a calculator will be needed in order to sketch ideas.

2.3 Risk management: alternative plans

Every project has potential risk and obstacles, and they should be analysed ahead to be prepared in case they appeared. Next we have a list of the potential problems and their plan:

- **Deadline:** It may happen that the deadline of the project cannot be met due to a bad estimation of the needed time for each task,

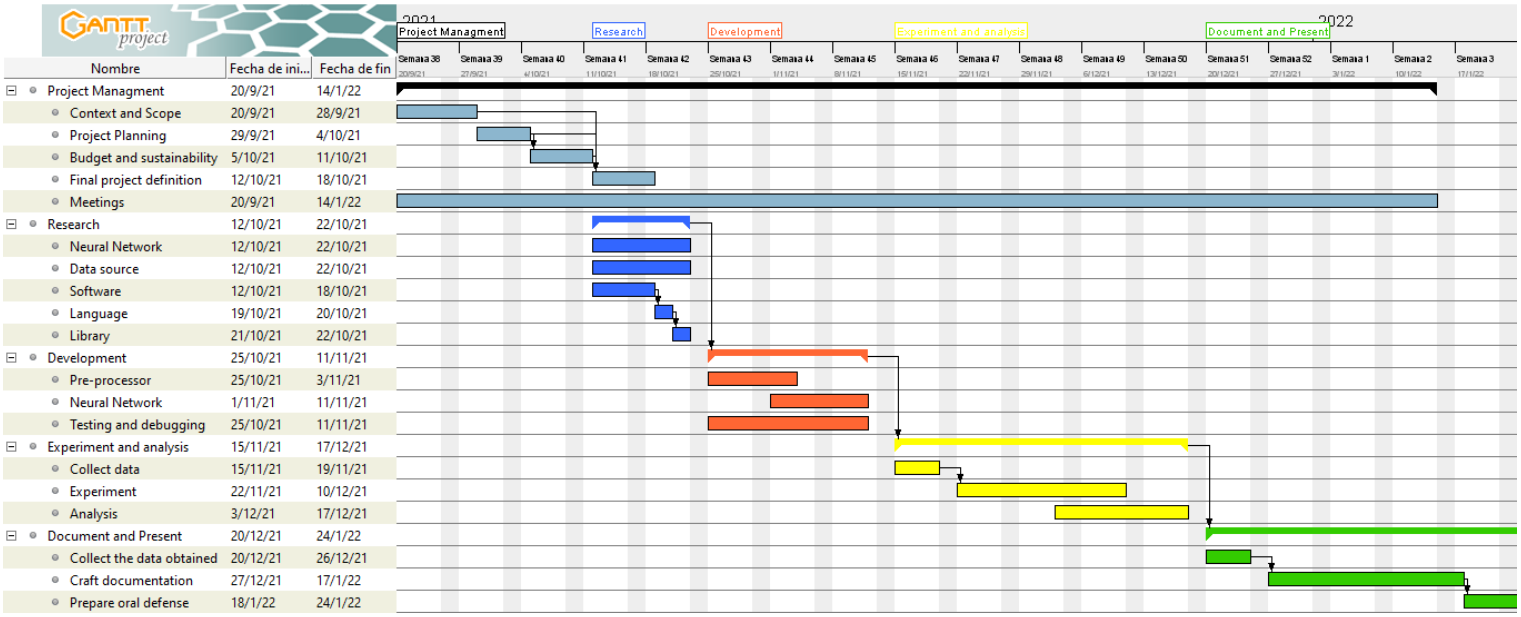
so if during the project it seems as though there is not enough time to finish something as planned, I should re-plan that part of the project and its dependence's accordingly. This would affect T3, T4 and T5 with an estimated impact of 20 hours.

- **Bug in library:** Sometimes libraries of code have bugs in it, generating wrong results. Our software choices should take stability in account to avoid this, but if a bug was found I should use another version of the same library. If another version doesn't solve the problem, I should change the library used for a similar one. This would affect T3 with and estimated impact of 5 hours.
- **Bad training data:** If the data used happens to be of bad quality, meaning it has a lot of gaps in it or misinformation, then I should chose another data source. This would affect T4 with an estimated impact of 5 hours.
- **Very long training times:** Training neural networks tends to be a lengthy task, depending on the number of neurons it can go from minutes to even days. So, if the network I built has a training time too long making experimentation in the planned time unfeasible, I should rebuild the neural network making it more light, that is, reducing the number of neurons per layer or reducing the number of layers. If I had access to more computers with better GPUs, I could use them speed up the training. This would affect T4 with and estimated impact of 20 hours.
- **Computer break:** Although its very improbable, the computer I will use for the development and experimenting could break. In that case I should find a new computer to use capable of performing the needed task, luckily I have access to another computer if that happened. This would affect T2, T3, T4 and T5 with an estimated impact of 5 hours.

2.4 Gantt chart and Task table

Id	Name	Hours
T1	Project Management	85
T1.1	Context and Scope	25
T1.2	Project Planning	10
T1.3	Budget and sustainability	10
T1.4	Final project definition	20
T1.5	Meetings	20
T2	Research	60
T2.1	Neural Network	20
T2.2	Data source	10
T2.3	Software	10
T2.4	Language	10
T2.5	Library	10
T3	Development	110
T3.1	Pre-processor	40
T3.2	Neural Network	50
T3.3	Testing and debugging	20
T4	Experiment and analysis	160
T4.1	Collect data	30
T4.2	Experiment	80
T4.3	Analysis	50
T5	Document and Present	125
T5.1	Collect the data obtained	20
T5.2	Craft documentation	80
T5.3	Prepare oral defense	25
	Total	**540**

Table 2.1: Estimated task workload. [Own calculations]



Nombre	Fecha de ini...	Fecha de fin
Project Managment	20/9/21	14/1/22
Context and Scope	20/9/21	28/9/21
Project Planning	29/9/21	4/10/21
Budget and sustainability	5/10/21	11/10/21
Final project definition	12/10/21	18/10/21
Meetings	20/9/21	14/1/22
Research	12/10/21	22/10/21
Neural Network	12/10/21	22/10/21
Data source	12/10/21	22/10/21
Software	12/10/21	18/10/21
Language	19/10/21	20/10/21
Library	21/10/21	22/10/21
Development	25/10/21	11/11/21
Pre-processor	25/10/21	3/11/21
Neural Network	1/11/21	11/11/21
Testing and debugging	25/10/21	11/11/21
Experiment and analysis	15/11/21	17/12/21
Collect data	15/11/21	19/11/21
Experiment	22/11/21	10/12/21
Analysis	3/12/21	17/12/21
Document and Present	20/12/21	24/1/22
Collect the data obtained	20/12/21	26/12/21
Craft documentation	27/12/21	17/1/22
Prepare oral defense	18/1/22	24/1/22

Figure 2.1: Gantt chart. [Own creation]

2.5 Changes to initial planification

Unfortunately, due to personal issues the time available for the project was less than I expected, so I decided to do the defense on April instead of January to be able to finish the project with the quality and completeness I envisioned.

2.5.1 Final task workload

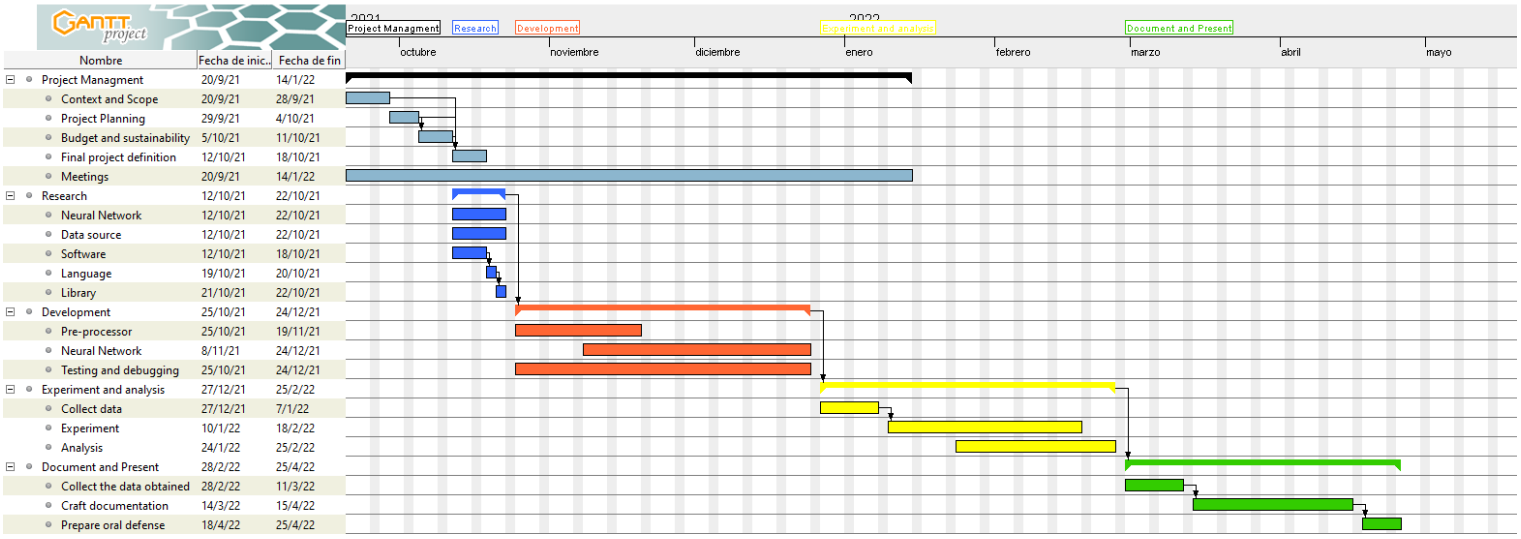
I underestimated the time I would need for the Development phase, Task T3.2 and T3.3 took more time than expected, T3.2 was completed with 70 hours approximately, and T3.3 with 30 hours. On Table 2.2 you can see the final workload.

2.5.2 Final gantt chart

On Figure 2.2 is the final gantt chart with the planification I followed in order to defend by April.

Id	Name	Hours
T1	Project Management	85
T1.1	Context and Scope	25
T1.2	Project Planning	10
T1.3	Budget and sustainability	10
T1.4	Final project definition	20
T1.5	Meetings	20
T2	Research	60
T2.1	Neural Network	20
T2.2	Data source	10
T2.3	Software	10
T2.4	Language	10
T2.5	Library	10
T3	Development	140
T3.1	Pre-processor	40
T3.2	Neural Network	70
T3.3	Testing and debugging	30
T4	Experiment and analysis	160
T4.1	Collect data	30
T4.2	Experiment	80
T4.3	Analysis	50
T5	Document and Present	125
T5.1	Collect the data obtained	20
T5.2	Craft documentation	80
T5.3	Prepare oral defense	25
	Total	**570**

Table 2.2: Final task workload. [Own calculations]



Nombre	Fecha de inic..	Fecha de fin
Project Managment	20/9/21	14/1/22
Context and Scope	20/9/21	28/9/21
Project Planning	29/9/21	4/10/21
Budget and sustainability	5/10/21	11/10/21
Final project definition	12/10/21	18/10/21
Meetings	20/9/21	14/1/22
Research	12/10/21	22/10/21
Neural Network	12/10/21	22/10/21
Data source	12/10/21	22/10/21
Software	12/10/21	18/10/21
Language	19/10/21	20/10/21
Library	21/10/21	22/10/21
Development	25/10/21	24/12/21
Pre-processor	25/10/21	19/11/21
Neural Network	8/11/21	24/12/21
Testing and debugging	25/10/21	24/12/21
Experiment and analysis	27/12/21	25/2/22
Collect data	27/12/21	7/1/22
Experiment	10/1/22	18/2/22
Analysis	24/1/22	25/2/22
Document and Present	28/2/22	25/4/22
Collect the data obtained	28/2/22	11/3/22
Craft documentation	14/3/22	15/4/22
Prepare oral defense	18/4/22	25/4/22

Figure 2.2: Final Gantt chart. [Own creation]

3 | Budget and Sustainability

Here I will describe the budget needed for the project, divided into costs per task, generic costs and other costs. Moreover, I will define management control mechanism in order to control the possible deviation that may appear in the project due to any problem that may arise. And finally, I will give some words regarding the sustainability of the project.

3.1 Budget

3.1.1 Personnel costs per activity

In this section I will estate all the tasks and their corresponding assignee, with cost estimations for each task derived from an estimation of each personnel salary. This project has five types of personnel involved, this are:

- **Project Manager:** The project manager is the one responsible for the planning, guidance, and supervision of the correct development of the project. The director, GEP Tutor and me will be involved in this role.
- **Programmer:** Is the one in charge of coding everything. I will play this role.
- **Tester:** The Tester is there to test the code and find any bugs or problems it may have, to ensure it runs well when experimenting. I will also do this role.
- **Researcher:** The researcher has to do the necessary research in order to have a clear understanding of everything needed for the project. I will be the researcher.
- **Technical writer:** Once the experimenting has been done and the results are ready, everything needs to be documented, and that's what the technical writer is in charge of. I will be documenting the project.

Following, there is a table displaying the salary and price per hour for each role, assuming that a year has 1750 working hours.

Role	Annual Salary (€)	Total including SS (€)	Price per hour (€)	Assignee
Project Manager	39.004	50.705,2	28,97	GEPT, T, S
Programmer	26.198	34.057,4	19,46	S
Tester	20.592	26.769,6	15,29	S
Researcher	35.259	45.836,7	26,19	S
Technical Writer	26.263	34.141,9	19,50	S

Table 3.1: Salaries for the different roles involved in the project.
GEPT: GEP Tutor, T: Tutor, S: Student. Data source: [6]

With this information I will compute the cost for each task, based on the number of hours assigned to that task. On Table 3.2 I detailed all the costs per task, this is known as CPA.

3.1.2 Generic costs

Amortization

I have to take into account the amortization of the resources used in this project, basically the laptop I will be using. It's estimated that the project will have a total workload of 540h. I've had this laptop for 5 years so the formula would be:

Amr (€) = Price * (1/5 years) * (1/126 days) * (1/4 hours) * Hours used

That gives an amortization of 214.28€.

Electric cost

Electricity cost is at 0,253€/kWh. This cost is only applied while the laptop is in use, so that's 540h of electricity use. My laptop consumes 135W, so 135W during 540h is 72kWh. With 72kWh at a price of 0,253€/kWh means an electric cost of 18.22€.

Internet cost

The internet rate cost is 60€/month. So the internet cost will be 540h * (1day/24h) * (1month/30day) * (60€/month) = 45€

Water cost

The water cost in my area is about 30€/month per person, so the water cost is 540h * (1day/24h) * (1month/30day) * (30€/month) = 22.5€.

