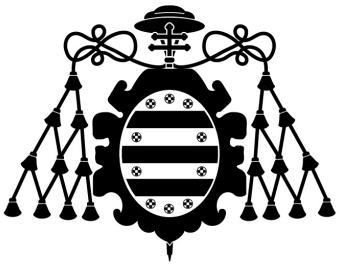


Universidad de Oviedo

School of Computer Science Engineering

Degree Dissertation

TensorFlow and PyTorch: Analysis and application to the stock market

Author

Guillermo Arce Poyal

Supervisor

Juan Luis Mateo Cerdán, PhD

Oviedo
June 15, 2021

Abstract

This project provides a developer point of view analysis of two of the most popular ML libraries nowadays: TensorFlow and PyTorch. For such a purpose, both are applied into financial data with the intention of building several models that can face the complexity of the market when making minute price predictions. Moreover, a web application prototype that simulates a real-time financial prediction system is designed and implemented.

Keywords

TensorFlow, PyTorch, deep learning, neural networks, LSTM, stock market, web application prototype.

Acknowledgment

I would like to recognize and thank the support done by my supervisor Juan Luis Mateo Cerdán, PhD, for his knowledge, flexibility and availability, as well as for helping me stay aligned with the objectives of the project.

Moreover, I would like to thank my family, my parents and my sister, who are always there and encourage me to go forward.

Finally, I would also like to thank my girlfriend, Andrea, and her family, for their enthusiastic support during these last four years.

Declaration of originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, except where due acknowledgement is made. Therefore, I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged.

Información básica

- Título: "TensorFlow y PyTorch: análisis y aplicación al mercado de valores".
- Resumen: El presente proyecto proporciona un análisis desde el punto de vista del desarrollador de dos de las librerías de aprendizaje automático más populares en la actualidad: TensorFlow y PyTorch. Para tal fin, ambas se aplican a datos financieros con la intención de construir varios modelos que puedan enfrentarse a la complejidad del mercado a la hora de realizar predicciones en el precio. Además, se diseña e implementa un prototipo de aplicación web que simula un sistema de predicción financiera en tiempo real.
- Palabras clave: TensorFlow, PyTorch, aprendizaje profundo, redes neuronales, LSTM, mercado de valores, prototipo de aplicación web.
- Resumen de las conclusiones: TensorFlow destaca por su inmensidad y su habilidad para crear modelos con una sintaxis sencilla e intuitiva. Por otro lado, PyTorch proporciona un servicio con una mayor curva de aprendizaje inicial pero con una flexibilidad que captura a la mayoría de los investigadores. En relación al rendimiento de los modelos, se han alcanzado resultados con más de un 70% de acierto en la dirección de las predicciones del precio (para predicciones de un minuto). De todas formas, todos los modelos encuentran dificultades a la hora de enfrentarse a cambios muy bruscos en el precio.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

1.1 Project motivation

Artificial intelligence is nowadays used everywhere: healthcare, social media, e-commerce, gaming, etc. It is no more something of the future, but from the present. From gaining insight about customer behaviour and preference trends, to automatic identification of cancer cells; the contributions that a good use of AI can bring are very meaningful. That is why, computer science community is getting more interest in this topic.

When not just some innovative companies but the whole community are interested in something, it is when the knowledge starts to spread and new open-source tools begin to be developed. In this case, Google Brain and Facebook AI teams built TensorFlow and PyTorch, respectively. These libraries provide high-level APIs that help developers build their own models. In this way, the developer does not waste time in low level software engineering and mathematics issues but rather on model configuration. That is, these tools provide us with a door to enter the world of artificial intelligence in a much more efficient and intuitive way.

Apache MXNet, Theano, Scikit-learn, etc. are all libraries that also have a niche in the market. However, at the moment of this project, TensorFlow and PyTorch are the two most popular and the ones that Google and Facebook use, respectively. Due to this popularity, to the widespread use in the industry, and to the interest in learning about them, both libraries were chosen for the development of this project.

There is a lot of information about them in the Internet, however, most of it can be confusing for a user without experience in the field. That is why, in this project we are analysing it from the developer point of view. "What can I do with this library?", "Is this library good for my use case?", "I do not have much knowledge about ML, which library is better for me?", "Which library is better for production uses? And for research?"; all these questions and more may arise when the developer starts to immerse in ML libraries. With this project, we may help users who are struggling taking the decision of which library to use (between TensorFlow and PyTorch), as well as users that want to learn something about them.

In order to apply these two libraries, financial data will be used. When using financial data to make predictions in the market, we are taking part of a new industry that is recently emerging: Fintech. As one of the most popular AI applications nowadays, by improving and automatizing financial activities and decisions, it is gaining interest not only for companies but for researchers. Given the extremely complex nature of the market, it arises very challenging questions that are worth the effort of deep investigations. Even not the case of this project, it was considered an interesting field for applying TensorFlow and PyTorch and see what models we can achieve with the resources and knowledge at our hand.

Also, with some of the models obtained, a web application prototype will be developed. The idea is to create a prototype that shows the user the ability of the models and how can they be used in real life applications. In our particular case, we will be developing a simulation of a real time system for financial predictions.

1.2 Objective

The objectives of the project are the following ones:

- Apply and analyse (from the developer point of view) TensorFlow and PyTorch.
- Get, at least, one model from each of the libraries that can be applied to financial data with reasonable results.
- Build and deploy a web application prototype that makes use of some of the models obtained. Concretely, a real time simulation financial prediction system.

1.3 Current situation

Due to the exploratory nature of the project, it may be difficult to compare it to others. As we are not developing something to compete in the market but making a study that combines several objectives, finding a project with the same ambitions can be hard. However, in relation to the web prototype and to the models application field, we can find some potential competitors:

- FinBrain: It is a company that provides financial predictions from AI technologies. In addition to direct market predictions, they provide an API that can be used to create own trading bots from their results.
  - Website: https://finbrain.tech/
- StocksNeural: This firm supply deep learning predictions for stock prices and their trends. They use TensorFlow and RNNs and CNNs in order to make the predictions.
  - Website: https://stocksneural.net/

2 General considerations

For the sake of a better understanding of the project, the following concepts may be considered:

- Neural networks: *Computing systems developed to simulate the human nervous system for machine learning tasks by treating the computational units in a learning model in a manner similar to human neurons* [6].

  Neural networks, which are theoretically capable of learning any mathematical function with sufficient training data, are gaining popularity nowadays. This is due to the increase in the computational capacity which allows models to be trained within a reasonable amount of time and cost.

  There are different types of neural networks with their own characteristics and motivations. We will be working with LSTMs, which are classified as RNNs.

- RNN: A recurrent neural network is a class of artificial neural network that allows previous outputs to be used as inputs and directly interact with hidden layers. This means that RNNs are able to keep an internal state which allow them to persist information during some processing. That makes them ideal to process time-series data, like the one we will be working with.

- Deep learning: Deep Learning can be defined as a sub-field of machine learning that is based on artificial neural networks that learn from multiple levels in order to provide a model that represents complex relations among data.

  The "deep" in deep learning comes from the use of multiple layers in the network. Stacking multiple layers composed of non-linear neurons allows us to solve non-linear complex problems. For this purpose, this computational system has been chosen, as we will be trying to find patterns on a considered stochastic process.

- TensorFlow: TensorFlow[1] is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. Developed by Google.

- PyTorch: PyTorch[2] is a free and open-source machine learning library based on the Torch library. Developed by Facebook's AI Research lab.

[1] TensorFlow Official Website: https://www.tensorflow.org/
[2] PyTorch Official Website: https://pytorch.org/

Task	Hours of work	Percentage from total work
Initial studies	20	6.66 %
Models development	177	59 %
Web application prototype building	53	17.67 %
Project documentation development	50	16.67 %

Table 1: Hours of work to complete each task of the project.

- Stock market: *The stock market refers to the collection of markets and exchanges where regular activities of buying, selling, and issuance of shares of publicly-held companies take place* [7]. These operations are ensured by institutions which evaluate the legitimacy and the order of the transactions that continuously occur.
- Technical analysis: *Technical analysis is a trading discipline employed to evaluate investments and identify trading opportunities by analysing statistical trends gathered from trading activity, such as price movement and volume* [8]. Practitioners of this discipline defend that past trading activity and price changes of a security can be valuable indicators of the security's future price movements.

3 Initial project planning and budget

The project is estimated to be completed after 300 hours of work. These estimated hours are the result of the sum of the estimates for the different tasks that make up the project. For the highest level project tasks, we find the distribution of working hours in Table 1.

These hours of work are distributed in such a way that the starting date is the 16th November of 2020 and the expected end date is the 4th June of 2021, with an estimated 25 hours weekly work. However, from the 26th of January to the 26th of May, the project will be paused due to the development of the second semester of the university.

3.1 WBS/PBS

In relation to the tasks that are going to be performed throughout the project, it can be interesting to refer to the WBS in Figure 1, as well as to the PBS in Figure 2 in order to see the result products.

3.2 Planning

Now that we know the tasks that need to be performed in order to complete the project, let us see how are they distributed along the time[3].

Initial studies Represents the tasks of studying some basic knowledge for the development of the project. Deep learning concepts, as well as neural networks and their application into the stock market are considered (Table 2).

Task	Work	Start date	End date	Responsible
Initial studies	20 hours	mon 16/11/20	thu 19/11/20	
Deep learning study	6 hours	mon 16/11/20	tue 17/11/20	Software Eng.
Neural networks study	7 hours	tue 17/11/20	wed 18/11/20	Software Eng.
Deep learning applied to stock market study	7 hours	wed 18/11/20	thu 19/11/20	Software Eng.

Table 2: "Initial studies" planning.

[3]Note that milestones are represented in italics.

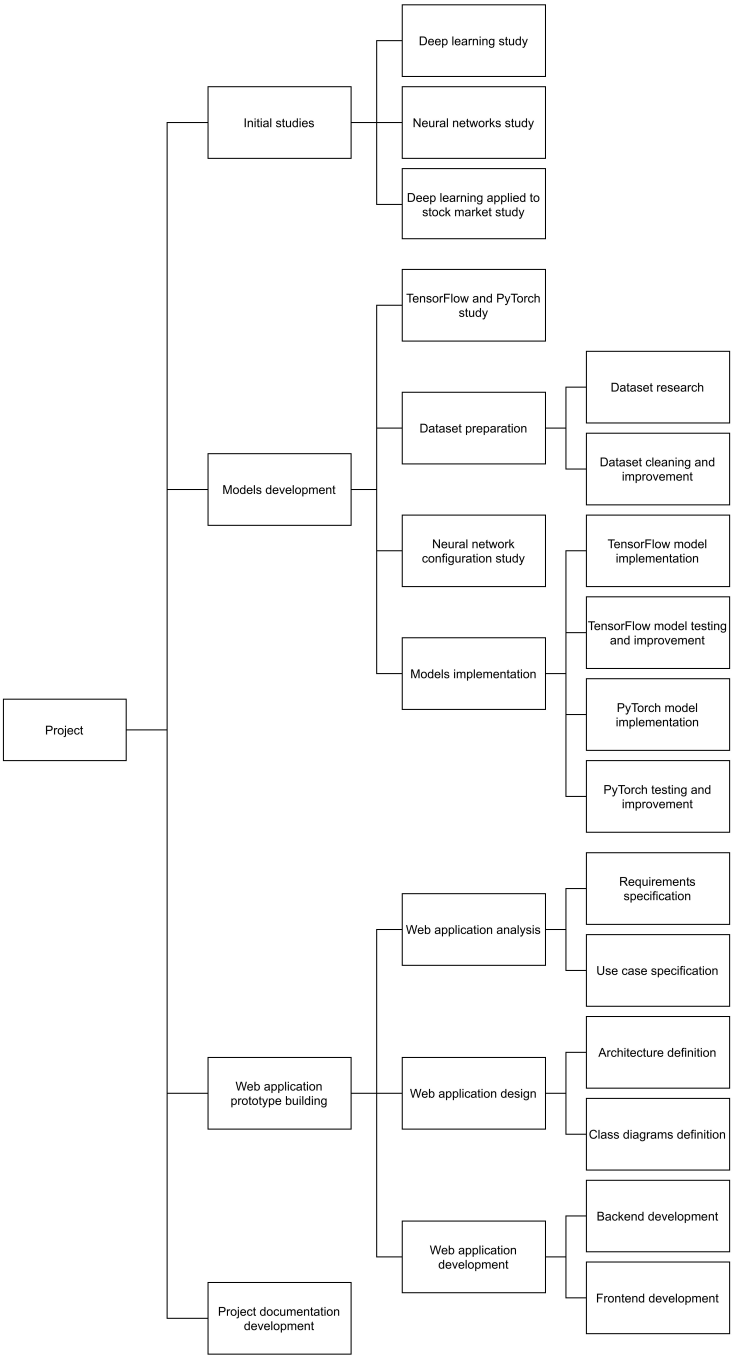


Figure 1: Project WBS.

Figure 2: Project PBS.

Models development Represents the whole process of TensorFlow and PyTorch models development, from the study of both libraries to the implementation and testing of the models (Table 3).

Task	Work	Start date	End date	Responsible
Models development	177 hours	mon 16/11/20	tue 05/01/21	
TensorFlow and PyTorch study	30 hours	fri 20/11/20	fri 27/11/20	Software Eng.
Dataset preparation	25 hours	mon 30/11/20	fri 04/12/20	
Dataset research	5 hours	mon 30/11/20	mon 30/11/20	Software Eng.
Dataset cleaning and improvement	20 hours	tue 01/12/20	fri 04/12/20	Software Eng.
Neural network configuration study	30 hours	fri 04/12/20	thu 10/12/20	Software Eng.
Models implementation	92 hours	mon 16/11/20	tue 05/01/21	
TensorFlow model implementation	20 hours	thu 10/12/20	wed 16/12/20	Software Eng.
TensorFlow model testing and improvement	26 hours	wed 16/12/20	wed 23/12/20	Software Eng.
TensorFlow model	0 hours	wed 23/12/20	wed 23/12/20	
PyTorch model implementation	20 hours	wed 23/12/20	tue 29/12/20	Software Eng.
PyTorch model testing and improvement	26 hours	tue 29/12/20	tue 05/01/21	Software Eng.
PyTorch model	0 hours	tue 05/01/21	tue 05/01/21	

Table 3: "Models development" planning.

Web application prototype building Considers the process of building the web application prototype going through all the classical software development phases (Table 4).

Task	Work	Start date	End date	Responsible
Web application prototype building	53 hours	tue 05/01/21	wed 20/01/21	
Web application analysis	4 hours	tue 05/01/21	wed 06/01/21	
Requirements specification	2 hours	tue 05/01/21	wed 06/01/21	Software Eng.
Use case specification	2 hours	wed 06/01/21	wed 06/01/21	Software Eng.
Web application design	9 hours	wed 06/01/21	fri 08/01/21	
Architecture definition	5 hours	wed 06/01/21	thu 07/01/21	Software Eng.
Class diagrams definition	4 hours	thu 07/01/21	fri 08/01/21	Software Eng.
Web application development	40 hours	fri 08/01/21	wed 20/01/21	
Backend development	30 hours	fri 08/01/21	mon 18/01/21	Software Eng.
Frontend development	10 hours	mon 18/01/21	wed 20/01/21	Software Eng.
Web application	0 hours	mon 20/01/21	wed 20/01/21	

Table 4: "Web application prototype building" planning.

Project documentation development Finally, the task that considers the process of representing the work done in a written documentation (Table 5).

Task	Work	Start date	End date	Responsible
Project documentation development	50 hours	wed 20/01/21	fri 04/06/21	Software Eng.
Project documentation	0 hours	fri 04/06/21	fri 04/06/21	

Table 5: "Project documentation development" planning.

3.3 Summarized budget

Below, in Table 6, there is the summary of the estimated budget broken down by items. Taxes are not included.

Item	Description	Total
01	Initial studies	899.35 €
02	Models development	7,959.21 €
03	Web application prototype building	2,383.27 €
04	Project documentation development	2,248.36 €
TOTAL		**13,490.19 €**

Table 6: Summarized budget.

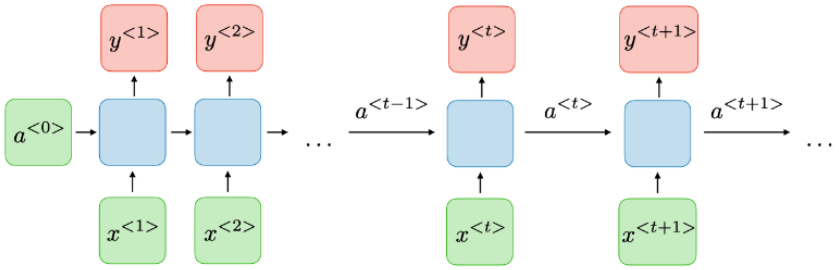


Figure 3: RNN architecture [1]

For a more detailed initial budget, please check Appendix 12.2.

4 Model configuration analysis

A model can be defined as a mathematical representation of a process. That is, a model can be seen as a mathematical function which represents a pattern extracted from a training process over a set of data. In the current project, the idea is to build a model able to get a pattern from the stock market price data.

Many considerations should be taken into account when building a deep learning model. For that reason, we are going to define the general characteristics of the one designed for the project.

4.1 Neural network typology

In the stock market, prices have relationships among them, that is, data is not independent from one another. This format can be found on several data types such as time-series, text, and biological data.

In a time-series, which is the case of the stock market data, the values on successive time-stamps are closely related to one another. If one uses the values of these time-stamps as independent features, then key information about the relationships among the values of these time-stamps is lost. Therefore, the two main wishes for the processing of sequences include *(i) the ability to receive and process inputs in the same order as they are present in the sequence, and (ii) the treatment of inputs at each time-stamp in a similar manner in relation to previous history of inputs* [6].

As we know, RNNs allow previous outputs to be used as inputs and directly interact with hidden layers, keeping an internal state which allow them to persist information during some processing. That makes them the perfect candidate for our objective. They are typically represented as in Figure 3.

For each time-step t, the activation $a^{<t>}$ (which can be found as $h^{<t>}$ on some books, making reference to the hidden state) and the output $y^{<t>}$ are expressed as follows:

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \tag{1}$$

$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y) \tag{2}$$

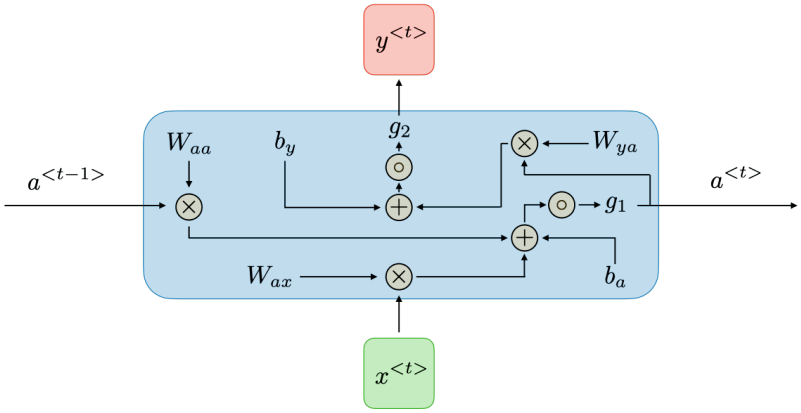


Figure 4: Detail of a RNN cell [1]

where W_{ax}, W_{aa}, W_{ya}, b_a, b_y are coefficients that are shared temporally (the weights and biases, respectively) *and g_1, g_2 activation functions.* [1]

In this way, as can be seen in Figure 4, each input $x^{<t>}$ leads to a $y^{<t>}$ output that is the result of an activation function g_2 applied to the product of the current activation weights and the activation of the current cell, plus the bias of the output.

This activation $a^{<t>}$, in turn, comes from the result of another activation function g_1 applied to the sum of: the product of the activation from $t-1$ and the weights for the activation ($W_{aa}a^{<t-1>}$), the product of the input for the current time-step and the current cell weights ($W_{ax}x^{<t>}$), and the bias of the activation.

This whole mechanism leads to outputs $y^{<t>}$ which are calculated based on previous states. Giving, as a result, outputs that are sequence dependent.

However, in the context of RNNs, we can suffer the vanishing and exploding gradient phenomena which result in a hampering of learning data of long sequences. That can happen due to the difficulty in capturing long term dependencies *because of multiplicative gradient* (which carry information used in the RNN parameter update) *that can be exponentially decreasing/increasing with respect to the number of layers* [1].

These phenomena appear during the optimization of the model. Once the error function from the prediction is calculated, the backpropagation algorithm calculates the gradient to update the model parameters. The gradient of a scalar-valued differentiable function f is the vector field ∇f whose value at a point p is the vector whose components are the partial derivatives of f at p. Once the gradient is calculated, we need to apply an optimization process to reach the minimum of the loss function with respect to the parameters of the gradient recently calculated. At that moment, the gradient can be vanishingly small (for different reasons such as the *tanh* activation function that leads to small numbers between -1 and 1), effectively preventing the weight from changing its value. In the worst case, this may completely stop the neural network from further training. For that purpose, a plain RNN was rejected as the neural network for the model. Instead, a variation from the RNN was chosen; an LSTM neural network.