Id	Name	Total Hours	Hours					Cost (€)
			Project Manager	Programmer	Tester	Researcher	Technical Writer	
T1	Project Management	85	85	20	20	20	20	4.071,25
T1.1	Context and Scope	25	25	0	0	0	0	724.25
T1.2	Project Planning	10	10	0	0	0	0	289.7
T1.3	Budget and sustainability	10	10	0	0	0	0	289.7
T1.4	Final project definition	20	20	0	0	0	0	579.4
T1.5	Meetings	20	20	20	20	20	20	2188.2
T2	Research	60	0	0	0	60	0	1571.4
T2.1	Neural Network	20	0	0	0	20	0	523.8
T2.2	Data source	10	0	0	0	10	0	261.9
T2.3	Software	10	0	0	0	10	0	261.9
T2.4	Language	10	0	0	0	10	0	261.9
T2.5	Library	10	0	0	0	10	0	261.9
T3	Development	110	0	90	20	0	0	2057.2
T3.1	Pre-processor	40	0	40	0	0	0	778.4
T3.2	Neural Network	50	0	50	0	0	0	973
T3.3	Testing and debugging	20	0	0	20	0	0	305.8
T4	Experiment and analysis	160	0	0	0	160	0	4190.4
T4.1	Collect data	30	0	0	0	30	0	785.7
T4.2	Experiment	80	0	0	0	80	0	2095.2
T4.3	Analysis	50	0	0	0	50	0	1309.5
T5	Document and Present	125	25	0	0	0	100	2674.25
T5.1	Collect the data obtained	20	0	0	0	0	20	390
T5.2	Craft documentation	80	0	0	0	0	80	1560
T5.3	Prepare oral defense	25	25	0	0	0	0	724.25
	CPA cost	**540**	**110**	**110**	**40**	**240**	**120**	**14564.5**

Table 3.2: Estimated task workload and corresponding cost. [Own calculations]

Work space

The development of the project will take place at my parents house. My area has a cost of 12€/m2 approximately and my room has 8m2 so that would mean 96€/month. I have to add the use of other rooms in the house, so that would be a plus of 150€approximately. Summing all I have a work space cost of 246€/month. The project lasts 5 months so that would be a total of 246€/month * 5month = 1230€.

Generic cost of the project

On the following table we can see all the generic costs and their total sum. The generic cost is known as CG cost.

Concept	Cost (€)
Amortization	214.28
Electric cost	18.22
Internet cost	45
Water cost	22.5
Work space	1230
CG cost	**1530**

Table 3.3: Generic cost of the project [Own calculations]

3.1.3 Other costs

Contingencies

During the project obstacles may arise that eat part of the budget, so it's necessary to add a contingencies margin of 15%. The total cost of the project is 16094.2€(CPA + CG), so the contingency margin will be of 2414.13€.

Incidental costs

I also need to be prepared in case any of the risks described on the previous chapter happen. In order to calculate the each cost I make an estimation of the cost if that incident happened and it's probability, then I multiply both to get the cost. Everything is detailed on table 3.4.

3.1.4 Total cost

The total cost is computed on Table 3.5.

Incident	Estimated Cost (€)	Risk (%)	Cost (€)
Deadline (20h)	400	20	80
Bug in library (5h)	100	5	5
Bad training data (5h)	100	15	15
Very long training times (20h)	400	15	60
Computer break (5h)	100	5	5
Total			**165**

Table 3.4: Incidental costs [Own calculations]

Activity	Cost (€)
CPA cost	14564.5
CG cost	1530
Contingency	2414.13
Incidental cost	165
Total cost	**18673.63**

Table 3.5: Total cost [Own calculations]

3.1.5 Management control

I need a mechanism to model the possible budget deviations, as they will help visualize the deviation that may occur. If a deviation is negative, meaning that it's using more money that estimated, I will have to use the contingencies budget. Here all the necessary models are detailed:

Human resources deviation

Refers to the deviation in budget that may occur due to any of the personnel working less or more than the estimation. It's calculated as follows:

Human resources deviation =

$$\sum_{i \in pit} (est\_cost\_per\_hour_i - real\_cost\_per\_hour_i) \times total\_real\_hours_i$$

where pit refers to personnel involved in task.

Amortization deviation

Refers to the deviation that may occur due to not using the laptop as much as estimated or alot more. It's calculated as follows:

Amortization deviation =

$$\sum_{i \in hr} (est\_usage\_hours_i - real\_usage\_hours_i) \times price\_per\_hour_i$$

where hr refers to all hardware resources.

Total cost deviation

Joins all the deviations in the different tasks, not taking incidents and contingencies into account.

Total cost deviation = $est\_general\_cost - real\_general\_cost$

3.2 Sustainability

3.2.1 Economic dimension

Regarding PPP: Reflection on the cost you have estimated for the completion of the project

If find that the estimation is realistic and could be carried out in real life. Also it's cost is small compared to the potential profit from the project, so I conclude that the cost is well adjusted.

Regarding Useful Life: How are currently solved economic issues (costs...) related to the problem that you want to address (state of the art)?

There are few studies that are public regarding this problem, so any company or business that would like to take advantage of market predictions would find it very difficult to do so as they would have to develop a proprietary solution. With this project there would be no need to develop a proprietary solution, as they could directly contract a service developed by me that would provide them with useful predictions based on the models developed on this project.

How will your solution improve economic issues (costs ...) with respect other existing solutions?

Being able to make predictions using a neural network is very efficient once trained, so there would be little cost in making the predictions. Any company or business that wanted predictions and currently have a proprietary method or a service contracted that has a lot of cost could reduce that cost. Moreover, any company that need to buy or sell anything to the market could optimize their orders using the results of this project, because, even though this project it's focused on the Forex market, it's results could easily be translated to other markets, there would only be a need for a new model very similar to the one developed in this project.

3.2.2 Environmental dimension

Regarding PPP: Have you estimated the environmental impact of the project?

I have not estimated the environmental impact of the project. Nevertheless, I have to say that the impact generated from this project will probably very low as I will only use my laptop and I will not generate any material waste. There will be some mild electricity consumption when experimenting and training the neural network. The electricity consumption could be very heavy if I developed a huge neural network that needed a lot of training, but this will not be the case here.

Regarding PPP: Did you plan to minimize its impact, for example, by reusing resources?

I did not plan to minimize it's impact intentionally but using a light neural network instead of a huge one will minimize the impact.

Regarding Useful Life: How is currently solved the problem that you want to address (state of the art)?

It is solved by doing extensive calculations and predictions based on probability and statics by companies with big budgets for research. It is also solved using neural networks, as many companies claim to do so, but their models are not accessible to public.

How will your solution improve the environment with respect other existing solutions?

As stated on the economic dimension part, using neural network models is very efficient once trained, so the environment impact derived from those predictions would be greatly reduced.

3.2.3 Social dimension

Regarding PPP: What do you think you will achieve -in terms of personal growth- from doing this project?

From this work I will gain a more profound understating of neural networks and how to work with them, especially in how to use them to predict a future event. I will also grow my understanding of the markets and how efficient they really are. It will also help me improve in my project management skills and programming skills in general.

Regarding Useful Life: How is currently solved the problem that you want to address (state of the art)?

As said before, it's solved by doing extensive computation using probability and statistics.

How will your solution improve the quality of life (social dimension) with respect other existing solutions?

The project aims to produce a model capable of generating predictions, and that will prevent unnecessary work from people wanting to achieve something similar, so that resources could be allocated somewhere else.

Regarding Useful Life: Is there a real need for the project?

It could be discuss whether trying to make profit of the market is a ethic thing to do or not, but there will always be individuals and companies interested in doing so, so providing a efficient way to predict the market is needed in order to reduce the cost and impact it would be caused by using less efficient methods. Also there is a personal need that will impact society, as this project, if successful, will be the backbone for the funding of other projects I have in mind, such as companies that provide to society, so in a sense that makes the project very needed.

4 | Research

The first part of the project consists on doing research on different aspects and deciding which tool, sources or technologies to use. This part is crucial as the correct development of the whole project resides on the quality of decisions made on this phase.

4.1 Data source

I needed to find a data source to use, from where I would download the raw data I would use to train and validate the model.

4.1.1 Available options

Here are three options I found:

1. **histdata.com:** A web page offering free forex historical data, limited to one year of data per dowload. Data needs to be merged. [8]
2. **eatradingacademy.com:** A site that has a tool to download historical forex data with different formats and time-frames easily, limited to 200000 bars per dowload. Data needs to be merged. [4]
3. **Metatrader 5:** A trading software offering the possibility to download the data visualized, and as many bars as you wanted as long as they where provided, and the data provided dates back to 1980. No need to merge data. [13]

4.1.2 Final decisions

My final decision was to use Metatrader 5 for three main reasons, first that the data didn't need to be merged so I would spend less time on pre-processing, secondly because it is very easy to use and the data comes well formatted, and lastly because I already knew the software and knew it provided accurate and quality data.

4.2 Programming Language

The programming language to use was a clear choice, I would use Python [14] as it's the most used language in the machine learning community, with lots of libraries and examples.

4.3 Neural Network library

Then I had to decide wich python library/ies I would use in order to create the model and train it.

4.3.1 Available options

There were two main options to consider:

1. **Pytorch** An open-source machine learning framework based on the Torch library, claiming to accelerate the process of going from research to production. [15]
2. **Tensorflow & Keras** Tensorflow [17] is an open-source library for developing machine learning models and automated learning in general. Keras [10] is a library made for python to easily develop neural networks on top of Tensorflow, Microsoft Cognitive Toolkit or Theano.

4.3.2 Final decisions

I decided to use the Tensorflow & Keras mix as as I already had some experience with it so I could spend less time on learning, and also because Google collab has native support for it.

4.4 Neural Network architecture

For the architecture of the neural network I decided to use LSTM layers as they are really good at finding relationships on timeseries data. LSTM [9, 11, 12] stands for long short-term memory as their goal is to find relationships on timeseries data on the short and long term. The final neural network should have three parts as follows:

1. **Input layer** First there is an input layer with shape = (n_timesteps_data, n_features), this input layer also has a dropout layer that can be adjusted.
2. **Hidden layer/s** The output of the input layer is passed on to the first layer of the hidden layers. The number of hidden layers can be adjusted and goes from 1-Inf. The hidden layers produce an

output shape of (n_neurons) on the final layer and have a shape of (n_timesteps_data, n_neurons) on intermediary layers. There is also a dropout layer at the end that can be adjusted.

3. **Output layer** Finally the output from the hidden layers is passed onto the output layer, this layer is in charge of producing a single value so it has an output shape of (1) with a linear activation in order to be able to generate any value.

4.5 Programming environment

I needed to choose an environment that made developing a neural network easy and efficient.

4.5.1 Available options

I found two good candidates:

1. **Visual Studio Code & Anaconda** Visual studio code [19] is and IDE with support for python syntax and Jupiter notebooks, it makes developing code easy and enjoyable. Anaconda [1] is an environment creation software designed to make it easy to start working without having to configure all the libraries.
2. **Google Colab** A free tool provided by google that functions as an online IDE for Jupyter notebooks, is synchronized with Google drive, executes the code on the cloud and comes with all the needed libraries preinstalled. It's designed to be used by machine learning researchers, and the executions have access to GPU's to speed up the training times. [7]

4.5.2 Final decisions

Because of the easy of use and the ability to run on the cloud without occupying my computer I decided to use Google Colab. Also, running the code on Google Colab tend to be faster than on a local machine since it has access to a lot of fast GPU's. Because of the fact that Google Colab already stores a copy on the cloud, I'm not going to use GitHub.

5 | Development

I splited the development phase in two parts, one for the pre-processor and another one for the neural network. At the end I developed two pre-processors and two neural networks but they are essentially the same. I did this because the director advised me to test with other data since the results I was getting were not good. Using the new data proved that the model was correctly implemented and the bad results where due to the fact that the forex market seems to be nearly equivalent to a random walk, I will go on detail about this on the conclusions chapter 7.

From now on I will refer to the moving average as being the average of a sliding window with a certain size known as period, I will use MA for short. I will also refer to the future moving average being the MA that is x steps ahead, x being the period of the MA, I will use FMA for short.

5.1 Preprocessor notebook

The pre-processor was fairly easy to write since I didn't need to do much cleaning, I only needed to parse the data, generate extra columns for the moving averages and save to a CSV file. To manipulate the data I used pandas and numpy since they are efficient and easy to use.

Since I developed in Jupiter notebook I will go block by block detailing how the pre-proccesor works.

5.1.1 Initialization

The first block corresponds to the initialization, where I import the needed libraries. Listing 5.1.1.

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import os
```

Listing 5.1.1: First block of the pre-processor

5.1.2 Processing

The next block is in charge of reading the raw data file, renaming columns to a more readable format and generating the new moving average columns. The MA and FMA columns are generated on the fly using a list of periods, so a new MA with period x is generated for each entry on the list. At the end, rows with empty values on MA or FMA columns are discarded, and finally the resulting data frame is printed for verification. Block on listing 5.1.2.

```
col_names = [covers
  '<DATE>',
  '<TIME>',
  '<OPEN>',
  '<HIGH>',
  '<LOW>',
  '<CLOSE>',
  '<TICKVOL>',
  '<VOL>',
  '<SPREAD>']
raw_data = "drive/MyDrive/TFG/Data/Raw/EURUSD_M1.csv"
df = pd.read_csv(raw_data, sep='\t')
df = df.rename(columns={
    "<DATE>": "date",
    "<TIME>": "time",
    "<OPEN>": "open",
    "<HIGH>": "high",
    "<LOW>": "low",
    "<CLOSE>": "close",
    "<TICKVOL>": "tickvol",
    "<VOL>": "vol",
    "<SPREAD>": "spread"})

periods = [20, 50, 100, 200, 500, 1000]
for period in periods:
    ma_name = f"ma{period}"
    fma_name = f"fma{period}"
    df[ma_name] = df['close'].rolling(period).mean()
    ma = df[ma_name].values
    fma = np.zeros((len(ma)))
    fma[0:-period] = ma[period:]
    df[fma_name] = fma

df = df.iloc[max(periods):-max(periods)]
print(df)
```

Listing 5.1.2: Second block of the pre-processor

5.1.3 Visual validation

This block generates two plots in order to visually validate the data generated on the previous block, both of the plots show the close price and all the MAs. The first plot allows use to see how de MAs behave as its only the last 1000 steps (Figure 5.1), whereas the second plot covers all the steps (Figure 5.2). The code also saves the plots in a .svg in order to use them later. Block on listing 5.1.3.

```
plt.close('all')
plt.figure(figsize=(16.18,10))
plt.title('Last 1000 steps')
plt.plot(df['close'].values[-1000:], label='close')
for period in periods:
    ma_name = f"ma{period}"
    plt.plot(df[ma_name].values[-1000:], label=ma_name)
plt.legend()
plt.savefig('last_1000.svg', bbox_inches='tight')

plt.figure(figsize=(16.18,10))
plt.title('Whole dataset')
plt.plot(df['close'], label='close')
for period in periods:
    ma_name = f"ma{period}"
    plt.plot(df[ma_name], label=ma_name)
plt.legend()
plt.savefig('whole_dataset.svg', bbox_inches='tight')

plt.show()
```

Listing 5.1.3: Third block of the pre-processor

5.1.4 Saving

The last block is in charge of saving the dataset to a .csv, it also creates the needed folder if it doesn't exist. Listing 5.1.4.