LSTM stands for Long Short Term Memory, and it is a modified version of recurrent neural networks in which the vanishing gradient problem is solved. From its own name, we can derive that it works not only for short term sequences, but for long too (which refers to the vanishing/exploding gradients phenomena). Therefore, LSTMs are well-suited to classify, process and predict time series given time lags of unknown duration.

LSTMs can be seen as an enhancement of the recurrent neural network architecture in which we change the recurrence conditions of how the hidden states $a^{<t>}$ (or $h^{<t>}$) are propagated. However, an additional hidden vector is needed to retain that long-term memory. That additional hidden vector is denoted by $c^{<t>}$ and referred to as the cell state. *One can view the cell state as a kind of long-term*

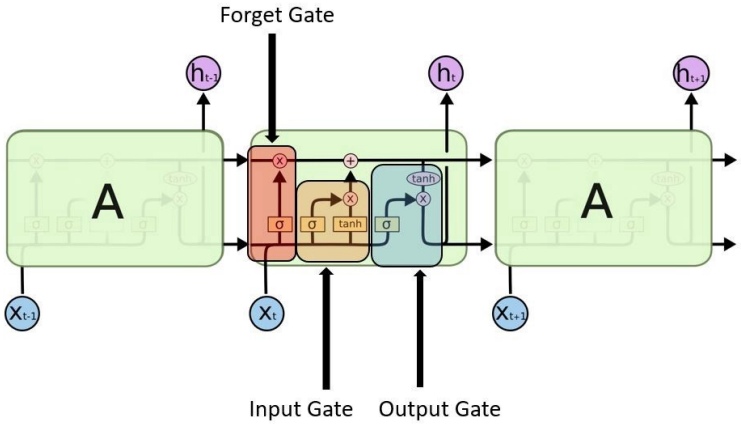


Figure 5: LSTM architecture [2]

memory that retains at least a part of the information in earlier states by using a combination of partial "forgetting" and "increment" operations on the previous cell states [6].

Those partial "forgetting" and "increment" operations are performed by three gates which control the flow of information into and out the cell. Those three gates are called the input, the output and the forget gate (see Figure 5).

As detailed in Figure 6, each gate has its own function. The forget gate is used to decide what information we are going to throw away from the cell state, by applying a *sigmoid* function. It "decides" according to the result of the application of the *sigmoid* function over the hidden state from the previous cell $h^{<t-1>}$ and the input from the current cell $x^{<t>}$. This *sigmoid* function outputs a number between 0 (completely forget) and 1 (completely keep) for each number in the cell state $c^{<t-1>}$.

On the other side, the input gate is in charge of discovering which value from the input should be used to modify the cell state or memory. In this case, we are working with *tanh* function in order to weight the importance of the input values and *sigmoid* function for the rejection or acceptance of values to update in the cell state.

Finally, the output gate is used to decide the output, based on the current cell state. The *sigmoid* function decides which values to let through, whose product with the *tanh* applied to the current cell state $c^{<t>}$, results in the hidden state output at t, $h^{<t>}$.

For more information regarding LSTMs, [2] can be useful.

Once we have understood the mechanics of the LSTM, which will be necessary for the implementation of the model (especially on the PyTorch one), we can sense that it may work for our purpose. However, we do not need to reinvent the wheel and go through extensive analysis to be sure that it may be a good option; we can just check on already done studies. For example, according to [9], LSTM shows the more accurate results with the highest model fitting ability. In this study, LSTM competed against several architectures such as decision trees (bagging, random forest, adaptive boosting, etc.) or artificial neural networks (ANN) with a plain recurrent neural network (RNN). There are many papers showing that LSTM is one of the most appropriated architectures for our topic. In [10], the authors review a high number of studies in which LSTMs were analysed and compared with other architectures; showing LSTM remarkable results for most of them.

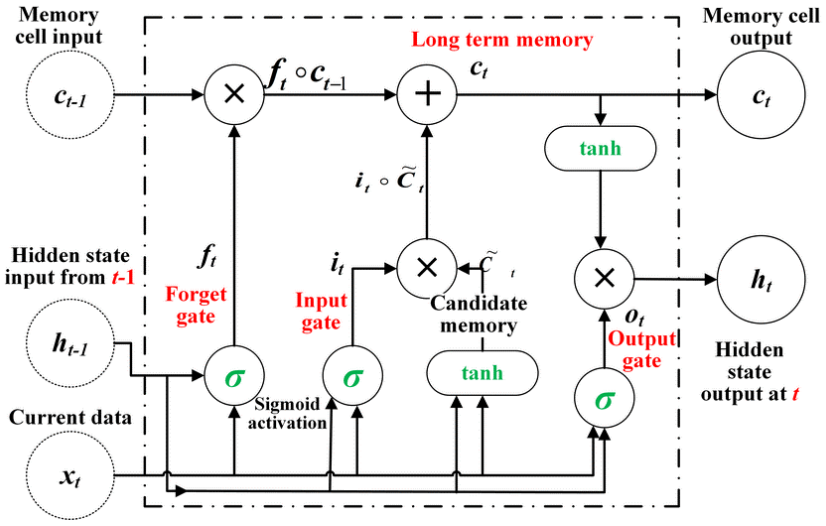


Figure 6: Detail of a LSTM unit [3]

4.2 Activation functions

Complex problems such as the one we are facing are not usually linearly separable. That is, we need some activation functions that add non linearity to our system in order to be able to reach a possible solution. There are many possible activation functions to work with; some of the most popular are the following: *sigmoid*, *ReLU* (rectified linear unit), *tanh* or linear activation. As we are working with an LSTM network, we will be using *sigmoid* and *tanh*, which are the "default" ones for the architecture.

Sigmoid specifically, is used as the gating function for the three gates we mentioned before, since it outputs a value between 0 and 1, it can either let no flow or complete flow of information throughout the gates. On the other hand, the output from *tanh* can be positive or negative, allowing for increases and decreases when added in the internal state. However, the choice of these activation functions may have caused one of the possible errors/improvements detected in the system, which will be discussed in a future chapter. It has to do with the saturation at the edges of the functions.

4.3 Loss function

As we are working with a regression problem, we will make use of the Mean Squared Error Loss or MSE. As the default loss function for regression problems, it was assigned to be the function to be evaluated first and only changed for a good reason. However, one of the suggested improvements for the model deals with the modification of the loss function, but it was considered out of the scope of the project.

4.4 Optimizer

Known optimizers like Adagrad, RMSProp or Adam are optimization algorithms whose objective is to minimize the loss function (by tweaking the weights of the model) calculated after the prediction; they are iteratively looking for minima in the loss function. All of them are gradient-descent algorithms, in which the gradient of the loss function is used to make parameter updates. However, they can sometimes behave unexpectedly because while optimizing they do not always point in the best direction of improvement. The way and the efficiency in which they solve this issue is what makes them different.

For this project, Adam optimizer was the one selected. Adam was released in 2014 as a new algorithm for first-order gradient-based optimization of stochastic objective functions [4]. Furthermore, it was subjected to tests that demonstrate that it works well in practice and compares favourably to other stochastic optimization methods. About the learning rate α used for the models, initially it is set to

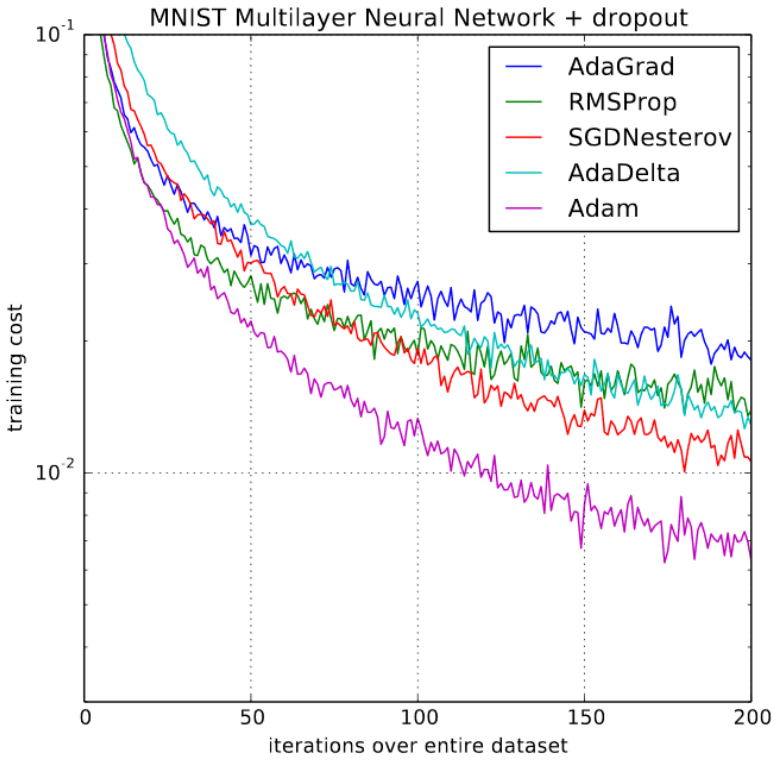


Figure 7: Training of multilayer neural networks on MNIST images [4].

0.001. *Adam, as opposed to stochastic gradient descent, computes different learning rates as learning unfolds; that is, learning rate is not static* [4]. In terms of performance, we can find some good result comparisons against other optimizers in the Adam release paper (Figure 7).

5 Dataset

5.1 Dataset research

Finding the right dataset is a key part of every deep learning process. As we have stated in the general considerations (Chapter 2), deep learning algorithms need a high amount of data in order to work properly.

At a first sight, finding datasets about the stock market may look like an easy task, and it is. Finding free datasets is completely another thing. We should not forget that we are working in a financial activity in which the profits of some, are the losses of others. It is not the same as if we were working on an environmental or linguistic issue whose investigation benefits everyone.

In addition, as we are working with deep learning algorithms, we should ensure to have a big amount of data for the model to work appropriately. That is, daily frequency stock market data, which is the one that can be found free easily, needs to be discarded. If we were working with it, considering that the stock market opens about 253 days in a year and supposing that the firm we are interested in is in the market from 2000, we will only have about 5060 entries in our dataset. Furthermore, we should use a piece of our dataset for validating our model, taking away an important number of entries. For that purpose, the idea of trying to get a model from daily data was rejected, as we would get a very poor dataset for such a complex field.

Despite the nature of the field in which we are willing to work, a free data provider has been found.

time	open	high	low	close	volume
31/12/2018 04:01:00	38.40398	39.40398	38.40398	38.40398	2196

Table 7: Raw data entry.