```
if not os.path.exists('drive/MyDrive/TFG/Data/Pre'):
    os.makedirs('drive/MyDrive/TFG/Data/Pre')
df.to_csv('drive/MyDrive/TFG/Data/Pre/pre_EURUSD_M1.csv')
```

Listing 5.1.4: Fourth and last block of the pre-processor

Figure 5.1: First verification plot showing the last 1000 steps of close price and MAs

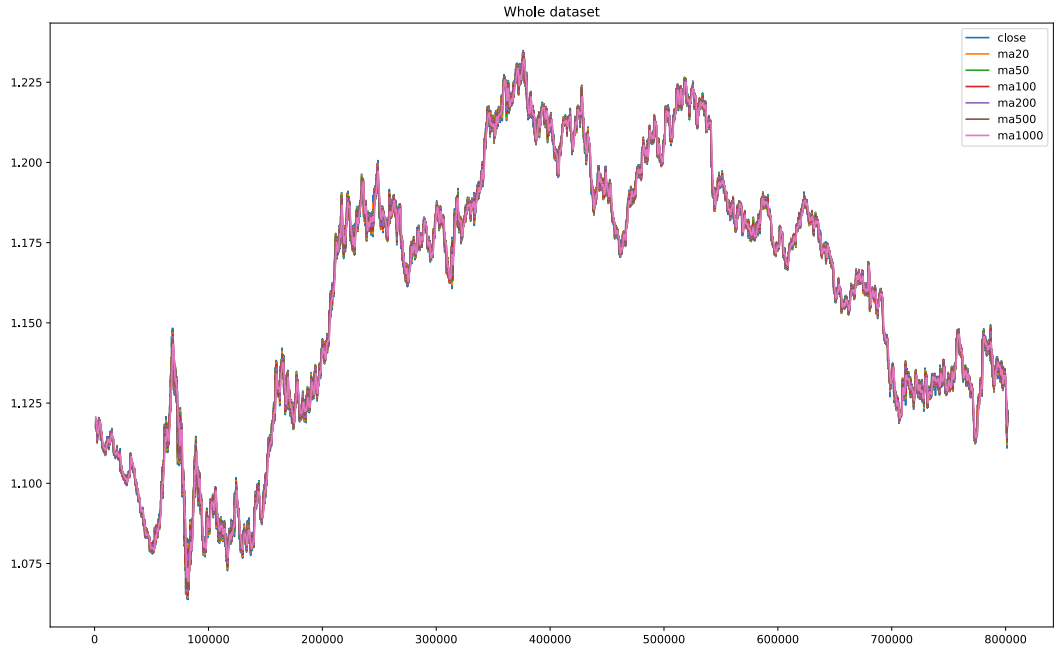


Figure 5.2: Second verification plot showing the whole dataset of close price and MAs

5.2 Neural Network notebook

The neural network was quite more laborious than the pre-processor. I did the neural network development in two parts. The first was developing a neural network that worked correctly, meaning it could train and produce a model, regardless of the quality of the predictions; this part also included developing all the code needed to read and prepare the data for the neural network; the code is stored at "NN_TFG.ipynb". The second part was developing a testing environment around this, in the sense of having a way to specify experiments and execute them; this code is stored at "NN_TFG_v2.ipynb". There is also a file called "NN_TFG_v2_House.ipynb" for testing with the alternate dataset, its essentially the same code as in "NN_TFG_v2.ipynb" but I preferred to have the code in a new notebook.

On the following subsections I will detail all the blocks of the final version.

5.2.1 Initialization

Here the initialization is quite more extensive, as it covers both the importation of all the needed libraries, and the implementation of custom a function for getting the data in batches as needed for the neural network. Block on listing 5.2.1.

```
# Initialitzation

import keras, math, os, json
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.callbacks import EarlyStopping
from matplotlib import pyplot as plt
from sklearn.preprocessing import StandardScaler
from keras.layers import LSTM, Bidirectional, Dense, Input, Dropout
from keras.losses import MeanSquaredError
from keras.models import load_model
from tensorflow.keras.optimizers import Adam
from joblib import dump, load
import keras.backend as kb

def get_bacthes(batch_size,
                start_idx,
                test_per,
                n_timesteps_data,
                sequence_stride,
                data,
                targets):
  first_idx = start_idx + n_timesteps_data
  n_rows = np.size(data, axis=0)
  n_features = np.size(data, axis=1)
```

```
27    sample_range = list(range(first_idx, n_rows, sequence_stride))
28
29    max_batch_size = len(sample_range)
30    if batch_size > max_batch_size:
31        batch_size = max_batch_size
32
33    train_batch_size = int(batch_size * (1-test_per))
34    test_batch_size = batch_size - train_batch_size
35
36    train_idx = sample_range[0:train_batch_size]
37    test_idx =
      ↪  sample_range[train_batch_size:train_batch_size+test_batch_size]
38
39    x_train = np.zeros((len(train_idx), n_timesteps_data, n_features))
40    y_train = {}
41    for key in targets.keys():
42      y_train[key] = np.zeros((len(train_idx), 1))
43    for i, i_train in enumerate(train_idx):
44        x_train[i] =
          ↪  np.copy(data[i_train-(n_timesteps_data-1):i_train+1])
45        for key in targets.keys():
46          y_train[key][i] = targets[key][i_train]
47
48    x_test = np.zeros((len(test_idx), n_timesteps_data, n_features))
49    y_test = {}
50    for key in targets.keys():
51      y_test[key] = np.zeros((len(test_idx), 1))
52    for i, i_test in enumerate(test_idx):
53        x_test[i] = np.copy(data[i_test-(n_timesteps_data-1):i_test+1])
54        for key in targets.keys():
55          y_test[key][i] = targets[key][i_test]
56
57    return x_train, y_train, x_test, y_test, first_idx
```

Listing 5.2.1: Initialization block of the neural network notebook

5.2.2 Data preparation

On this block is where the preprocessed data is readed and then prepared for the neural network. After reading the preprocessed data I get the close price list, from that list I get a diff list, that is, the list of the differences between adjacent values, and I also add a 0 at the beginning so that it has the same size as the list of close; this list is stored at **data_diff**. Then I create needed folders if they don't exist, and proceed to generate the **targets** and **target_scalers** dictionary. The targets dictionary is generated using the fma columns of the dataset, and the target_scalers is created using the StandardScaler() object from sklearn library.

Once I have the data_diff and targets ready, I use the custom function get_batches to get the data in a format ready for the neural network,

since what we have now is data_diff with shape **(length(data_diff), feature)** but we want data in the shape of **(batch, n_timesteps, feature)**. This function also splits the data for training and testing, generating x_train, y_train for train and x_test, y_test for validating. Block on listing 5.2.2.

```
# Data prepatation

plt.close('all')
#Read preprocessed data
df_train = pd.read_csv("drive/MyDrive/TFG/Data/pre_data_1m.csv",
                       sep = ',',
                       parse_dates=False)

#Clean nulls if present
if df_train['close'].isnull().any():
  print("There are null values, setting them to 0")
  df_train = df_train.fillna(0)

#Get close price and compute diffs
data = df_train[['close']].values
print(f"Data shape: {np.shape(data)}")
data_diff = np.concatenate([[[0]], np.diff(data, axis=0)])
print(f"Data diff shape: {np.shape(data_diff)}")

#Create needed folders
tfg_path = "drive/MyDrive/TFG"

experiments_path = tfg_path + "/Experiments"
if not os.path.exists(experiments_path):
    os.makedirs(experiments_path)

scalers_path = tfg_path + "/Scalers"
if not os.path.exists(scalers_path):
    os.makedirs(scalers_path)

#Prepare targets and target_scalers dict
periods = [20, 50, 100, 200, 500, 1000]
targets = {}
target_scalers = {}
for period in periods:
  col_name = f"fma{period}"
  if col_name in df_train.columns:
    fma = df_train[[col_name]].values
    target = fma - data
    target_name = f"{col_name}_diff"

    target_scaler = StandardScaler()
    target_re = target.reshape(-1, 1)
    target_scaler.fit(target_re)
    target = target_scaler.transform(target_re)
    scaler_path = os.path.join(scalers_path, f"{target_name}/")
    if not os.path.exists(scaler_path):
        os.makedirs(scaler_path)
```

```
49      dump(target_scaler, scaler_path +"/target_scaler.bin")
50      target_scalers[target_name] = target_scaler
51
52      targets[target_name] = target
53
54  print(targets.keys())
55
56  #Transform data to the format used by NN
57  bacth_size = 4000000
58  start_idx = 0
59  n_timesteps_data = 200
60  test_per = 0.2
61  stride = 100
62  x_train, y_train, x_test, y_test, _ = get_bacthes(bacth_size,
63                                                    start_idx,
64                                                    test_per,
65                                                    n_timesteps_data,
66                                                    stride,
67                                                    data_diff,
68                                                    targets)
69
70  n_features = np.size(x_train, axis=2)
71  print(f"n_features:{n_features}")
```

Listing 5.2.2: Data preparation block of the neural network notebook

5.2.3 Model generation

This block is in charge of reading the experiments.json file and creating the corresponding model for each configuration. The parameters that can be set are:

- **name:** The name of the experiment
- **hidden_layer:** Number of hidden_layers, minimum 1
- **neurons:** Neurons per hidden_layer
- **lr:** Learning rate
- **input_drop:** Dropout to be applied to the input
- **lstm_out_drop:** Dropout to be applied to the last LSTM layer output
- **target:** Which target to be predicted, can be any from the targets dictionary

On listing 5.2.3 there is an example of a possible experiments.json. Block on listing 5.2.4.

```
[
    {
        "name": "n20_hl1_fma100",
        "neurons": 20,
        "hidden_layers": 1,
        "lr": 0.001,
        "input_drop": 0,
        "lstm_out_drop": 0.2,
        "target": "fma100_diff"
    },
    {
        "name": "n100_hl1_fma100",
        "neurons": 100,
        "hidden_layers": 1,
        "lr": 0.001,
        "input_drop": 0,
        "lstm_out_drop": 0.2,
        "target": "fma100_diff"
    }
]
```

Listing 5.2.3: An example of a experiments.json file

```
# Model generation

experiments_json_path = os.path.join(experiments_path,
↪  "experiments.json")

#Read the config
with open(experiments_json_path) as json_file:
  experiments = json.load(json_file)
  print(experiments)

for config in experiments:
  #Read all parameters
  name = config.get('name', 'Test')
  config['name'] = name
  hidden_layers = config.get('hidden_layers', 2)
  config['hidden_layers'] = hidden_layers
  neurons = config.get('neurons', 20)
  config['neurons'] = neurons
  lr = config.get('lr', 0.001)
  config['lr'] = lr
  input_drop = config.get('input_drop', 0)
  config['input_drop'] = input_drop
  lstm_out_drop = config.get('lstm_out_drop', 0.2)
  config['lstm_out_drop'] = lstm_out_drop
  target = config.get('target', list(targets)[-1])
```

```
  config['target'] = target

  model_path = experiments_path+f"/{name}/Model"
  if not os.path.exists(model_path):
    os.makedirs(model_path)

  #Write this config to the own experiment path
  config_json = json.dumps(config, indent=4)
  config_path = os.path.join(experiments_path, f"{name}/config.json")
  with open(config_path, 'w') as outfile:
    outfile.write(config_json)

  #Generate, compile and save model
  kb.clear_session()

  input = Input(shape=(n_timesteps_data, n_features))
  drop_input = Dropout(input_drop)(input)

  if hidden_layers == 1:
    lstm_out = LSTM(neurons, return_sequences=False)(drop_input)
  else:
    lstm_prev = LSTM(neurons, return_sequences=True)(drop_input)
    for i in range(1,hidden_layers):
      if i == hidden_layers-1:
        lstm_out = LSTM(neurons, return_sequences=False)(lstm_prev)
      else:
        lstm_prev = LSTM(neurons, return_sequences=True)(lstm_prev)
  drop_lstm_out = Dropout(lstm_out_drop)(lstm_out)

  output_change = Dense(1)(drop_lstm_out)

  model = tf.keras.Model(inputs=input, outputs=output_change)
  model.summary()

  model.compile(loss=MeanSquaredError(),
  ↪  optimizer=Adam(learning_rate=lr))
  model.save(model_path+f"/model.h5")
```

Listing 5.2.4: Model generation block of the neural network notebook

5.2.4 Model training

On this block is where the models get trained one by one. The code checks whether a training was already performed in order to avoid training an already trained model. The block also generates a plot with the training loss, a train_info.json with the training information and a history.bin with the object returned by the fit function. Block on listing 5.2.5.

```
# Model training

for dir in os.listdir(experiments_path):
```

```
    full_dir = os.path.join(experiments_path, dir)
    if os.path.isdir(full_dir):
      with open(full_dir + '/config.json') as json_file:
        config = json.load(json_file)

      #Prepare paths
      model_path = os.path.join(full_dir, 'Model/')
      original_model = os.path.join(model_path, 'model.h5')
      fitted_weights = os.path.join(model_path, 'weights.h5')

      target = config.get('target')
      name = config.get('name')
      if os.path.exists(original_model) and target is not None:
        #Fit the model if no fitting present
        history_path = os.path.join(model_path, 'history.bin')
        if  not os.path.exists(fitted_weights) \
            or not os.path.exists(history_path):

          #Clear keras sesssion, load the original model and prepare
          ↪  the callbacks
          kb.clear_session()
          model = load_model(original_model)
          model_checkpoint_callback =
          ↪  tf.keras.callbacks.ModelCheckpoint(
              filepath=fitted_weights,
              save_weights_only=True,
              monitor='val_loss',
              mode='min',
              save_best_only=True)
          early_callback = EarlyStopping(monitor='val_loss',
                                         patience=10,
                                         mode='min',
                                         restore_best_weights=True)
          callbacks = [model_checkpoint_callback, early_callback]

          #Fit the model
          history = model.fit(x_train,
                              y_train[target],
                              validation_data=(x_test, y_test[target]),
                              batch_size=32,
                              shuffle=True,
                              epochs=100,
                              verbose=1,
                              callbacks=callbacks)

          dump(history, history_path)
          del model

        #Generate train info report if not present
        train_info_path = os.path.join(model_path, 'train_info.json')
        history = load(history_path)
        if not os.path.exists(train_info_path):
          val_loss = history.history['val_loss']
          loss = history.history['loss']
```

```
56
57          train_info = {
58              'min_val_loss': min(val_loss),
59              'final_loss': loss[np.argmin(val_loss)],
60              'min_loss': min(loss),
61              'epochs': len(loss),
62              'history': history.history
63          }
64
65          train_info_json = json.dumps(train_info, indent=4)
66          with open(train_info_path, 'w') as outfile:
67            outfile.write(train_info_json)
68
69        plt.figure()
70        plt.plot(history.history['loss'], label='Train')
71        plt.plot(history.history['val_loss'], label='Test')
72        plt.title(f"Model loss of {name}")
73        plt.ylabel('Loss')
74        plt.xlabel('Epoch')
75        plt.legend()
76        plt.savefig(os.path.join(model_path, f"model_loss_{name}.png"),
          ↪  bbox_inches='tight')
77        plt.savefig(os.path.join(model_path, f"model_loss_{name}.svg"),
          ↪  bbox_inches='tight')
```

Listing 5.2.5: Model training block of the neural network notebook

5.2.5 Prediction generation

This block is in charge of generating predictions for all the models, to do so it uses the custom function get_batches but with a strive of 1 in order to generate all the data that will be passed to the predict() function of the model. It only generates predictions if no predictions for that model are present to avoid unnecessary computation. The predictions are stored inside predictions.bin, the targets are stored in targets.bin. Block on listing 5.2.6.

```
1  #Prediction generation
2
3  plt.close('all')
4
5  for dir in os.listdir(experiments_path):
6    full_dir = os.path.join(experiments_path, dir)
7    if os.path.isdir(full_dir):
8      with open(full_dir + '/config.json') as json_file:
9        config = json.load(json_file)
10
11     #Prepate paths
12     original_model = os.path.join(full_dir, 'Model/model.h5')
13     fitted_weights = os.path.join(full_dir, 'Model/weights.h5')
```

```
    results_path = os.path.join(full_dir, 'Results/')
    predictions_path = os.path.join(results_path, 'predictions.bin')

    #Get data for generating predictions, no strive used
    x, y, _, _, first_idx = get_bacthes(40000000,
                                        0,
                                        0,
                                        n_timesteps_data,
                                        1,
                                        data_diff,
                                        targets)

    target = config.get('target')
    if os.path.exists(original_model) \
        and os.path.exists(fitted_weights) \
        and not os.path.exists(predictions_path) \
        and target is not None:

      #Load the model and the trained weigths
      model = keras.models.load_model(original_model)
      model.load_weights(fitted_weights)

      #Generate predictions
      pred = model.predict(x)

      #Save pred and y[target]
      if not os.path.exists(results_path):
          os.makedirs(results_path)
      dump(pred, predictions_path)
      dump(y[target], os.path.join(results_path, 'targets.bin'))
```

Listing 5.2.6: Prediction generation block of the neural network notebook

5.2.6 Result generation

This last block is used to generate the results that will be used to help me analyse and evaluate the model, that is, a collection of plots showing the predictions made at different index. First I specify a list with all the index I want to use, then for each one four plots are generated and stored, all plots show unscaled data. Next are the files for each plot:

- **diff_nt_(name)_(idx):** This plot show the prediction alone with no transformations.
- **diff_t_(name)_(idx):** Same as the previous plot but with the target value also ploted.
- **pred_(name)_(idx):** This plot shows the data, the target FMA and the transformed prediction.

- **recap_(name)_(idx):** This plot is a recap of all the three previous plots in a single image for convenience.

Block on listing 5.2.7.

```
#Result generation

plt.close('all')

for dir in os.listdir(experiments_path):
  full_dir = os.path.join(experiments_path, dir)
  if os.path.isdir(full_dir):
    with open(full_dir + '/config.json') as json_file:
      config = json.load(json_file)

    #Prepate paths
    results_path = os.path.join(full_dir, 'Results/')
    predictions_path = os.path.join(results_path, 'predictions.bin')
    targets_path = os.path.join(results_path, 'targets.bin')

    target = config.get('target')
    name = config.get('name')
    if os.path.exists(predictions_path) \
        and os.path.exists(targets_path) \
        and target is not None:

      #Prepare data
      predictions = load(predictions_path)
      y = load(targets_path)
      target_scaler = target_scalers[target]
      pred_unscaled =
      ↪  np.ravel(target_scaler.inverse_transform(predictions))
      y_unscaled = np.ravel(target_scaler.inverse_transform(y))

      interval = 200
      start_idxs = [-1000, -5000, -10000, -15000, -20000]

      #Generate figures for each index in start_idxs with interval
      for i, idx in enumerate(start_idxs):
        res_path = os.path.join(results_path, f"Res_{idx}")
        if not os.path.exists(res_path):
          os.makedirs(res_path)
        #Big plot with all the results
        fig = plt.figure(figsize=(20, 10))
        fig.suptitle(f"Prediction results of {name} at index {idx}",
        ↪  fontsize=16)

        ax0 = plt.subplot2grid((2, 3), (0, 0))
        ax0.set_title('Difference without target')
        ax0.plot(pred_unscaled[idx:idx+interval], '--b',
        ↪  label='Prediction')
        ax0.set_xlabel('Step')
        ax0.set_ylabel('FMA - Data')
        ax0.legend()
```

```
47
48          ax1 = plt.subplot2grid((2, 3), (1, 0))
49          ax1.set_title('Difference with target')
50          ax1.plot(pred_unscaled[idx:idx+interval], '--b',
            ↪  label='Prediction')
51          ax1.plot(y_unscaled[idx:idx+interval], '-g', label='Target')
52          ax1.set_xlabel('Step')
53          ax1.set_ylabel('FMA - Data')
54          ax1.legend()
55
56          data_used = np.copy(np.ravel(data[first_idx:]))
57          pred_fma = np.add(data_used, pred_unscaled)
58          real_fma = np.add(data_used, y_unscaled)
59
60          ax2 = plt.subplot2grid((2, 3), (0, 1), colspan=2, rowspan=2)
61          ax2.set_title('Prediction')
62          ax2.plot(pred_fma[idx:idx+interval], '--b',
            ↪  label='Prediction')
63          ax2.plot(real_fma[idx:idx+interval], '-g',
            ↪  label='Target(FMA)')
64          ax2.plot(data_used[idx:idx+interval], '-y', label='Close
            ↪  Price')
65          ax2.set_xlabel('Step')
66          ax2.set_ylabel('Data')
67          ax2.legend()
68
69          svg_path = os.path.join(res_path,
            ↪  f"recap_({name})_({idx}).svg")
70          png_path = os.path.join(res_path,
            ↪  f"recap_({name})_({idx}).png")
71          fig.savefig(svg_path, bbox_inches='tight')
72          fig.savefig(png_path, dpi=300, bbox_inches='tight')
73
74          #Individual plots
75          #Diff without target
76          fig = plt.figure()
77          fig.suptitle(f"Diff without target of {name} at index {idx}")
78
79          plt.plot(pred_unscaled[idx:idx+interval], '--b',
            ↪  label='Prediction')
80          plt.xlabel('Step')
81          plt.ylabel('FMA - Data')
82          plt.legend()
83
84          svg_path = os.path.join(res_path,
            ↪  f"diff_nt_({name})_({idx}).svg")
85          png_path = os.path.join(res_path,
            ↪  f"diff_nt_({name})_({idx}).png")
86          fig.savefig(svg_path, bbox_inches='tight')
87          fig.savefig(png_path, dpi=300, bbox_inches='tight')
88
89          #Diff with target
90          fig = plt.figure()
91          fig.suptitle(f"Diff with target of {name} at index {idx}")
```