That one is Alpha Vantage[4], which delivers a free API for real time financial data and most used finance indicators in a simple *json* or *pandas* format. This API can be very interesting for real-time projects, however, for the moment we just need intraday (minute frequency) stock market data for the period of a year. With this period and frequency, we can get a dataset with about 165000 entries, which is far better than the initial one.

Once we have the dataset, we have completed an important part of a typical deep learning process. In many cases, this phase is one of the hardest and may imply the need of collecting data manually for a long time.

5.2 Data pre-processing

Prior to input data into the desired model, it should be cleaned and organized according to the model structure. Similar to when we cook something before eating, our neural network expects data in a certain format in order to work properly. For that purpose, some pre-processing needs to be applied on it. Our current raw data, looks like Table 7.

An entry from our dataset is composed of the time (minute precision), the price of the security at market opening time, the highest price registered for that time, the lowest one, the price at market closing time, and the volume. The volume is the number of shares of a security traded during a given period of time. Draw conclusions about the market behaviour with just this information may be hard for the model. For that reason, it was decided that data should be complemented with some extra information that could help the network to make more robust predictions. That complement was found in the technical analysis.

5.2.1 Technical analysis indicators

Technical analysis is a trading discipline employed to evaluate investments and identify trading opportunities by analysing statistical trends gathered from trading activity, such as price movement and volume [8]. Practitioners of this discipline defend that past trading activity and price changes of a security can be valuable indicators of the security's future price movements. Despite the fact that not everyone believes on the assumptions made by the technical analysis, we will be adding some technical indicators trying to enhance model's prediction ability. Being sceptical about the subject, we could test it experimentally. In any case, we can always turn to studies that have already done it for us. In [11], they evaluate the impact of technical indicators on stock forecasting; arriving to two main conclusions: *(i) lagging technical indicators such as the Exponential Moving Average and Weighted Moving Average, when used as isolated inputs of the neural networks, can improve the accuracy of the stock forecast when compared to forecasts made with the original series of closing prices; and (ii) the combination of different indicators as inputs to the same neural network can improve even more the forecasting performance.*

Technical indicators can be classified according to what they look after. There are basically four main types we will be working with: trend, momentum, volatility and volume. Trend indicators are used to get information from the underlying trend of the price. Momentum indicators show us the strength of the trend. Volatility ones can give us information about the stability of the market: whether the market is very volatile or not at a specific moment. Finally, volume indicators can help us understand the volume changes on the stock securities.

The idea is to get a set of indicators from each category and adding them as extra information to the dataset (Table 8). They were selected according to two main criteria: availability on Technical Analysis

[4]Alpha Vantage Official Website: https://www.alphavantage.co/

Category	Indicator	Name
Trend	SMA	Simple Moving Average
	WMA	Weighted Moving Average
	MACD	Moving Average Convergence Divergence
Momentum	ROC	Price Rate Of Change
	STCK	Slow stochastic oscillator
	STCD	Fast stochastic oscillator
	RSI	Relative Strength Index
	LWR	Williams Percent Range
Volatility	ATR	Average True Range
	BB	Bollinger Bands
Volume	AD	Accumulation/Distribution
	VPT	Volume Price Trend

Table 8: Technical indicators selected.

close	close_sma	trend_sma_slow	trend_ema_fast	trend_ema_slow
38.63594949	38.64754784	38.68487858	38.65710181	38.65687905
volatility_atr	momentum_rsi	volume_vpt		
0.03155724	46.86771017	-0.135481899		

Table 9: Entry with some technical indicators already processed.

library[5] and analogies with similar studies ([9] and [12]). The addition of this set of technical indicators results in a considerable increase in the dataset entry size; causing some changes in the future PyTorch model both to lighten the dataset and not to overload it with information that may include noise.

It is worth to mention that, as a result of their incorporation, the original indicators "open", "high", "low" and "volume" are deleted; as they are considered redundant with all the technical indicators. "time" is also deleted for manageability purposes. Instead, "close" indicator is still in the dataset as it represents the real price data. At this moment, a simplified (not all mentioned indicators are included) entry from our data would look something like Table 9.

5.2.2 Data smoothing

If we observe how the price changes, we can appreciate that it follows a quite erratic pattern of continuous upwards and downwards trends. This behaviour is typically found on time series data and it is usually desirable to treat it accordingly; by smoothing the data. Smoothing is a technique applied to remove the fine-grained variation between time steps. The objective of smoothing is to remove noise and better expose the signal of the underlying causal processes. It can also be seen as a remover of low relevance trends that may confuse our model. In our data-processing, smoothing is applied to the "close" price indicator through a SMA with a window size equal to 4. The results can be appreciated in Figure 8. Now, the data entry would look like the one in Table 9 but removing the raw close price column, which is replaced by the smoothed close price.

5.2.3 Data scaling

It is known that neural networks perform better or converge faster when features are on a relatively similar scale and/or close to normally distributed. That is why we are willing to scale our data, so that features arrive in a more digestible form to the network.

RobustScaler from Scikit-Learn[6] was the tool used for the scaling of the data. After this process, the

[5]TA library: https://github.com/bukosabino/ta
[6]Scikit-Learn: https://scikit-learn.org/stable/

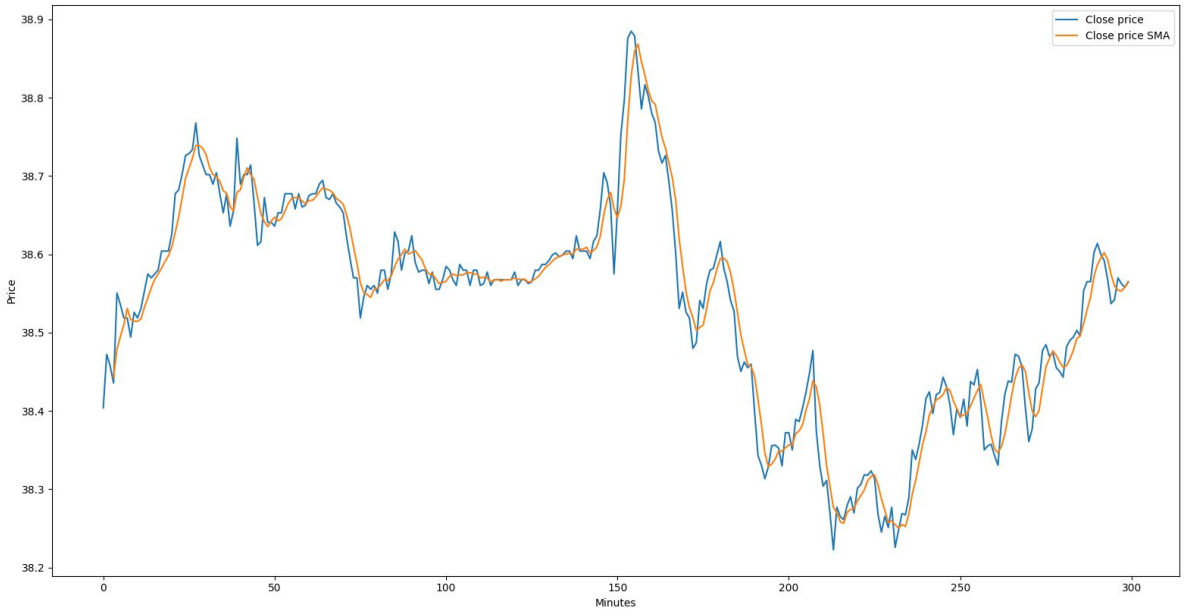


Figure 8: Example of raw and smoothed close price data.

close_sma	trend_sma_slow	trend_ema_fast	trend_ema_slow
-1.146647413	-1.143968589	-1.145974339	-1.146810925
volatility_atr	momentum_rsi	volume_vpt	
0.21757989	-0.2685393	-0.003430889	

Table 10: Entry with data already scaled.

features are the same but scaled (Figure 10).

5.2.4 Time series format

As studied during neural network analysis (Section 4.1), we will be working with time-series data. Otherwise, if we treat prices as independent features, we will lose a lot of information. For that purpose, we have chosen LSTMs to be the backbone of our model, and now, we should convert our data accordingly. A function with that intention has been designed. The idea behind the time-series feeding is to keep a fixed length time range and move forward. That is, we choose a time range (100 minutes for example) and we go minute by minute creating new blocks of time ranges' length minutes. The idea is that the LSTM can relate the previous values with the one is making the prediction to.

Depending on the time range (or time steps) we are using for our model, the prediction will be based on different sized contexts. If we choose to have a 1000 minutes time range, our prediction will be based on a 1000 minutes window, which may be too large for a prediction of the price of the next 1 minute but too little for a prediction of the price of the next 800 minutes. We should keep a balance between the time range we are picking and the number of predictions we would like to get. Otherwise, the network may have difficulties to detect patterns. We can find an analogy in the weather. If we want to know the weather for tomorrow (that is, a short prediction) we need to analyse the meteorological agents from the past week (a short time range). However, if we want to know the weather for the whole next month, we should study the meteorological conditions for a longer previous period, that is, a year for example. In the created function for converting data to the desired format, we are using time range as a parameter so that we can adapt our needs easily. Example from Figure 9 works with a time step of 100 minutes and a prediction of 10 minutes.