```
92
93          plt.plot(pred_unscaled[idx:idx+interval], '--b',
            ↪  label='Prediction')
94          plt.plot(y_unscaled[idx:idx+interval], '-g', label='Target')
95          plt.xlabel('Step')
96          plt.ylabel('FMA - Data')
97          plt.legend()
98
99          svg_path = os.path.join(res_path,
            ↪  f"diff_t_({name})_({idx}).svg")
100         png_path = os.path.join(res_path,
            ↪  f"diff_t_({name})_({idx}).png")
101         fig.savefig(svg_path, bbox_inches='tight')
102         fig.savefig(png_path, dpi=300, bbox_inches='tight')
103
104         #Prediction
105         fig = plt.figure()
106         fig.suptitle(f"Prediction of {name} at index {idx}")
107
108         plt.plot(pred_fma[idx:idx+interval], '--b',
            ↪  label='Prediction')
109         plt.plot(real_fma[idx:idx+interval], '-g',
            ↪  label='Target(FMA)')
110         plt.plot(data_used[idx:idx+interval], '-y', label='Close
            ↪  Price')
111         plt.xlabel('Step')
112         plt.ylabel('Data')
113         plt.legend()
114
115         svg_path = os.path.join(res_path,
            ↪  f"pred_({name})_({idx}).svg")
116         png_path = os.path.join(res_path,
            ↪  f"pred_({name})_({idx}).png")
117         fig.savefig(svg_path, bbox_inches='tight')
118         fig.savefig(png_path, dpi=300, bbox_inches='tight')
119
120     plt.show()
```

Listing 5.2.7: Result generation block of the neural network notebook

6 | Experiment and analysis

On this chapter I will detail all the experiments I performed and the results I gathered. The chapter will be divided in two sections, one for the main forex experiments and another one for the household power consumption experiments that I did in order to verify that there were no implementation issues on the code. Each section will be divided in subsections, one for each experiment. This subsection will be further divided in three parts:

- **Parameters:** List of the parameters used for the experiment
- **Training:** A comment on how the training went with a plot of the loss during training.
- **Results:** A comment on the results gathered and three plots to visualize the experiment results.

For each experiment I generated plots at 5 different indexs but I will only display at index = -5000 here, on the project files you can find the rest of the results under the Results folder inside an experiment folder.

The training of the models was performed using the MeanSquaredError() from keras as the loss function and Adam() from keras as the optimizer. As I'm using Adam, which is an adaptive moment estimation optimizer, I will stick with the recommended lr of 0.001 as changing it has little impact when using an adaptive optimizer. When training the model I set a callback to monitor the validation loss in order to stop when there have been 10 epochs with no improvement, also I set a callback to save the weights with the minimum validation loss. Using this two callbacks I prevent saving an overfitted model and wasting computation time on training a model that is already overfitting. All experiments have a dropout of 0.2 on the last LSTM layer to prevent overfitting.

The name of the experiments makes reference to the parameters being tested on that experiment, so experiment n20_hl1_fma100 mean that I'm will experiment using 20 neurons, 1 hidden layer and fma100 as target. The parameters I experimented with are the number of neurons, the number of hidden layers and the target, I thought that increasing the complexity could yield better results, that's why I tried different

combinations of neurons and hidden layers, as for the target, as I thought that depending on the length of the FMA it could be easier or harder to predict. The rest of the parameters where meant to be experimented after finding a good enough model in order to fine tune it, but as I couldn't get any good model there was no reason to experiment with them.

Each experiment took around 30 to 60 minutes to train, depending on the complexity of the neural network (number of neurons and number of hidden layers). A part from the experiment shown here, I did a lot more training's during the development phase to ensure everything worked as expected. I also did some experiments during the first part of the development when I didn't had a framework for experimentation, and although I didn't save the results they wouldn't contribute anything useful either. The prediction generation for the whole dataset took around 6-12 minutes for each experiment as it has to make 800000+ predictions each time.

In order to speed up the training a strive of 100 is used, that means that if I have 800898 rows as in the forex dataset, I generate 8007 batches with 200 timesteps, and each batch is 100 timesteps apart, so I end up with data the shape of (8007, 200, 1), although this is split in two part for training and validation. The batches have 200 timesteps in hopes of finding really long term relationships if present.

6.1 Forex experiments

For the main forex experiments I tried 8 different configurations, and although I would have liked to do more experimentation, the results I was getting didn't justify exploring further and wasting more time and energy. On the conclusions chapter 7 I will detail some alternatives that could be explored in the hopes of finding better results, unfortunately this alternatives fall out the scope of the project and exploring them could lead to the project being too long and unable to be finished on time.

6.1.1 n20_hl1_fma100

- **Parameters:**
  - ***neurons:*** 20
  - ***hidden_layers:*** 1
  - ***lr:*** 0.001
  - ***input_drop:*** 0
  - ***lstm_out_drop:*** 0.2
  - ***target:*** fma100_diff
- **Training:** As we can see on figure 6.1 the training loss is nearly flat, meaning that the training process was unable to find a set of weights for the neural network that improves the prediction loss over a random set of weights. On the conclusions chapter 7 I will explain why I think this is happening. Furthermore, on the next section 6.2, I detail the experiments I did on another dataset to verify there was no coding or implementation issue.