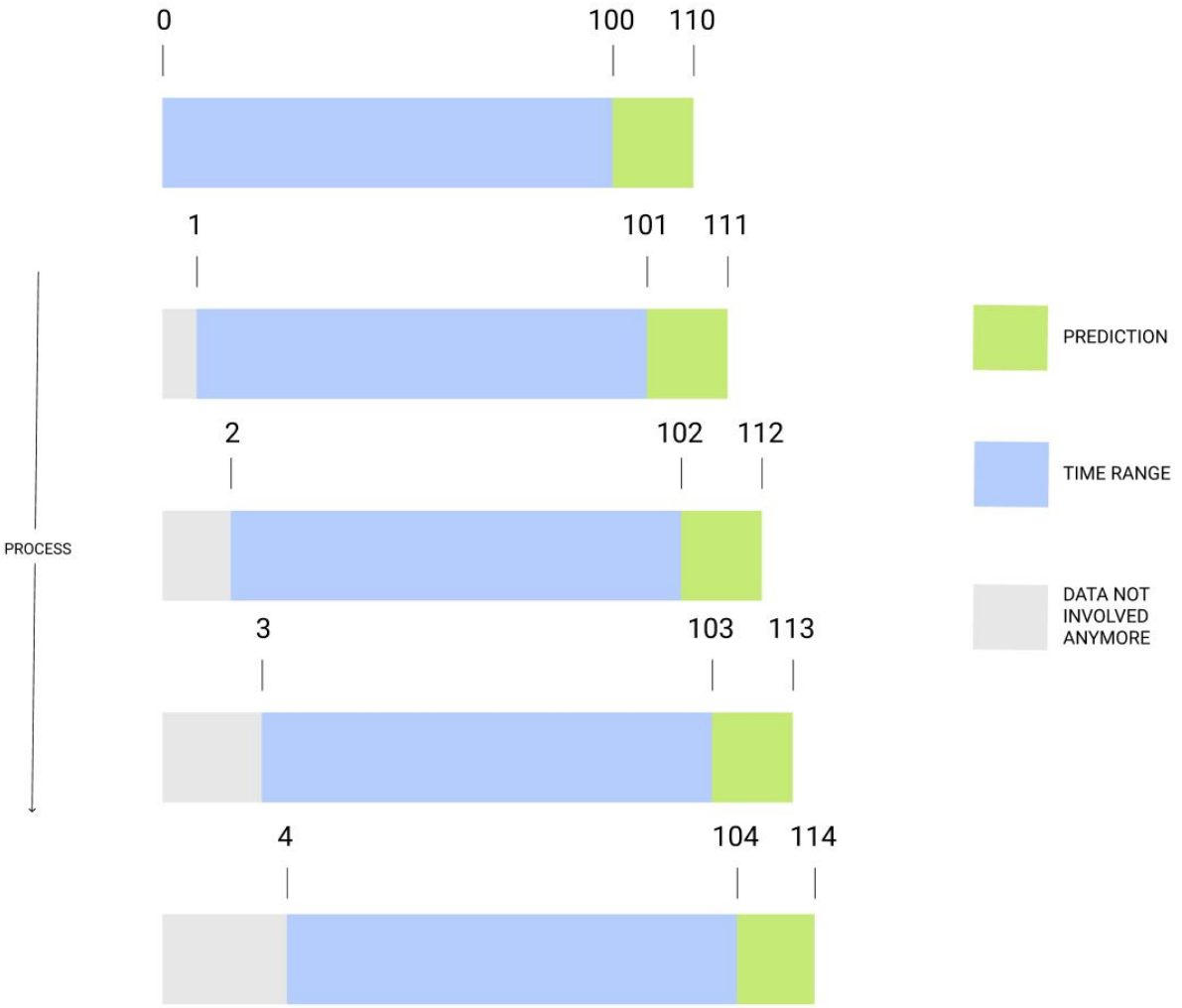


Figure 9: Time-series data conversion process (Time-step=100 and Number of predictions=10).

We are including the time-series formatting as a part of the pre-processing, however, it is worth noting that the time step and the number of predictions changes in accordance with the model parameters. That means that, particularly when testing, we should be concerned that data format may change if we decide to modify our model.

6 TensorFlow

As probably the most popular library for machine learning, TensorFlow will be our first candidate for the analysis. Developed by Google Brain team, has a particular focus on deep learning. As a brief introduction to TensorFlow, we can say that the name "TensorFlow" is derived from the operations that neural networks perform on multidimensional data arrays or tensors. We can think of tensors as the mathematical representation of multidimensional data arrays, which are data structures. In relation to the programming language interfaces, while the library has several for different programming languages, the Python's one is the most used and the one we will be working with.

It is worth noting that all TensorFlow content on this project is based on TensorFlow2, which basically removes the need of (i) the users to manually build together an abstract syntax tree (computational graph) and (ii) the use of sessions in order to run the model (apart from many other improvements such as the introduction of Keras API). With TensorFlow2 we can enjoy eager execution that evaluates operations immediately without building graphs; operations return concrete values instead of constructing a computational graph to run later. For particular information about TensorFlow, there is an extensive documentation available on their official website[7].

6.1 Model development

In order to build the structure of our model according to what we have defined in the design of the neural network (Section 4.1), we will be specially working with Keras Tensorflow's API. *Keras is a deep learning API written in Python*[8] which is built on top of Tensorflow2 and provides high-level functionalities such as the creation of layers or the optimizer definition. The structure of our model consists on several layers (concretely four layers) of LSTM and a final *Dense* layer for the output. Those LSTM layers make up a stacked LSTM structure in which all of them, but the last one, return the full sequence outputs. Last layer returns just the output of the last sequence, which is the one considered the prediction. This is achieved through the use of *return_sequence* parameter from the LSTM layers. In relation to the *Dense* layer, we just basically input the hidden state of the final LSTM and output the previously defined number of predictions. All these parts are inside a unique layer called *Sequential*, which is a Keras structure that holds a linear stack of layers. The specific code used is Algorithm 1.

```
model = Sequential()
model.add(LSTM(200,return_sequences=True,input_shape=(TIME_STEPS,X_train.shape[2])))
model.add(LSTM(200,return_sequences=True))
model.add(LSTM(100,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(NUMBER_PREDICTIONS))
```

Algorithm 1: Code to build the stacked LSTM model with TF.

Additionally, the loss function (Section 4.3) and the optimizer (Section 4.4) should be specified. Let us recall that we will be working with Adam, as the optimizer, and MSE, as the loss function. In TensorFlow, this can be done with single line of code (Algorithm 2).

```
model.compile(loss='mean_squared_error',optimizer='adam')
```

Algorithm 2: Code to specify the loss function and the optimizer with TF.

For debugging purposes, it is very handy to use Keras *summary* functionality because it shows a diagram about the structure of the model we are building (Figure 10).

[7] TF Official Website documentation: https://www.tensorflow.org/guide
[8] Keras Official Website about: https://keras.io/about/

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 100, 200)	179200
lstm_1 (LSTM)	(None, 100, 200)	320800
lstm_2 (LSTM)	(None, 100, 100)	120400
lstm_3 (LSTM)	(None, 50)	30200
dense (Dense)	(None, 1)	51

```
Total params: 650,651
Trainable params: 650,651
Non-trainable params: 0
```

Figure 10: Model summary displayed by TF.

It is important to note that we should take care of choosing the adequate number of layers and nodes. A lower number of parameters can cause that the network is not enough flexible. On the contrary, a very high number can make our network very sensitive to noise and provoke a remarkable increase in the training time. In fact, during the analysis of the different models we have realize that the number of parameters of our models should be reduced. This is due to the fact that our network may be too flexible for the data we are handling. This overabundance of parameters provokes that the network tends to memorize training data, not to adjust the weights to find the most adequate pattern. This election of number of layers and nodes is subject of further investigation, and that is why it is proposed as a potential future extension (Section 10.1).

Finally, taking into account that the training may lasts for some days, we must ensure everything is correct before launching the process. Regarding the number of epochs, they depend a lot on the batch size as well as in the amount of data using for the training, so it is not remarkable.

6.2 Analysis

Once the model development process is finished, in addition to having created our model whose results will be analysed in a future section (Chapter 8), we have drawn a series of conclusions as a consequence of the interaction with the library. These will be analysed in order to extract some information that may be of help to other developers who are starting with it.

The first time we face TensorFlow, we come across a tool which is so broad that we really do not know what it does and what it does not. On the official website they described themselves as a "platform" for the development of machine learning. The term "platform" does not really give us any clear idea about what you can achieve through TensorFlow, but that may be the intention. TensorFlow, more than a library such as Keras or Scikit-learn, it is an ecosystem that includes tools, libraries, documentation and community. For this reason, it is very difficult to understand what specific functionality performs, since it does not solve a specific problem as common libraries usually do, but rather covers a whole field of artificial intelligence.

This issue may cause a slowdown in learning. Although it seems contradictory, with so much information and such a wide range of possibilities, it is quite difficult for new developers to find their way. In addition, the information overload and the huge amount of documentation provoke that learning from scratch, is not that easy. It should be noted that all this argumentation focuses on emphasizing the difficulty of knowing what you want and what you can do when facing TensorFlow for the first time. Not to confuse with the ease of developing the first model with the library, which is straightforward by following the documentation. The key, in my experience, is to follow someone's path that can introduce you smoothly to TensorFlow's concept, either through classes at the university or through some online

courses.

As we mentioned before, building a model without even knowing what TensorFlow covers, it is possible. The documentation offers a series of examples with different difficulties which guide you to the creation of a first prototype model. We would consider this as one of the main advantages of TF, as everyone with almost nothing of machine learning knowledge can build something functional very easily. Even, if you are lucky and one of the examples resembles the model you want to build, it should not be difficult to make the necessary changes and build the model in no time. Furthermore, these small examples presented in the documentation, help the user to understand some of the possibilities offered by the library.

Thus, the complexity of the first step in TensorFlow can vary according to the objectives of each individual. If the idea is to use TensorFlow as a tool for a project with an objective already defined in advance, the user may collide with the large amount of information and possibilities that the library offers us; causing a considerable investment of time in understanding and evaluating the tasks to be performed by TF. On the other hand, if what the user wants is a quick immersion in the world of machine learning, it will not be difficult to find the way of developing a first functional prototype with the help of the tutorials provided in the documentation.

On the other hand, if we focus on the documentation, we can say it is the ideal place to solve specific doubts, not being the case to learn the library from scratch; due to the great extension of the documentation, as well as the different specific uses that each individual wants to give to the library. In any case, the documentation is well written and organized, making it very convenient to look up any issue you may have. For example, in the case of our project, the documentation was very useful to understand some of the parameters of the LSTM layers. Finally, as an advantage for users who have already experimented with the Android Developers documentation, the TF one (being of the same nature, Google) is very similar in format, so the way to interact will be familiar.

One of the drawbacks that have been found, which has caused certain problems when deploying Docker containers, is the restriction imposed by TensorFlow with Python versions. The library is only supported by Python 3.5 to Python 3.8 and in 64-bit versions. For Python 3.8, in fact, it only works as of TensorFlow 2.2. These restrictions can cause certain difficulties when it comes to adjusting versions with other software. This may seem banal for a software/computer engineer, but TF is used by mathematicians or physicists that may find more difficulties in order to fix it.

Nevertheless, being the current leader in Google searches compared to other libraries in the sector, such as PyTorch or Keras, TensorFlow has to be doing things right. In the past, TensorFlow1 was loosing its public due to the lack of a higher level approach that may be more adapted to all the users. That "weakness" was exploited by other libraries, like PyTorch, that started seeing an opportunity by developing a higher level library that provided a more intuitive process. The need of dealing with computational graphs and sessions was not something that all users were willing to do. That is why the release of TensorFlow 2.0 in 2019 made the library being very well received by users, who were looking for an intuitive tool with a more pythonic way of working. On the other hand, an important part of TensorFlow's success is the result of the feedback provided by the large community that surrounds the library. In any case, one of the most relevant keys for the library success is its versatility. TensorFlow can be used in multiple ways; in which each one requires a level of knowledge and learning. The use that an inexperienced user will make of it, is totally different from that of a professional user, who will be able to reach much higher levels of detail. However, both can achieve their goals. This makes TensorFlow adapted to users with all kinds of needs, providing a versatility from which much of the success of the library is born.

7 PyTorch

Parallel to TensorFlow, there are many other libraries that have a niche in the market. PyTorch is one of them. In fact, PyTorch is one of the libraries that today is able to compete with TensorFlow; getting to overcome it in certain areas. Similar to TF, PyTorch is an ecosystem rather than a library, since in addition to the library itself it has a lot of additional tools for different use cases. Primarily developed by

Facebook's AI Research lab, this library is focused on machine learning. If we refer to the documentation of what the PyTorch library itself would be, not PyTorch as an ecosystem, they themselves define their library as an optimized tensor library for deep learning using GPUs and CPUs. Referring to the issue of the GPUs, PyTorch has a module called CUDA, in which tensors that are defined through this module will be computed directly through the system's GPU. On the other hand, the main interface would be the Python one, even though it has another for C++. For more technical information regarding PyTorch, it may be useful to check on the documentation available on their official website[9].

7.1 Model development

PyTorch models, as with the TensorFlow one, follow the same configuration that we made at the beginning of the project (Chapter 4), since it is the design that we have considered suitable for our problem. Anyway, as we are handling a reduced amount of data (as we will see in Section 8.2), parameters such as the number of nodes will be affected. Also, training times will be shorter.

Although we are reusing the same data pre-processing as for the first model, the way of building and training the model for PyTorch is totally different than in TensorFlow. In PyTorch, we need to define a class that derives from the base class $nn.Module$. This class is the one that is used to create the model and should include at least (i) the constructor of the model and (ii) the execution method of the model (the forward passes of the model). The constructor has no great mystery, it is about defining all the layers that will comprise the model as well as defining attributes that are necessary for its creation and execution. Regarding the constructor that we are defining in our model, as in TensorFlow's one, we build a stacked LSTM structure with a linear layer for the output. In addition, we define two attributes that are used during the execution of the model: the number of hidden layers (LSTM ones) and the number of nodes per layer. Both are necessary for the initialization of a tensor that we will see later.

The model's execution method, or forward method, is subtly more complex. In this method we have to define what our model comprises from when the input arrives until the output is expelled, requiring a certain understanding of the layers that are being used. For that purpose, let us go back to LSTM explanation during model design (Section 4.1). If we recap, in LSTMs, in addition to the hidden state vectors $h^{<t>}$ (or $a^{<t>}$) already existing in the normal RNNs, we have an additional hidden vector which is denoted by $c^{<t>}$ and referred to as the cell state. We could see this last vector as a kind of long-term memory that retains at least a part of the information in earlier states by the participation of the "forget" and "input" gate operations on the previous cell states [6]. These two vectors ($h^{<t>}$ and $c^{<t>}$), which will be tensors, should be defined. According to the PyTorch documentation[10], both tensors must comply the following structure: $(num\_layers * num\_directions, batch, hidden\_size)$. Therefore, two empty tensors with the specified shape have to be inserted as input into our LSTM network.

In addition of the mentioned tensors, we necessarily have to input the data we want to process. To do so, in accordance with the use of the $batch\_first$ attribute set to $True$, the input will be inserted in the following format: $(batch, seq, feature)$.

Once we have defined the tensors to be introduced in the LSTM network, we must collect its output and load it in the Linear layer. Before, we must limit the output of the LSTM to the last time step (last 100 minutes). In such a way, after processing output of the stacked LSTM layers, the linear layer expels the desired prediction.

```
class LSTM(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_layers, output_dim):
        super(LSTM, self).__init__()
        # Hidden layer dimensions
        self.hidden_dim = hidden_dim
        # Number of hidden layers
        self.num_layers = num_layers
        # Building stacked LSTM
```

[9] PyTorch Official Documentation: https://pytorch.org/docs/stable/index.html
[10] PyTorch Official LSTM Documentation: https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

```
        self.lstm = nn.LSTM(input_dim, hidden_dim, num_layers, batch_first=True) #
            batch_first=True causes input/output tensors to be of shape (batch_dim,
            seq_dim, feature_dim)
        # Linear layer
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        # Initialize hidden state with zeros
        h_0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim)

        # Initialize cell state
        c_0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim)

        output, (h_n, c_n) = self.lstm(x, (h_0, c_0))

        #Linear layer process the last time-step data
        output = self.fc(output[:, -1, :])

        return output
```

Algorithm 3: Code to build the stacked LSTM model with PyTorch.

Additionally to model class definition, we should configure the loss function and the optimizer according to what we have designed (Algorithm 4). It is quite different from TensorFlow's way as in PyTorch both of them are defined apart from the model and not "compiled" on it. It is also worth noting that, during training, not only the calculation of the loss function, but the calculation of the gradient during the backward and the step of the optimizer should be all manually called; it is not hidden for the user as in TF.

```
#Loss function -> MSE
loss_fn = torch.nn.MSELoss(reduction='mean')

#Optimizer -> ADAM with learning rate = 0.001
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

Algorithm 4: Code to specify the loss function and the optimizer with PyTorch.

It may draw the attention the fact that we are not using the hidden and cell state (h_n and c_n) in the code. This is due to the way we are training this model, in which we are not dividing the input data into several batches but training with a single batch. Therefore, the hidden and cell state do not need to be transmitted from one batch sequence to another. This decision is committed to simplicity and demonstrates one of the features that most differs PyTorch from TensorFlow: the lower-level API.

In PyTorch, the division in batches is much more complex than in TF, in which it is just a parameter in the *fit* method (Algorithm 5). If we wanted to divide in batches the PyTorch training, we could do it manually or take approach of *torch.utils.data*[11] package. If we decide to go with the PyTorch package, that may be the best solution, we need to (i) define a class extending *Dataset* (a Map-style dataset or an iterable-style dataset, both are supported in the *DataLoader* constructor) that represents our dataset; (ii) create the *DataLoader* with the *Dataset* object and the rest of the desired parameters; (iii) and finally, use *DataLoader* methods in order to pass the different batches to the training.