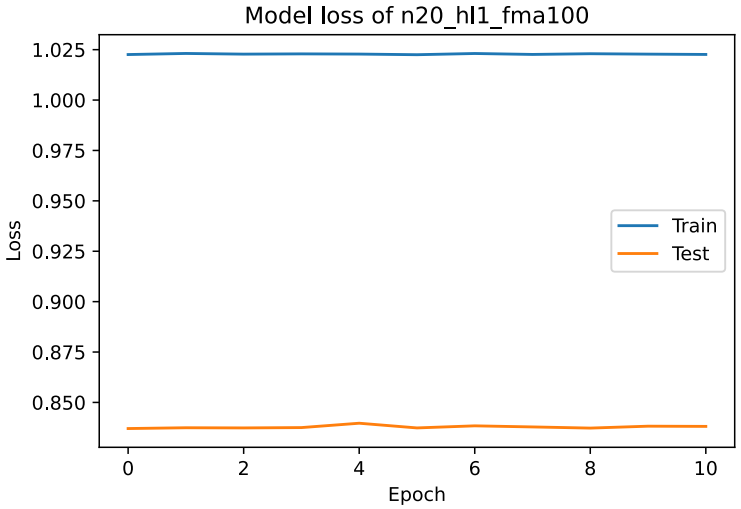


Figure 6.1: Model loss history for experiment n20_hl1_fma100.

- **Results:** On the first plot you can see how the predictions made are really small and although the the shape has some correlation with the shape of the target value, the predictions always stay below zero making the predictions useless. If the predictions always stay below zero for all the dataset that means that no real prediction value is achieved as it's always predicting the FMA to be below the data, and that's not the case, in reality the FMA tends to be 50% of the time above the data and 50% of the time below. With that in mind we see that the predictions will be correct about 50% of the time, which is no better than random guessing. Result plots on Figure 6.2

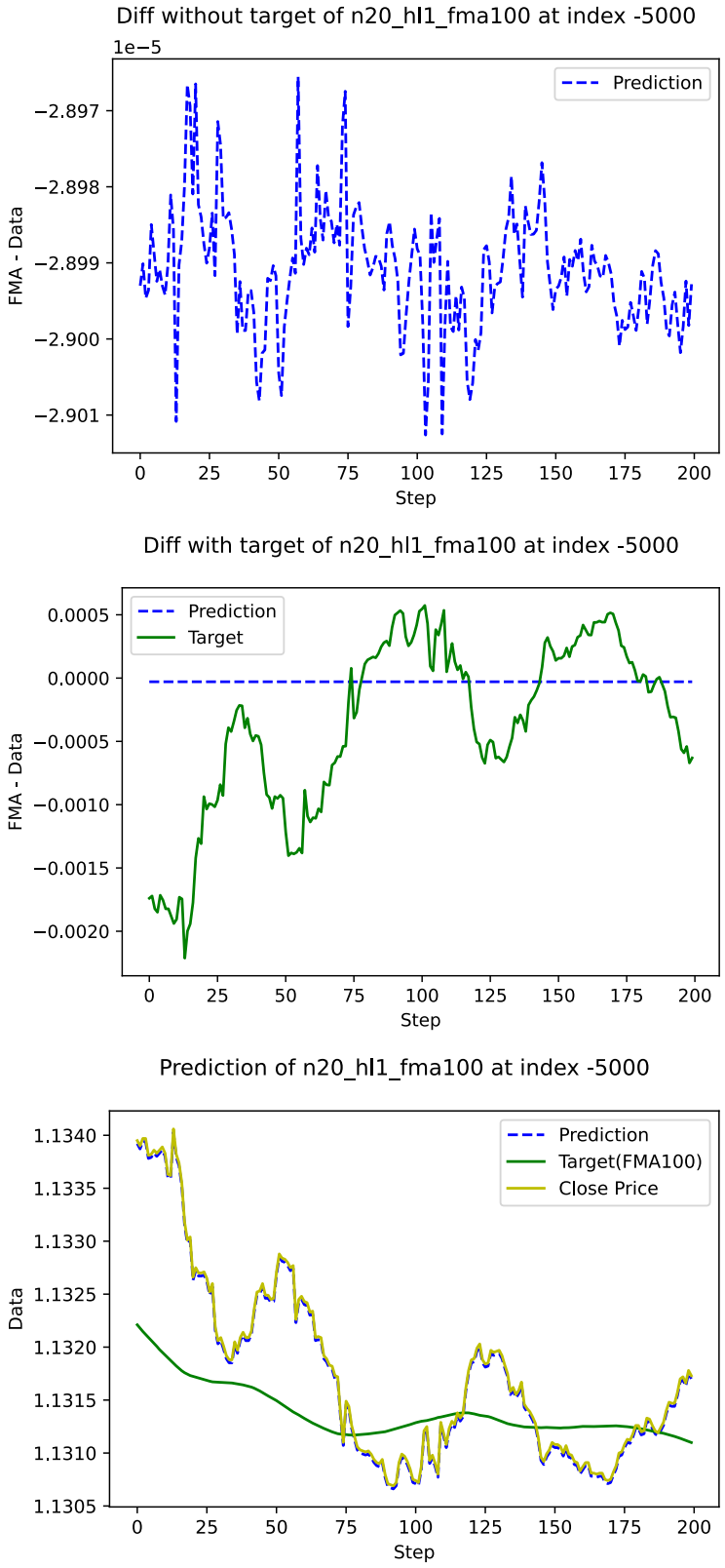


Figure 6.2: 200 adjacent predictions for experiment n20_hl1_fma100, starting at index=-5000.

6.1.2 n20_hl1_fma500

- Parameters:
  - ***neurons:*** 20
  - ***hidden_layers:*** 1
  - ***lr:*** 0.001
  - ***input_drop:*** 0
  - ***lstm_out_drop:*** 0.2
  - ***target:*** fma500_diff
- **Training:** No improvements over the previous experiment. Loss history on Figure 6.3.

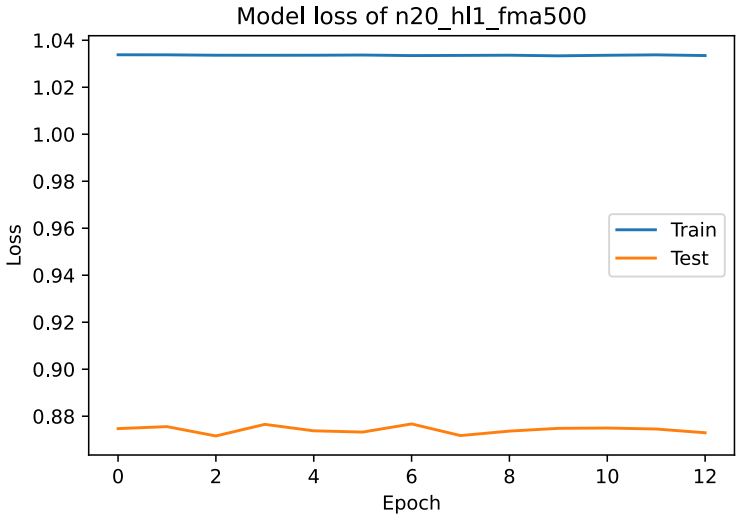


Figure 6.3: Model loss history for experiment n20_hl1_fma500.

- **Results:** No improvements over the previous experiment. Result plots on Figure 6.4.

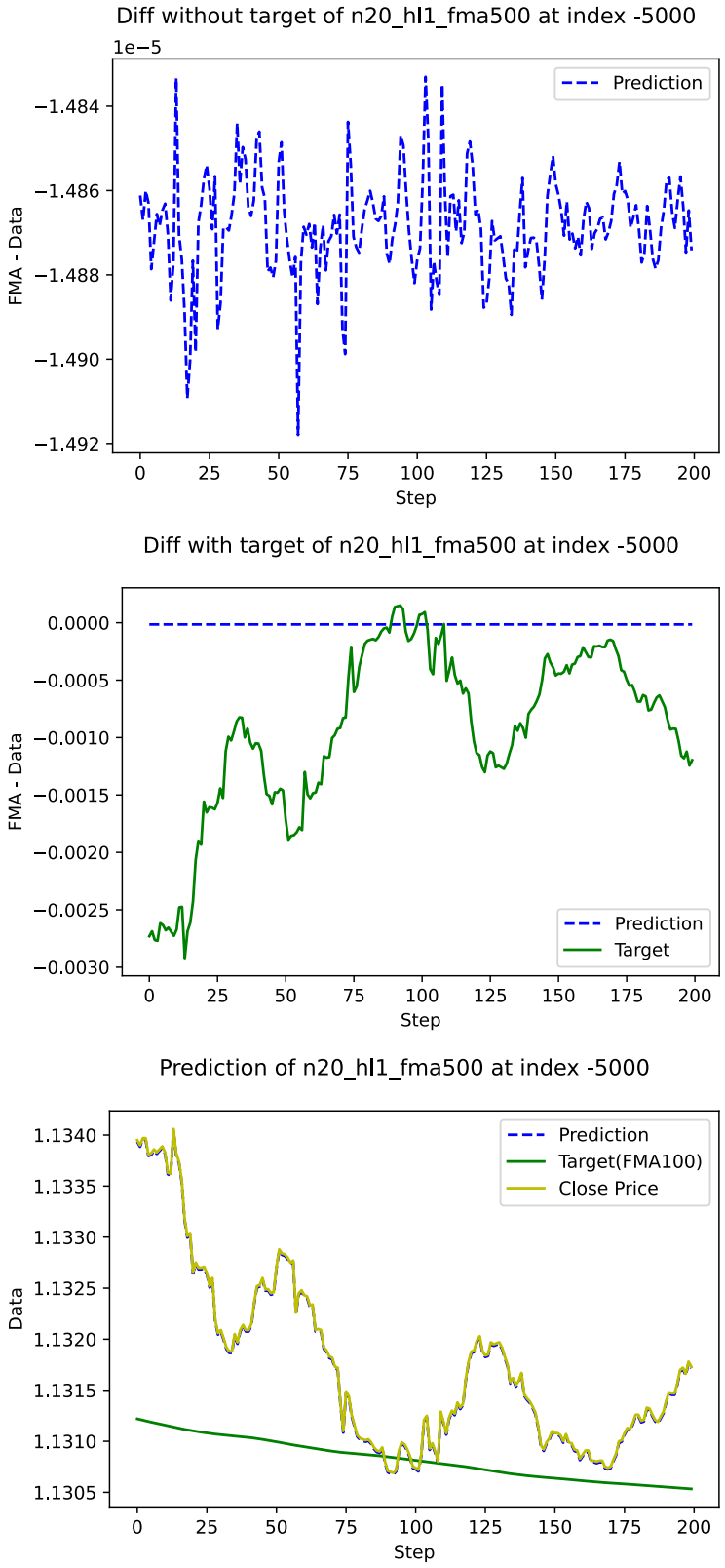


Figure 6.4: 200 adjacent predictions for experiment n20_hl1_fma500, starting at index=-5000.

6.1.3 n20_hl2_fma100

- Parameters:
  - ***neurons:*** 20
  - ***hidden_layers:*** 2
  - ***lr:*** 0.001
  - ***input_drop:*** 0
  - ***lstm_out_drop:*** 0.2
  - ***target:*** fma100_diff
- **Training:** No improvements over the previous experiments. Loss history on Figure 6.5.

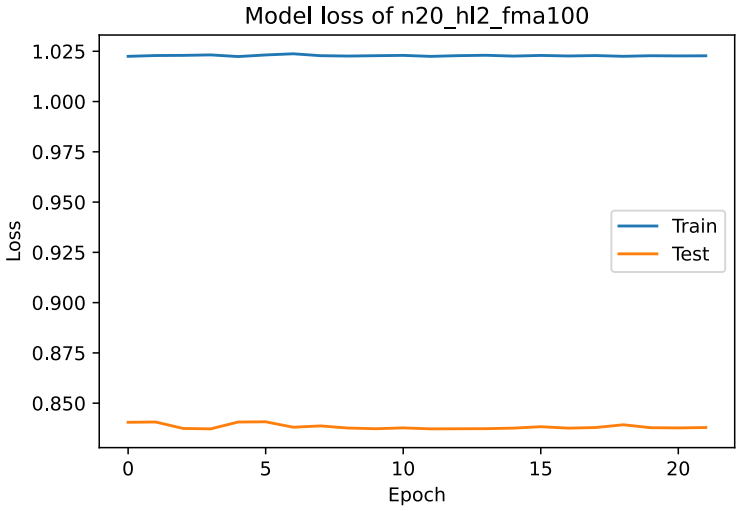


Figure 6.5: Model loss history for experiment n20_hl2_fma100.

- **Results:** No improvements over the previous experiments. Result plots on Figure 6.6.

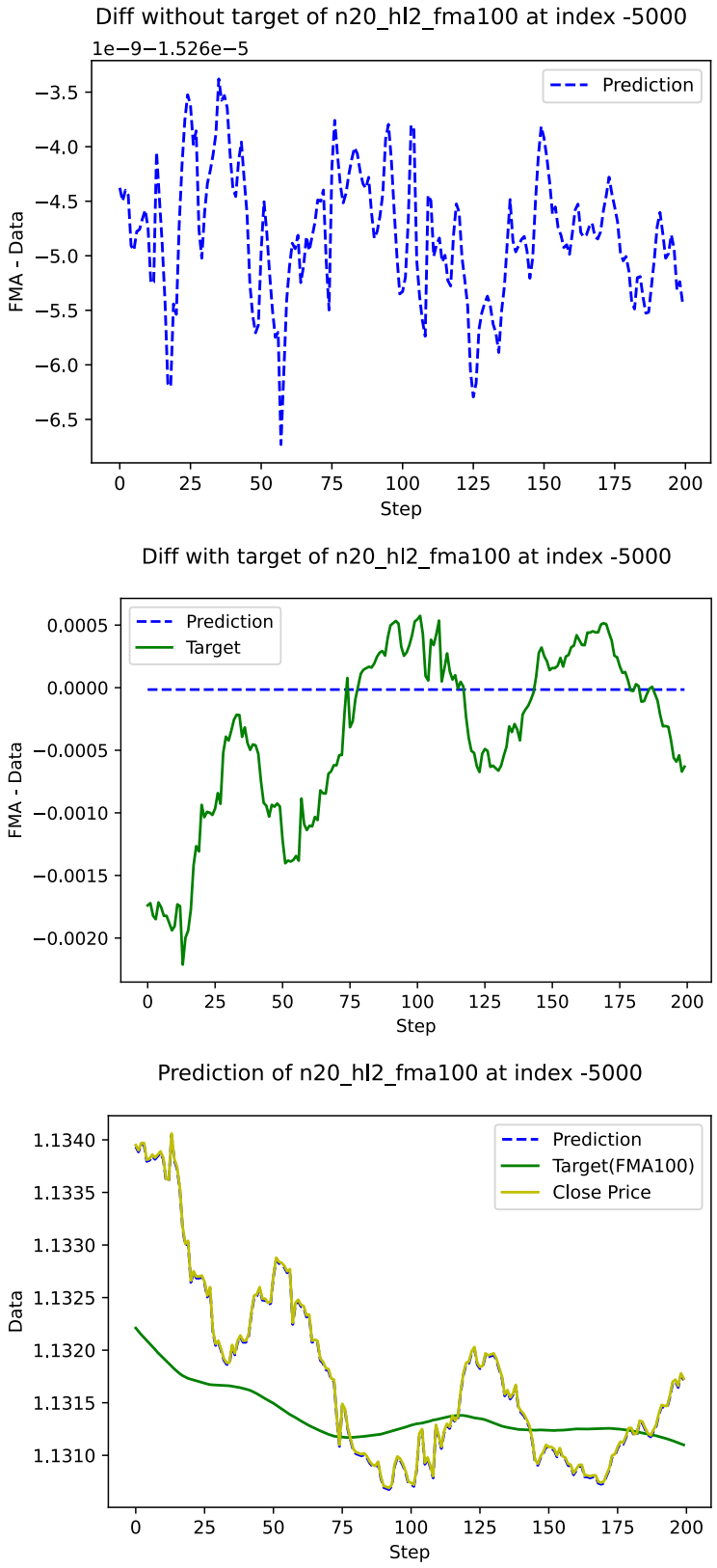


Figure 6.6: 200 adjacent predictions for experiment n20_hl2_fma100, starting at index=-5000.

6.1.4 n20_hl2_fma500

- Parameters:
  - ***neurons:*** 20
  - ***hidden_layers:*** 2
  - ***lr:*** 0.001
  - ***input_drop:*** 0
  - ***lstm_out_drop:*** 0.2
  - ***target:*** fma500_diff
- **Training:** No improvements over the previous experiments. Loss history on Figure 6.7

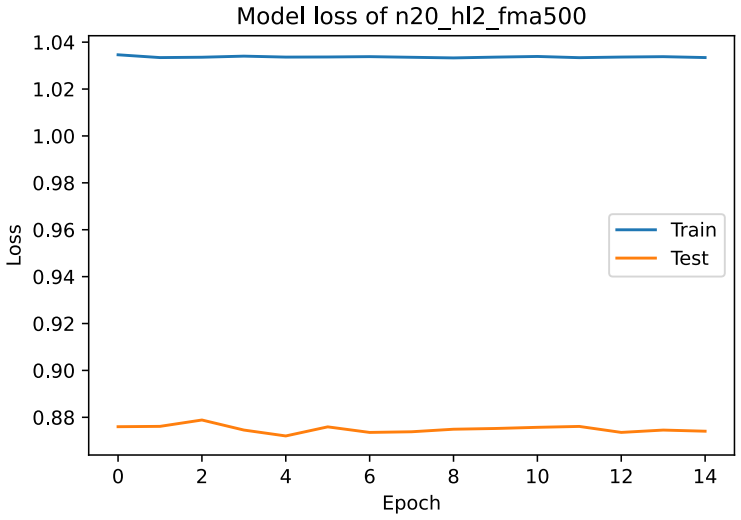


Figure 6.7: Model loss history for experiment n20_hl2_fma500.

- **Results:** No improvements over the previous experiments. Result plots on Figure 6.8

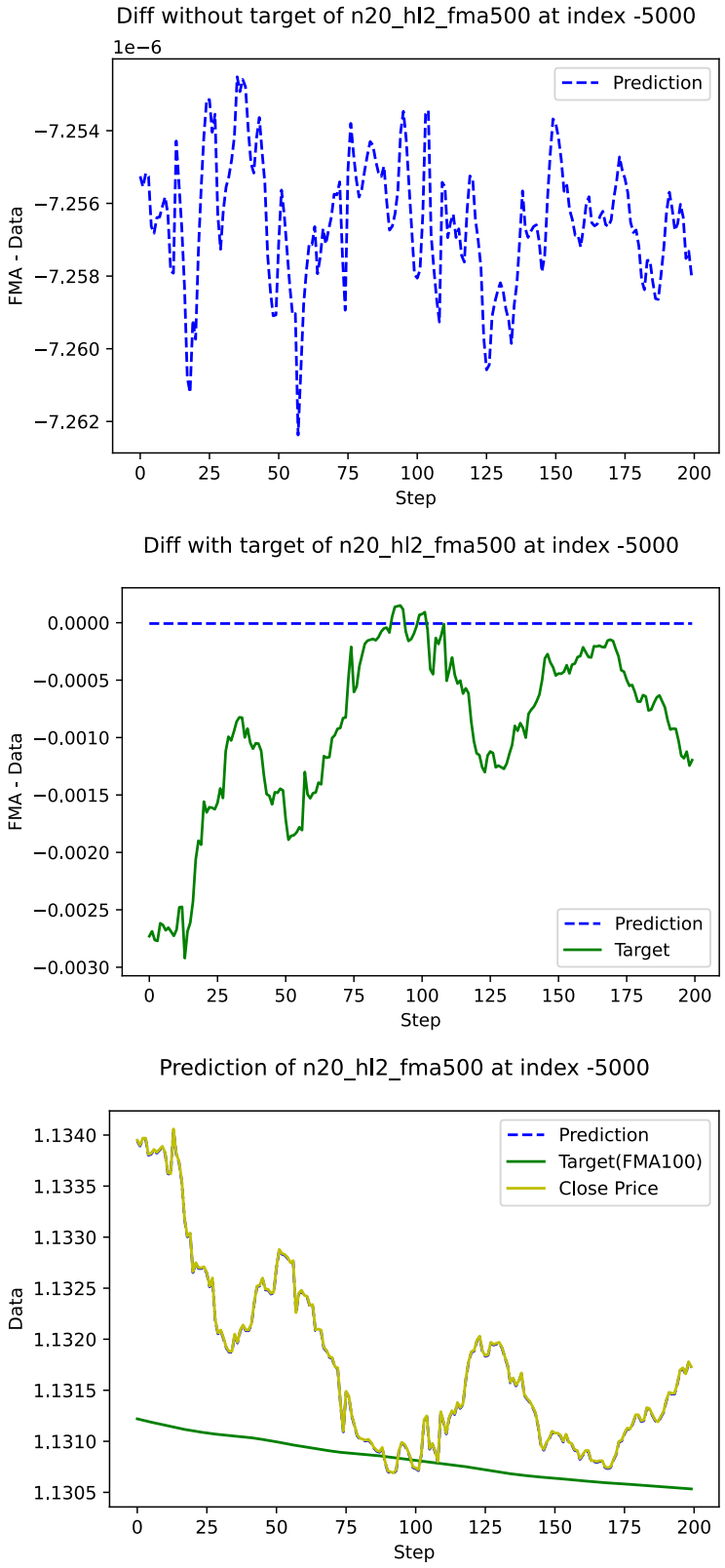


Figure 6.8: 200 adjacent predictions for experiment n20_hl2_fma500, starting at index=-5000.

6.1.5 n100_hl1_fma100

- Parameters:
  - ***neurons:*** 100
  - ***hidden_layers:*** 1
  - ***lr:*** 0.001
  - ***input_drop:*** 0
  - ***lstm_out_drop:*** 0.2
  - ***target:*** fma100_diff
- **Training:** No improvements over the previous experiments. Loss history on Figure 6.9

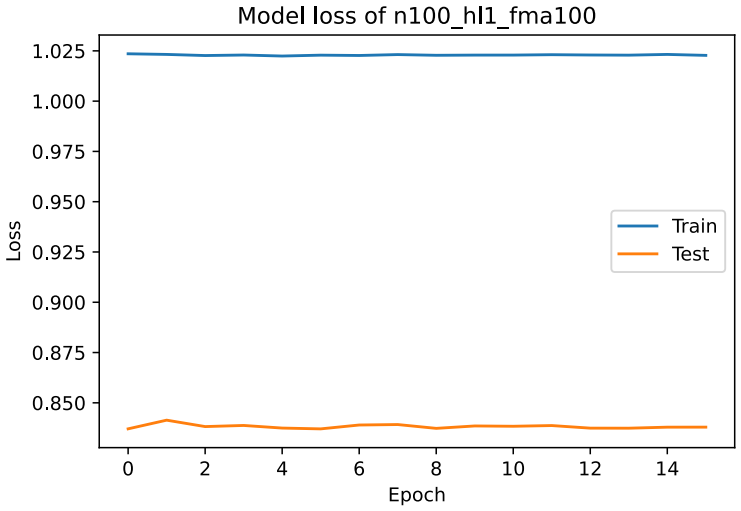


Figure 6.9: Model loss history for experiment n100_hl1_fma100.

- **Results:** No improvements over the previous experiments. Result plots on Figure 6.10

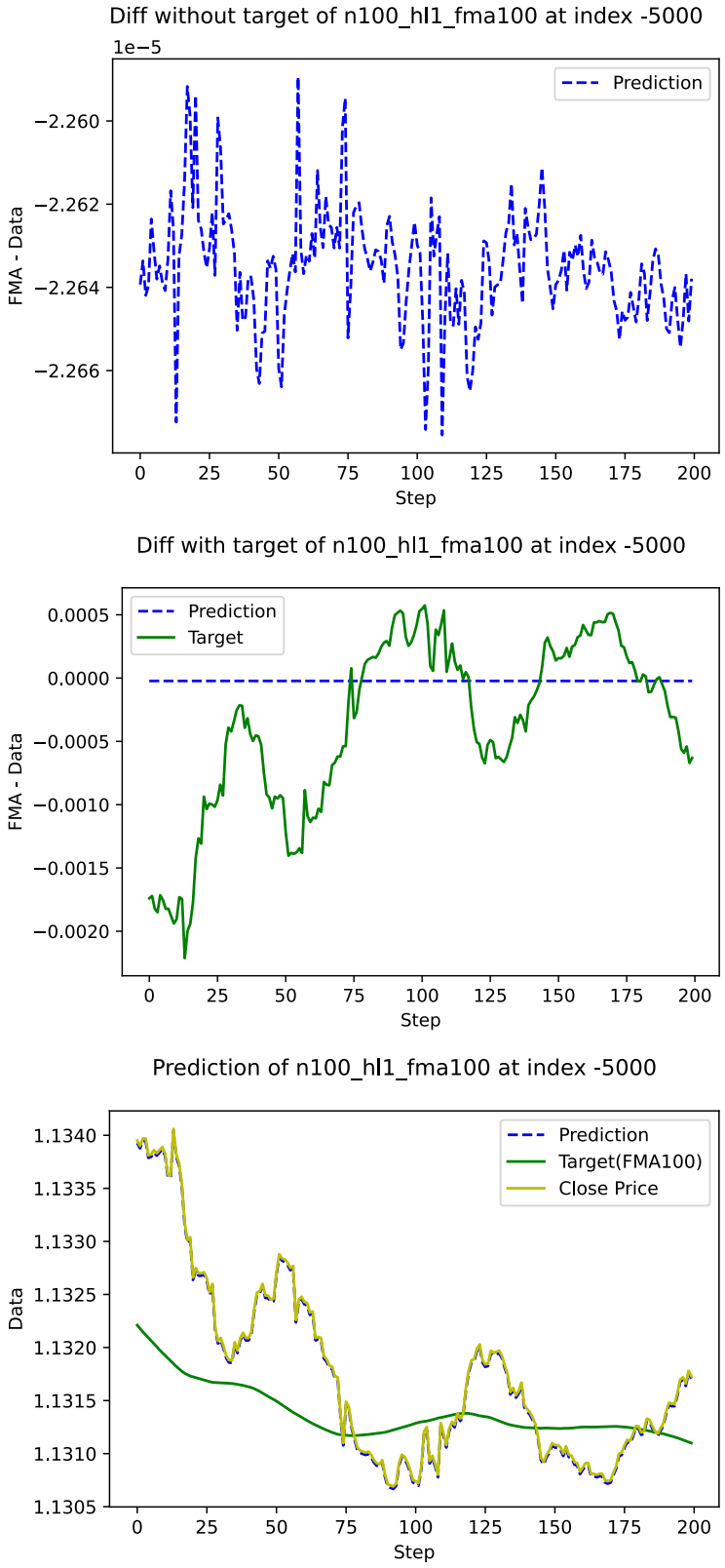


Figure 6.10: 200 adjacent predictions for experiment n100_hl1_fma100, starting at index=-5000.

6.1.6 n100_hl1_fma500

- Parameters:
  - ***neurons:*** 100
  - ***hidden_layers:*** 1
  - ***lr:*** 0.001
  - ***input_drop:*** 0
  - ***lstm_out_drop:*** 0.2
  - ***target:*** fma500_diff
- **Training:** No improvements over the previous experiments. Loss history on Figure 6.11

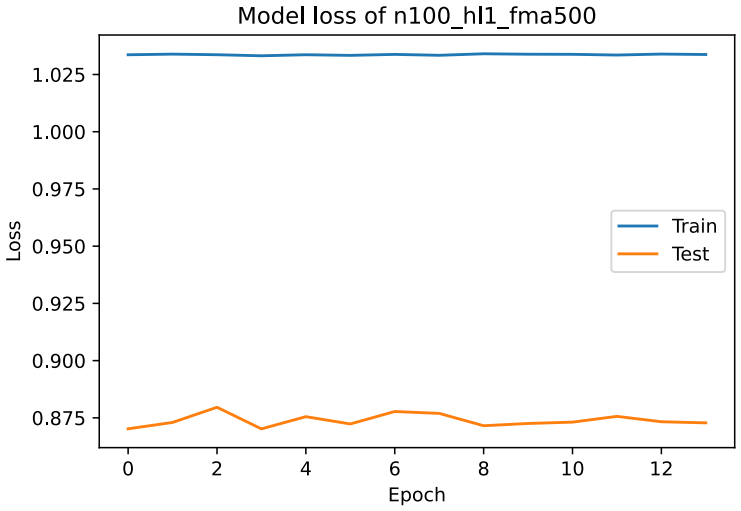


Figure 6.11: Model loss history for experiment n100_hl1_fma500.

- **Results:** No improvements over the previous experiments. Result plots on Figure 6.12

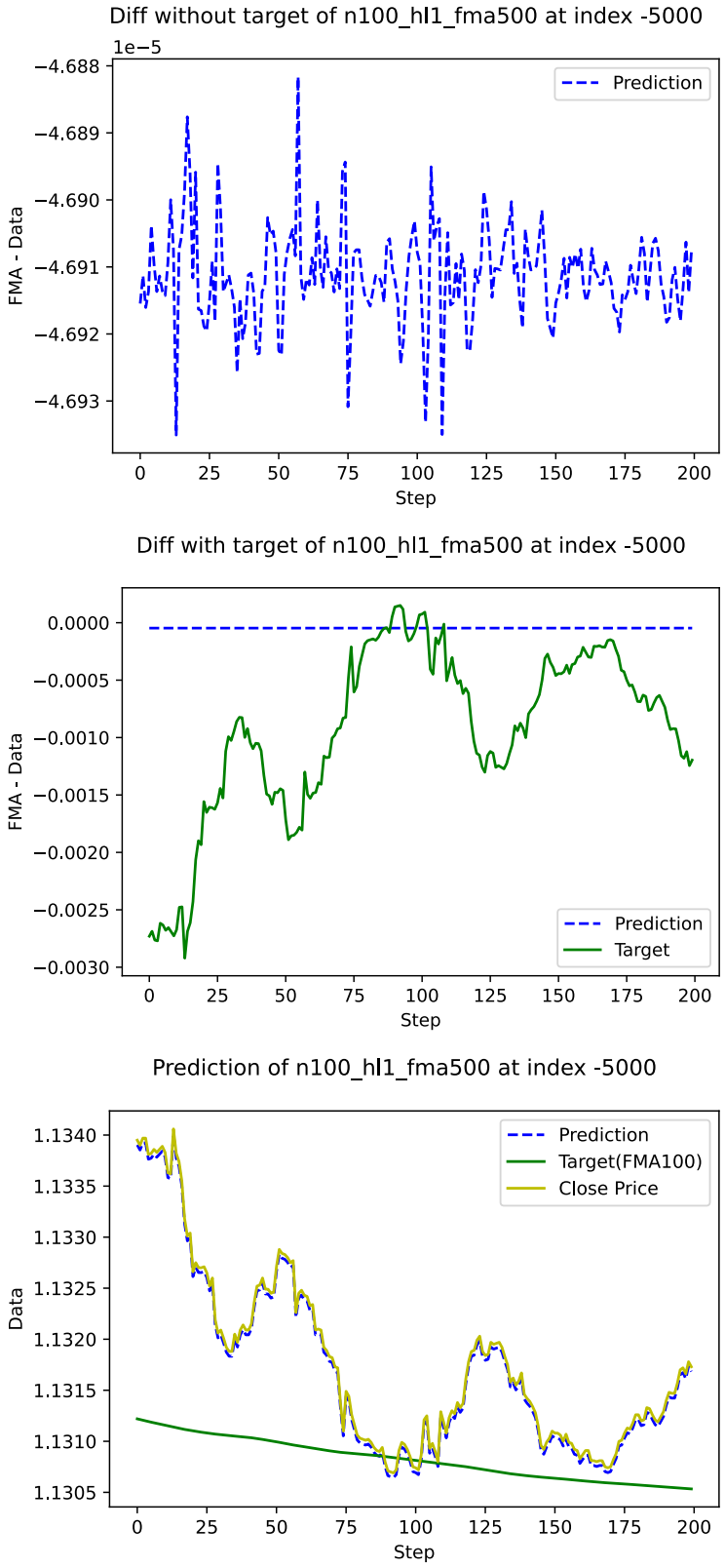


Figure 6.12: 200 adjacent predictions for experiment n100_hl1_fma500, starting at index=-5000.

6.1.7 n100_hl2_fma100

- Parameters:
  - ***neurons:*** 100
  - ***hidden_layers:*** 2
  - ***lr:*** 0.001
  - ***input_drop:*** 0
  - ***lstm_out_drop:*** 0.2
  - ***target:*** fma100_diff
- **Training:** No improvements over the previous experiments. Loss history on Figure 6.13

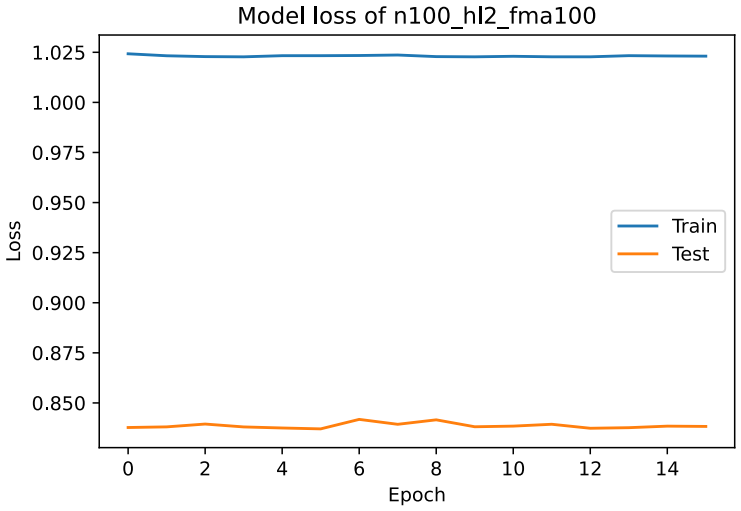


Figure 6.13: Model loss history for experiment n100_hl2_fma100.

- **Results:** No improvements over the previous experiments. Result plots on Figure 6.14

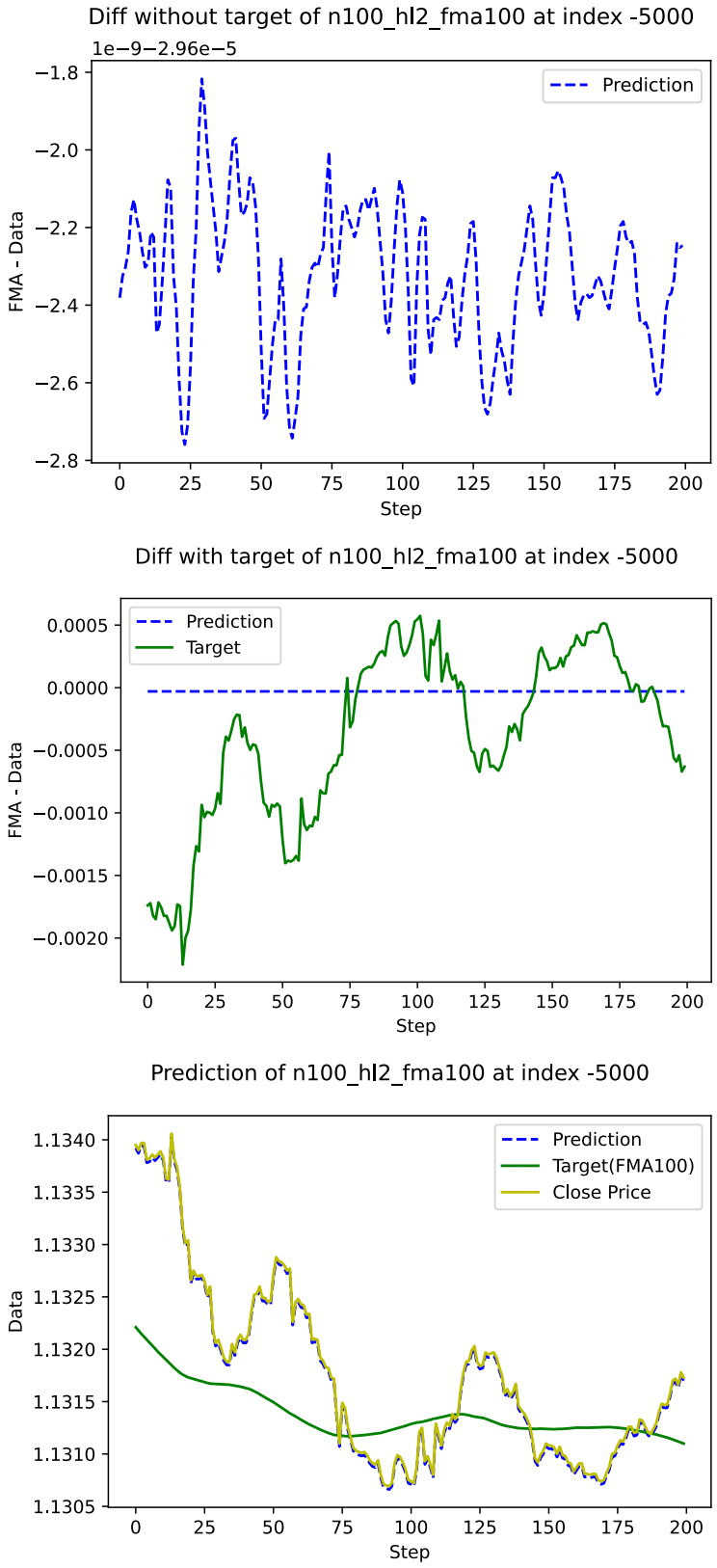


Figure 6.14: 200 adjacent predictions for experiment n100_hl2_fma100, starting at index=-5000.

6.1.8 n100_hl2_fma500

- Parameters:
  - ***neurons:*** 100
  - ***hidden_layers:*** 2
  - ***lr:*** 0.001
  - ***input_drop:*** 0
  - ***lstm_out_drop:*** 0.2
  - ***target:*** fma500_diff
- **Training:** No improvements over the previous experiments. Loss history on Figure 6.15

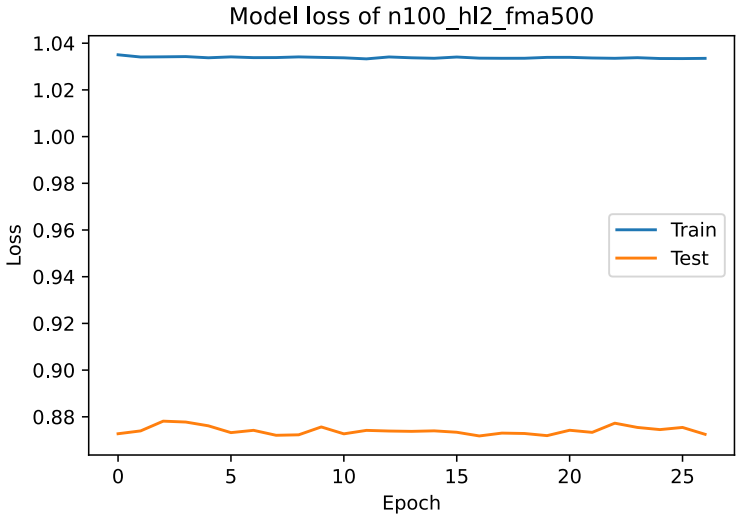


Figure 6.15: Model loss history for experiment n100_hl2_fma500.

- **Results:** No improvements over the previous experiments. Result plots on Figure 6.16

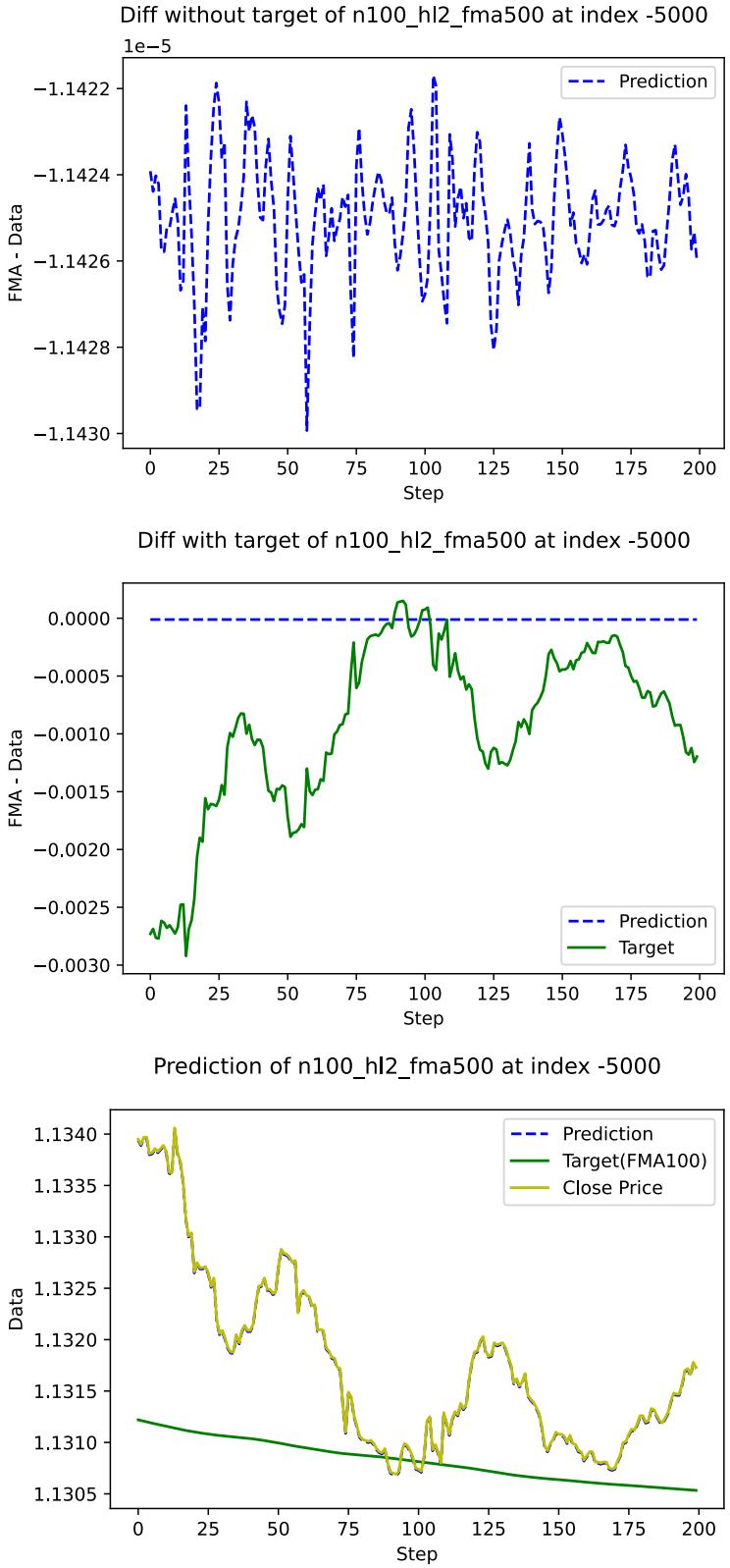


Figure 6.16: 200 adjacent predictions for experiment n100_hl2_fma500, starting at index=-5000.

6.2 Household power consumption experiments

6.2.1 n20_hl1_fma100_h

- **Parameters:**
  - ***neurons:*** 20
  - ***hidden_layers:*** 1
  - ***lr:*** 0.001
  - ***input_drop:*** 0
  - ***lstm_out_drop:*** 0.2
  - ***target:*** fma100_diff
- **Training:** As we can see on figure 6.17 the training loss is not flat as in the forex experiments, meaning that there are no implementation issues and that this implementation and architecture is able to train given the appropriate data. Here the loss decreases in the usual curve we see when training a neural network and stagnates at around 0.65 with a minimum of 0.65743 in validation loss and 0.59294 loss at the same epoch.