```
res = model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=10,batch_size=64,
    verbose=1)
```

Algorithm 5: Code to run the training in TF (batch division is done automatically).

As we have seen, there are notable differences in certain aspects of both libraries. In fact, in this last example, we could see how in TF batch division is solved with a single method parameter and in PyTorch requires even creating a new class. However, both may have their justifications for making that choices; let us go into the next chapter to analyse them.

[11] torch.utils.data package: https://pytorch.org/docs/stable/data.html

7.2 Analysis

As we have done with TensorFlow, next we will carry out an analysis on the use of PyTorch. This analysis is the result of the interaction with the library during the development of the model, without prior experience in it. In addition, as having already talked about TensorFlow, we will be focusing on some of the key differences between them. As with TensorFlow, when we talk about PyTorch we talk about an ecosystem of tools, libraries and community, which is already explained on its own website. This produces the same feeling of mental cloudiness as with TensorFlow, but with a more technical vibe. We highlight this since, as we will see, PyTorch has a subtly different audience.

When we run into PyTorch documentation, we can appreciate a separation between what would be the PyTorch library API, and the complementary libraries that make up the ecosystem. This makes us quickly understand the way in which PyTorch is divided, helping us to understand it better. Unlike TensorFlow, PyTorch provokes clearer first impressions based on the similarity to a typical library documentation and, on the selection of the right and necessary content. This, although it may seem trivial, it helps the developer to create a first idea of what the library can provide, speeding up the learning process and not loosing the interest of the user.

As we pointed out at the beginning of the analysis, PyTorch may have a more technical and less commercial look than TensorFlow. This, although not a remarkable attribute, fits with the use of the library. For someone beginning with PyTorch without almost any machine learning knowledge, getting started is going to be difficult. PyTorch, compared to TensorFlow (TensorFlow2, as always) since the incorporation of high-level Kera's API, offers a library with a lower-level API. This makes new users require a greater knowledge of the domain, losing a relevant part of the public that starts from scratch. If we remember from TensorFlow analysis (Section 6.2), the user was offered a series of easy-to-follow tutorials with different levels of difficulty, thus welcoming all types of audiences.

If we refer to the models developed with each of the libraries, we can extract a series of examples that will help us understand the different approach that PyTorch offers. As a first example, it is worth highlighting the fact that to create a model we must define a class with two methods. This idea is called model subclassing and is also available on TensorFlow2. This assumes certain knowledge of programming concepts, like class, constructor or inheritance, that is necessary to understand the development of the model. These types of concepts, although basic for an experienced developer, may be unknown to scientists from other fields of study. On the other hand, we also find the fact that training is not a single model method (as in TF), but must be structured manually. To do this, you must have certain theoretical notions in machine learning training, which you can do without in TensorFlow. Finally, mention that in TensorFlow you can create a model based on LSTM layers without having any idea of how they work internally; case that would make the development in PyTorch much harder since we must take into account the hidden and cell states from LSTM architecture. This set of examples is intended to support the idea that PyTorch assumes a certain level of theoretical knowledge without which it will not be possible to easily employ the library. This does not mean that they are not needed for TensorFlow, but they are much more expendable.

One of the key aspects that has made PyTorch the success it is today, is the fact that from its release it has used dynamic computation graphs. With this way of working, we can change the parameters of a neural network on the go, during execution, just like regular python code. This improves the usability of the library, in addition to follow Python typical way of working; unlike in Tensorflow1, which only worked with static computation graphs and sessions. Later, TensorFlow2 added the possibility of using dynamic computation graphs and cut the differences. However, PyTorch had already gain the attention of a big part of the public. In addition, the idea that PyTorch works with a lower-level API provides a more flexible environment; with the price of slightly reduced automation. This aspect, the more Pythonic look, and some other attributes causes the library to be preferred for an audience focused on research. In the end, scientists looks for a tool that is flexible, efficient [13], simple and the most similar possible to what they already know (in order to reduce the learning curve). TensorFlow is much simpler and intuitive, however it may have impacted in algorithm generalization and, consequently, in non-optimized algorithms for specific neural networks.

Supporting PyTorch dominance in the research sector, we can find some studies that show up the

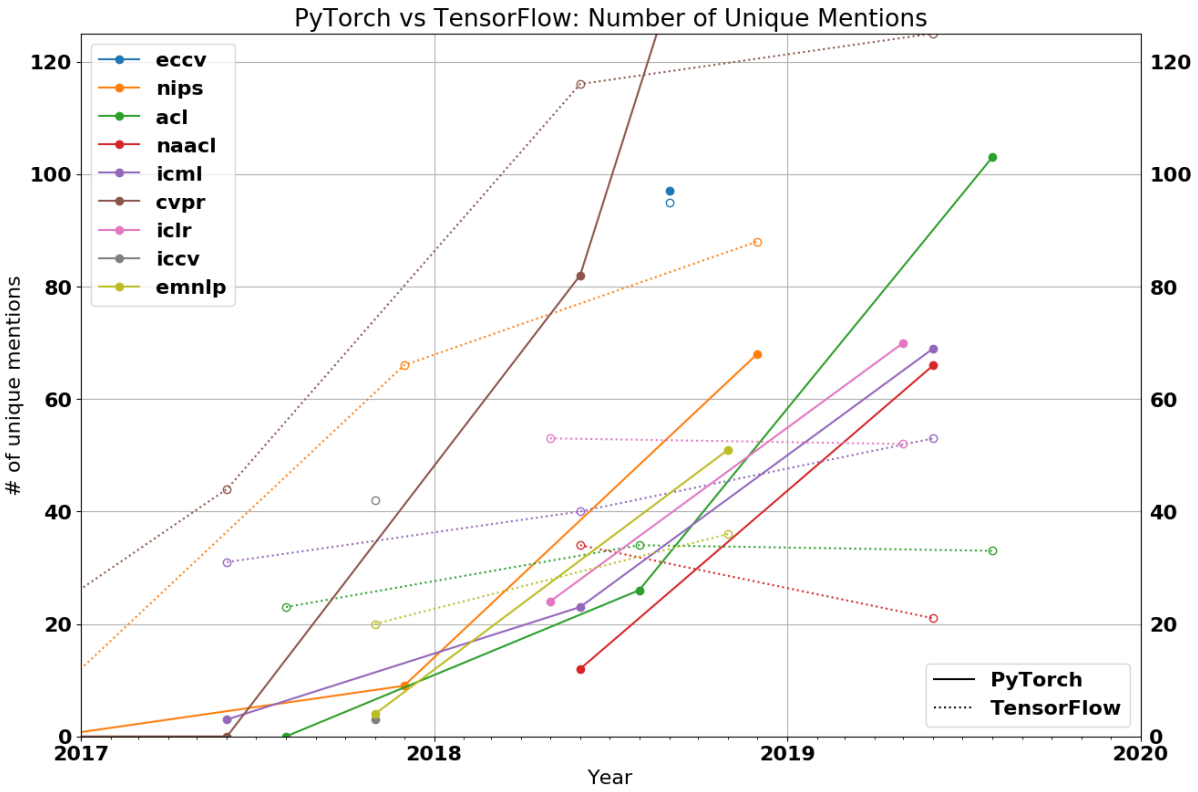


Figure 11: PyTorch's increasing dominance in research [5]

ratio between papers that use either TensorFlow or PyTorch at each of the top research conferences over time (Figure 11).

In relation to Python versioning support, PyTorch is less restrictive. It requires 64-bit version and does not support 2.x, but it works with 3.x versions which are the most common. Thanks to this, PyTorch has not caused any remarkable issue during the development of the project, contrary to TensorFlow.

We conclude the analysis by highlighting that when we talk about PyTorch, we are talking about one of the most relevant ML libraries nowadays. As we have seen, it not only competes with TensorFlow, but exceeds it in certain areas. When the library seems to be preferred in the scientific realm, it is likely that in time it will end up being used in production environments, in which TensorFlow is currently preferred. For this reason and for the many advantages that have been found, if you have some prior knowledge in machine learning, PyTorch will be undoubtedly a good choice to take into account.

8 Models results

During this section we will see the results of some models following the architecture designed in previous sections. In addition, we will see an attempt to improve the prediction system which was, finally, rejected.

8.1 TensorFlow model

The idea is to train the model with 2019 price data from Apple company (AAPL). Training data covers from the 1st January of 2019 to the 10th September (of that same year). Validation data covers the rest until the end of the year, making up a dataset with a total of about 160000 entries. The objective of the model is to predict price data, given some previous prices.

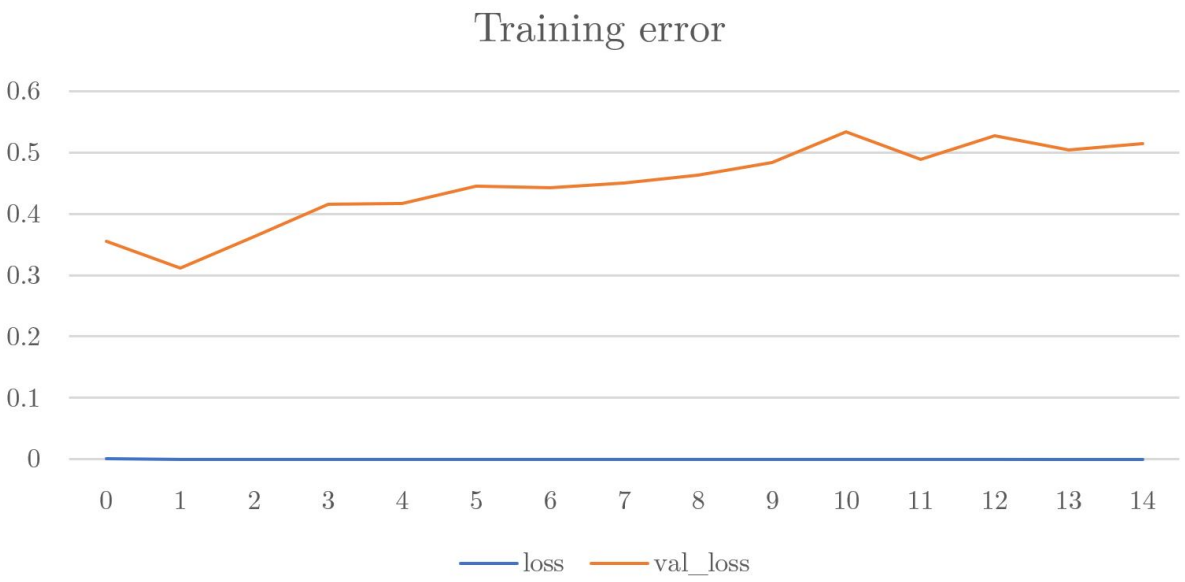


Figure 12: Training error for TF model.

In relation to the model itself, the one with the best performance will be the one studied in detail. After an investigation, it was concluded that the best model was the one with the following time series parameters:

- Time step: 100
- Number of predictions: 1

Other models with different time series parameters pairs like 100/10, 1000/100 or 600/60 have been tested and took part of the investigation, but they have shown worse results, so they will be discarded for the results analysis.

8.1.1 Training

Before looking at the performance of a model, it is always advisable to analyse training results. In this way, we can see if the model needs more epochs, is overfitted, or it seems good. This is done through both training and validation dataset errors registered during training. In an ideal scenario, training error will progressively decrease along with the validation error that remains slightly higher; meaning that the model has been able to adapt to training data but not excessively so as not to end up in overfitting, obtaining worse results in the validation dataset. If we look at the graph in Figure 12 we can see that this scenario is not fulfilled at all, in fact it looks erroneous. Validation loss look incredibly high, while training loss follows normal behaviour. However, these results have an interesting justification.