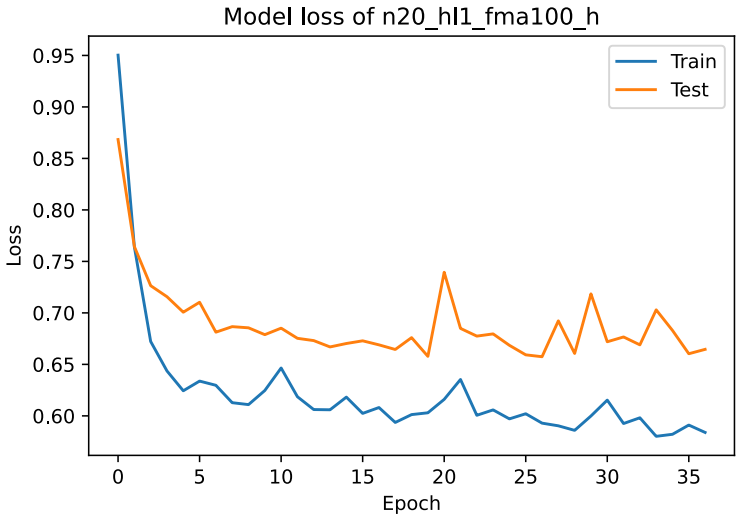


Figure 6.17: Model loss history for experiment n20_hl1_fma100_h.

- **Results:** Here the results are meaningful and give us information on whether the FMA will be above or below the current value, and although the prediction line is not able to follow the Target(FMA) perfectly, is good enough to make useful predictions about the future. On the last plot you can clearly see that when the FMA is above the data line the prediction is also above, and when the FMA is below the data, so is the prediction. At around step 75 and 125 you can see how it turns around correctly. Plots at 6.18

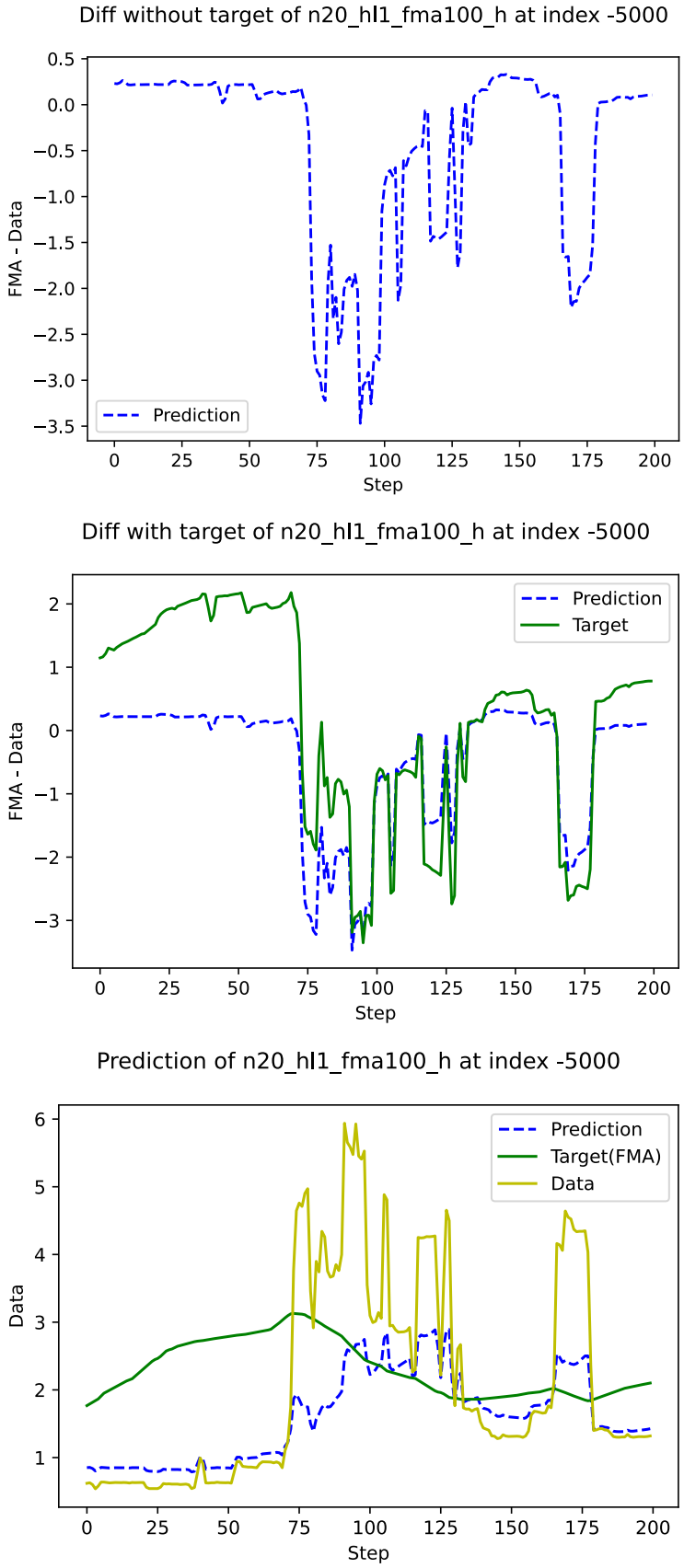


Figure 6.18: 200 adjacent predictions for experiment n20_hl1_fma100_h, starting at index=-5000.

6.2.2 n100_hl2_fma100_h

- Parameters:
  - ***neurons:*** 100
  - ***hidden_layers:*** 2
  - ***lr:*** 0.001
  - ***input_drop:*** 0
  - ***lstm_out_drop:*** 0.2
  - ***target:*** fma100_diff
- **Training:** As I increased the neurons and added a hidden layer the model overffits faster but it also produces a model with a lower validation loss of 0.63946 and a loss of 0.58767. Plot at 6.19

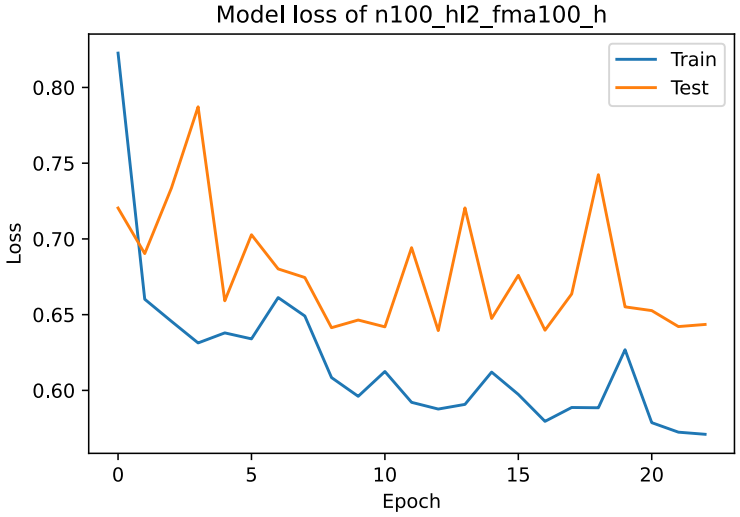


Figure 6.19: Model loss history for experiment n100_hl2_fma100_h.

- **Results:** The results are very similar to the previous experiment. Plots at 6.20

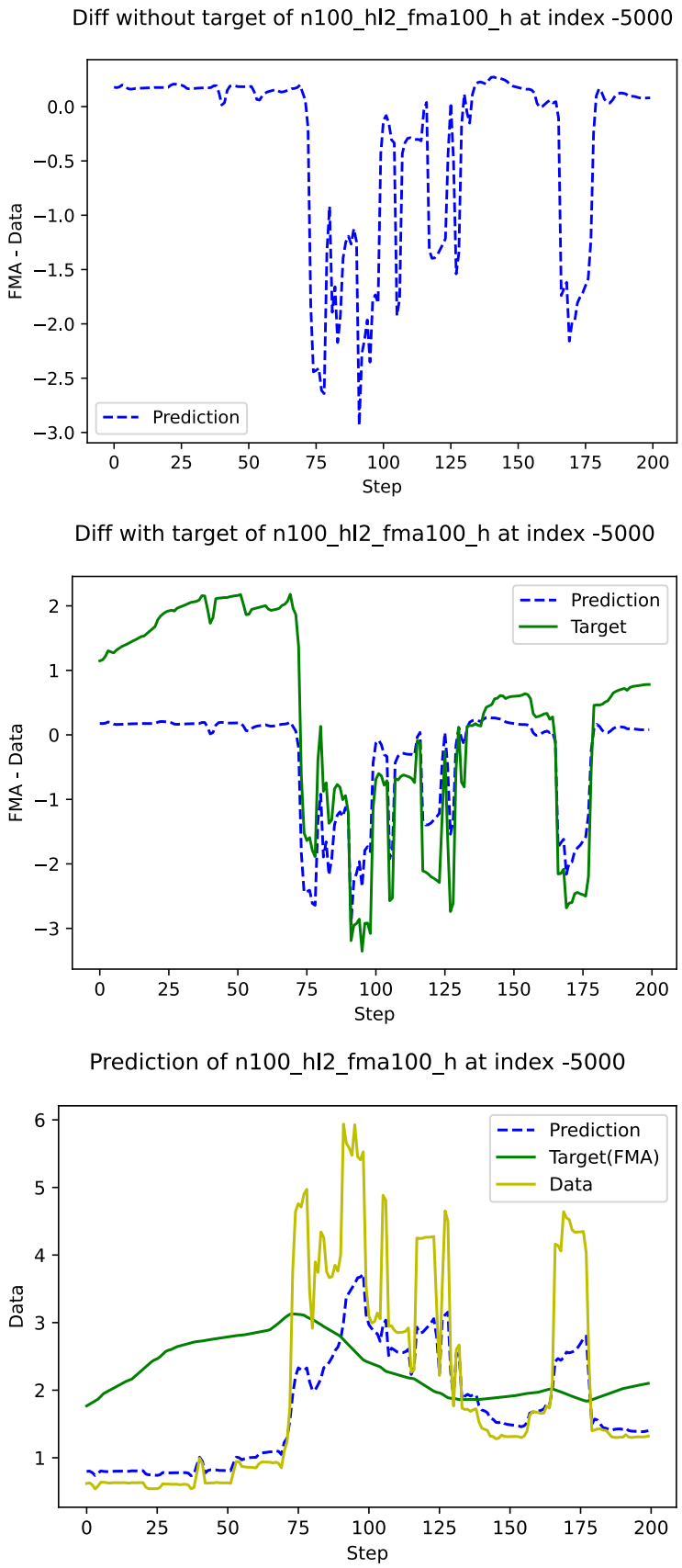


Figure 6.20: 200 adjacent predictions for experiment n100_hl2_fma100_h, starting at index=-5000.

7 | Conclusions

In this last chapter I will go over everything I did on this project and the results I gathered. I will also provide my view on different alternatives that could be explored in the hopes of finding better results.

7.1 Project Development

The development of the project was carried without any major obstacles, a part from the fact that I was unable to dedicate the time needed during the first part of the project forcing me to choose the April defense. Although this wasn't because of bad planing or the project itself, but rather because of personal issues, the major one being that I started working for a startup. A part from that, I dint find problems that hindered the project in any major way.

The research was carried nicely and took the time estimated.

The development took a little bit more than expected as I also developed the experimentation framework that I didn't had in mind when starting the project. Also the testing and debugging part took longer than expected.

The experimentation part took less than expected as the model wasn't generating the expected results, and with the findings I will describe on next section 7.2, there was no reason to continue investing resources on more experimentation.

And finally the documentation was carried as expected, and even though there was some learning involved, as I never used latex before on this level, I have to say I'm very satisfied with how the final document looks, and I feel as though learning to use latex properly was a very good investment.

7.2 Project Goal

The goal of this project was to generate a model using neural networks capable of predicting the forex market on a low timeframe (1 minute

bars). This could be further divided in two sub-objectives, the first would be developing a functional neural network, and the second one would be using said neural network to predict the forex market, I will go over both sub-objectives on the following subsections.

7.2.1 Functional neural network

The first part of the goal was accomplished successfully, I developed a functional neural network for timeseries prediction that could successfully train given the appropriate data. I learned a lot about neural networks and their implementation using python during this project, and even though the model couldn't provide the results I wanted I feel as though the learning was a success on its own.

7.2.2 Forex prediction model

Unfortunately I was unable to succeed in the second part of the goal, the model was unable to generate any useful prediction at all, but from my analysis I found that it was not because the model was not good enough or that I needed more data, it had nothing to do with that, the most likely reason behind the model being unable to generate useful prediction is due to the nature of the data. I found that the forex movements on low timeframes resemble the movements of a random walk, that is, a timeseries where each new step has no correlation whatsoever with previous steps, or what is the same, it's a temporal analogue for a coin toss where if you land heads you draw a line going up and if you land tails you draw a line down, meaning each new step is entirely random, there's always and equal chance of the value going up or down.

On Figures 7.1,7.2,7.3 you can see three comparisons between a random walk chart and a forex chart, Forex charts are taken from the EUR/USD pair with 1 minute bars. As you can see all charts look very similar, you couldn't tell if it was random walk or forex just by looking at the chart, also each comparison shows one of the three different states of a market, upward trend, downward trend and ranging, so what looks like trends can also be a product of randomness. If we tried to train any model with enough data from a random walk we would end up up with the same results as in the forex experiments, a training that is unable to improve because any modification to the weights may be good for some parts but equally bad for some others, there is no room for improvement training on random data. To generate the random walk data I used a free plugin for Metatrader 5 called Random Walk Chart[16].

Although I think the forex market on low timeframes is not entirely random, the imperfections that could provide and edge over a 50%-50% prediction are so minuscule that finding them constantly may be nearly impossible, and it would probably require a very specialized model. And

Figure 7.1: Upward trend chart comparison between a random walk chart and a EUR/USD forex chart. Random chart above, forex chart below. [Own screenshot]

even with a very specialized model that could find this imperfections, most of the time the movements would be random and no profitable trading could be done. It would be interesting to develop a model to try and find this momentary imperfections if any and quantify the rate of manifestation.

Figure 7.2: Downward trend chart comparison between a random walk chart and a EUR/USD forex chart. Random chart above, forex chart below. [Own screenshot]

Figure 7.3: Ranging chart comparison between a random walk chart and a EUR/USD forex chart. Random chart above, forex chart below. [Own screenshot]

7.3 Alternatives

Here I will describe some alternatives that could be explored in the hopes of finding a way to predict the market.

7.3.1 Higher timeframes

As said before, this project was aimed at generating a model that could predict the EUR/USD forex pair on a low timeframe (1 minute bars), one alternative would be using higher timeframes such as 1H to 4H bars, that way the data used would have less noise coming from the sheer amount of transactions that happen on the forex market. Using higher timeframes could lead to finding macroeconomic patterns that manifest on the long run.

7.3.2 Different market

I decided to use the forex market because it has lot of liquidity and the volume per day is huge, meaning that there are very low spreads and that very big orders can be filled easily, and that's very important if you want to develop a trading strategy that can scale on the amount traded. But there are also other markets, and although the may not be as liquid as the forex market, it could be far easier to find patterns as the numbers of participants is smaller and the patterns that emerge from human behaviour have more weight and prevalence.

The next market to try that comes to my mind would be the stock market, the place where company stocks are traded, a very known and regulated market used worldwide in daytrading and swingtrading strategies.

There is also the cryptocurrency market, although I would be very careful exploring the cryptocurrency market as its very unregulated and has a lot of huge fluctuations, that means there is a lot of risk but also a lot of potential profit.

8 | Technical skills

8.1 CCO1.3

Define, evaluate and select hardware and software development and production platforms for the development of computer applications and services of varying complexity. [Quite]

The project involved studding available options for the development of the the preproccesor and neural network, such as the libraries that where going to be employed or the IDE that was going to be used.

8.2 CCO2.1

Demonstrate knowledge of the fundamentals, paradigms and techniques of intelligent systems, and analyze, design and build computer systems, services and applications that use these techniques in any field of application. [Quite]

The project involved developing prediction models and analyzing them, it also involved developing an intelligent experimentation framework.

8.3 CCO2.2

Ability to acquire, obtain, formalize and represent human knowledge in a computable way for solving problems through a computer system in any field of application, particularly in those related to aspects of computing, perception and performance in smart environments or environments. [Quite]

The project involved using the knowledge I have in programming, data analysis and prediction model generation to develop a whole system for experimenting with prediction models and analyze its results.

8.4 CCO2.4

Demonstrate knowledge and develop computer learning techniques; design and implement applications and systems that use them, including those dedicated to the automatic extraction of information and knowledge from large volumes of data. [In depth]

On this project I developed neural networks that were trained using timeseries data with a huge dataset of more than 800000 rows, this neural networks aimed at extracting valuable future information about the forex market.

Bibliography

[1] *Anaconda*. URL: https://www.anaconda.com/ (visited on 10/29/2021).

[2] *Artificial neural network*. URL: https://en.wikipedia.org/wiki/Artificial_neural_network (visited on 10/18/2021).

[3] *Candlestick*. URL: https://www.definedge.com/candlestick-patterns/ (visited on 10/18/2021).

[4] *EA Trading Academy data*. URL: https://eatradingacademy.com/software/forex-historical-data/ (visited on 10/25/2021).

[5] *Forex average daily turnover*. URL: https://www.statista.com/statistics/1204111/euro-activity-trading-day-global-currency-market/ (visited on 09/27/2021).

[6] *Glassdoor*. URL: https://www.glassdoor.es/index.html (visited on 10/11/2021).

[7] *Google colab*. URL: https://colab.research.google.com/ (visited on 10/29/2021).

[8] *HistData.com*. URL: https://histdata.com/ (visited on 10/25/2021).

[9] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735. URL: https://www.researchgate.net/profile/Sepp-Hochreiter/publication/13853244_Long_Short-term_Memory/links/5700e75608aea6b7746a0624/Long-Short-term-Memory.pdf.

[10] *Keras*. URL: https://keras.io/ (visited on 10/27/2021).

[11] *LSTM Keras*. URL: https://keras.io/api/layers/recurrent_layers/lstm/ (visited on 10/27/2021).

[12] *LSTM Wikipedia*. URL: https://en.wikipedia.org/wiki/Long_short-term_memory (visited on 10/27/2021).

[13] *Metatrader 5*. URL: https://www.metatrader5.com/es (visited on 10/25/2021).

[14] *Python*. URL: https://www.python.org/ (visited on 10/27/2021).

[15] *Pytorch*. URL: https://pytorch.org/ (visited on 10/27/2021).

[16] *Random walk generator*. URL: https://www.mql5.com/es/market/product/42166 (visited on 10/17/2021).

[17] *Tensorflow*. URL: https://www.tensorflow.org/ (visited on 10/27/2021).

[18] *Trello*. URL: https://trello.com/es (visited on 10/18/2021).

[19] *Visual Study Code*. URL: https://code.visualstudio.com/ (visited on 10/29/2021).