It is clear that the error in the validation data is not due to a possible overfitting since from the first moment it has been very high and has not approached the error of the training data at any time. So, the idea is to check the original price data and analyse if there is any paranormal behaviour that is affecting our results. If we look at Figure 13, which shows the performance of the stock during 2019, we can see a strong rise in the price during the 4th quarter, surpassing historical maxima. This causes that our model, having been trained with a much lower data range, is not able to react against such a rise and totally loses its ability to make accurate predictions. In fact, our model does not lose its capacity at the beginning of the validation dataset, but as of October 10th. This date matches one of the strongest rises which definitively supersedes the predictions of the model.

Figure 13: 2019 price analysis.

8.1.2 Performance

If we refer to Figure 14, which shows the predictions for the validation data against the real price, we see how until the moment when the strong price rise begins, the model performs quite well. However, when the price starts to aggressively increase, the model losses its total capacity of making accurate predictions.

If instead of looking at the whole validation data we take a look at the section before the sudden rise of the price (Figure 15), we can see how good the model has performed in a the regular period of the stock (that is, prior to the sudden increase to price maxima).

The performance of the model in that section of data presents 76.30% of accuracy in the direction of the prediction and a RMSE of 0.57; which is not so high as we are dealing with values that are over 50. Taking into account that the validation data is completely unknown for the model, the results for that section are more than satisfactory; the hit rate in the direction provides a very important market advantage. Nevertheless, the model was not able to adapt to all the price movements, so let us reflect about how can we solve this.

From these results, some conclusions have been drawn. First, the closer the training values are to the validation ones, the more confident predictions we will have. Therefore, in order to obtain good results, it will always help that the validation window (that is, wherever we want to test our model) is not very large and that it is contiguous to the training one. With this idea we want to emphasize that if anyone wanted to use such a model for real market operations, it would be convenient to train the model with the closest possible data to the objective, in order to be working in familiar territory. On the other hand, the great complexity of the stock market causes that we never work on safe ground, being a problem for real operations, but causing very interesting questions such as those that arise during the development of this project. As we saw, the model performed quite well within a data range not so far from the training data; but not so good when the sudden increase in the price broke the potential pattern that the model could have found. For that reason, I wonder whether there might be a definitive model that accurately describes market behaviour, as there are many external factors such as sales data results, new product releases or changes in economic policies, that may provoke aggressive movements on the price. In any

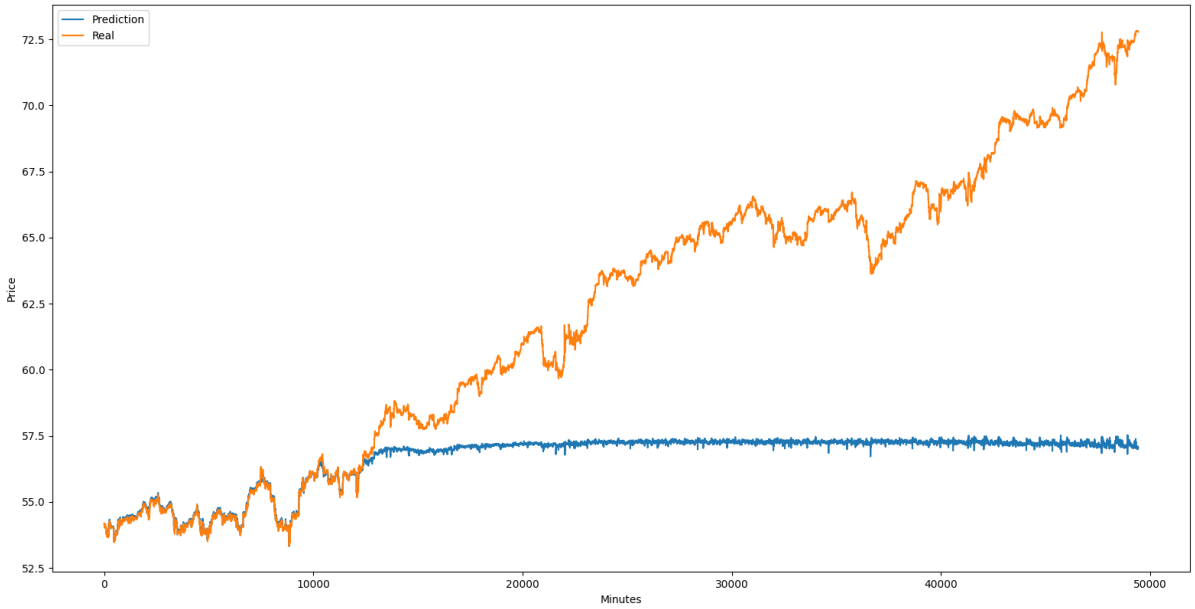
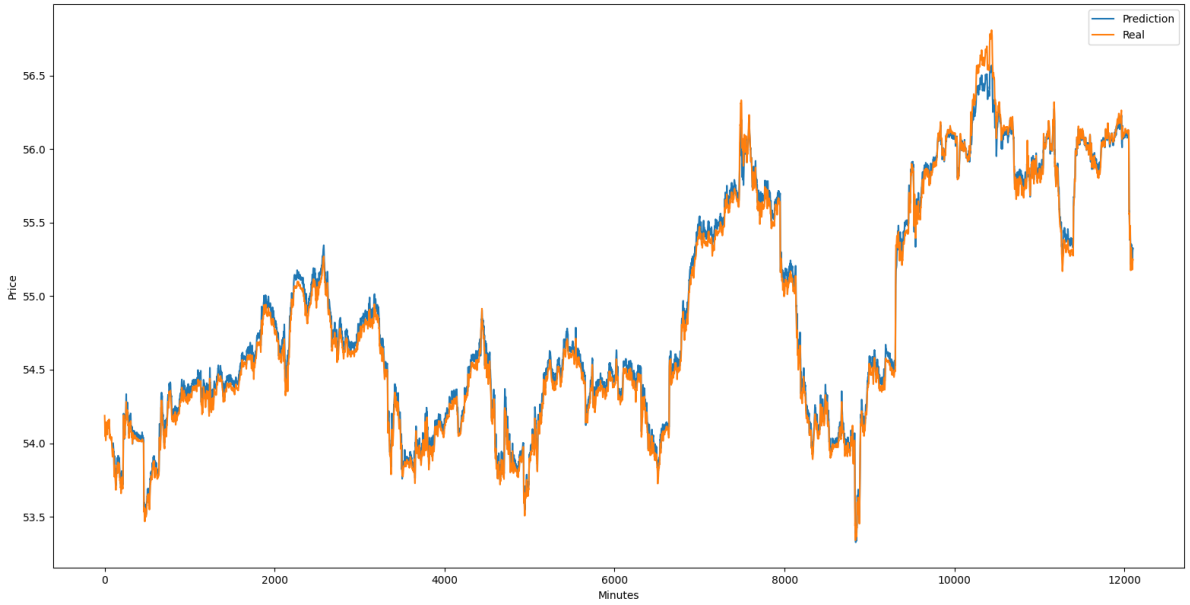


Figure 14: TF model performance on whole validation dataset.



RMSE	Direction Errors	Direction Accuracy
0.056588502699372	2844/11999	76.29802483540296 %

Figure 15: TF model performance on first section of validation dataset (until 10th October).

case, I believe that there may be models that perform correctly in close areas to training data; finding relationships among the closest data, but not finding the universal pattern for the prediction of future values. This last conclusion will be studied with the PyTorch models.

8.2 PyTorch models

Due to the conclusions extracted from the results of the TF model, we have decided to take a different approach for the PyTorch one. The idea is to experiment with different approaches, or to attack the problem from different points of view, with the intention of finding the ideal perspective to solve this problem. It is for this reason that, based on the conclusions drawn from the results of the model with TensorFlow, we will not face the problem of predicting prices in the market in the same way. From the model developed with TF we have seen how strong changes in the market are difficult to capture by a model trained in a context without them. This has caused the model to lose a relevant part of its capacity, despite remaining functional for a fairly important initial period. Based on one of the conclusions of the first model, which calls into doubt the idea that generating a "universal" model that defines market movements is possible, we can try to, at least, look for a pattern between relatively close data to perform predictions in a time range also closer. For this reason, with an experimental intention, it has been decided to change the paradigm for the development of the models with PyTorch. The idea is not to generate a single model based on all the available data, but rather to generate a series of models, based on a "use and throw away" culture, more adjusted to smaller sections of data; from which we may draw more accurate results. We do not know yet if it will be better or worse, but the idea is not to look for a function that defines the market, but a function that is close to the behaviour of the market in the current situation. Specifically, we will work with the last four months of 2019 (which where the ones with the strong price rises); dividing it into three models each trained with one month in order to predict the first week of the following month (for example, train with the whole September month, in order to make predictions during the first week of October). This period of time was chosen to test whether this model format is capable of coping with the strong price rises that ousted the model with TensorFlow. The main hypothesis goes as follows: if working with more concentrated data sections, even working with less data, model is able to get better results (in the proximity), since the pattern will be able to adapt better to the context.

In addition to what we have mentioned, we should define the time series parameters that our models will be working with. If we go back to the results from the TF model we can see that, as expected, the best results came from the 100/1 model (the others were not even shown for clarity purposes). That means that the relation between 100 minutes and 1 minute is better captured by the network, than the relation between 100 minutes and 10 minutes, for example. This balance between the time step and the prediction interferes heavily on the results of the network; predicting 10 minutes based on the 100 previous minutes would mean that the network is able to find a pattern between those two temporal sections, which is so hard when we talk about the stock market. That been said, we will work with the same time series parameters as for TF, that is:

- Time step: 100
- Number of predictions: 1

The three models we will be working with, can be summarized in Table 11.

	Model 1 (Sept/Oct)	Model 2 (Oct/Nov)	Model 3 (Nov/Dec)
Training data	September	October	November
Validation data	1st week of October	1st week of November	1st week of December

Table 11: Summary of the PyTorch models.

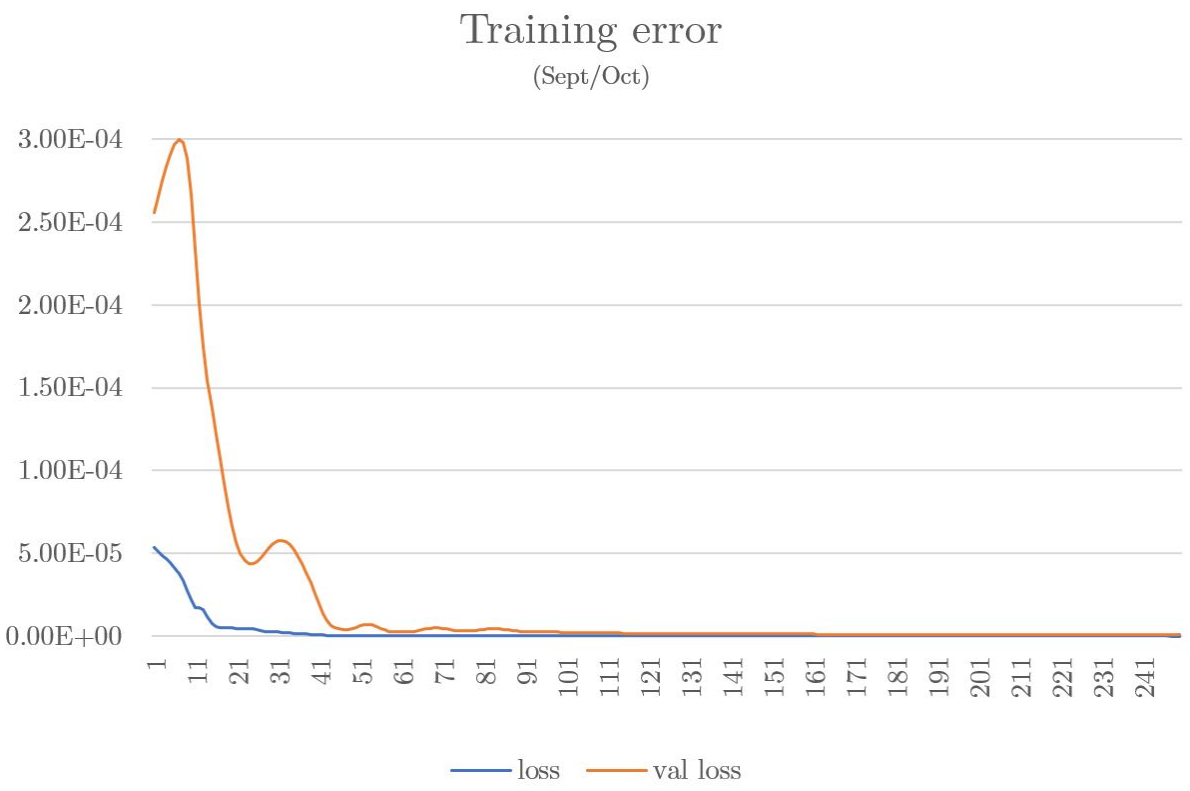


Figure 16: PyTorch training results for Sept/Oct model.

8.2.1 Training

For these models we do not have the problem we encountered with the TF one. That is, they are able to find a pattern for the whole data used for training and validation; there are no problems with the sudden increase in the price, as we suffer in the last model. The only model that may struggle a bit with the validation data is the model of Oct/Nov, as the first week of November there has been another notorious increase in the price. However, it stills have much better results than the one from TF.

8.2.2 Performance

Again, we see how after many tests the results obtained are quite good. They are not so confident as the ones obtained in TF for the initial data section, but here we are still over the 70% of accuracy in the direction prediction. In relation to the RMSE it is also slightly worse as the error is higher, especially for the Oct/Nov model. However, we are talking about models which respond to the stock market with an accuracy in direction that is over the 70%, for data that it has never seen before. Moreover, they all are able to adapt (better or worse) to all the price movements, as the variance in data they are exposed to, is lower than for the whole dataset.

Now that we know that these new lightweight models are working, let us compare their results with the TF's ones. For that purpose, we will compare the Sept/Oct PyTorch model results with the TensorFlow model in the same data section (Figure 22); both in validation data. This is not a random choice, let us remember that the "Good performance" section (Figure 13) of data used for TF is until the 10th of October; fitting with the Sept/Oct model whose validation data is the first week of that same month. If we compare the RMSE, we can find out that the model from TensorFlow is more precise. The lightweight model, has worsen the precision on a -16.11% and the accuracy on predictions direction on

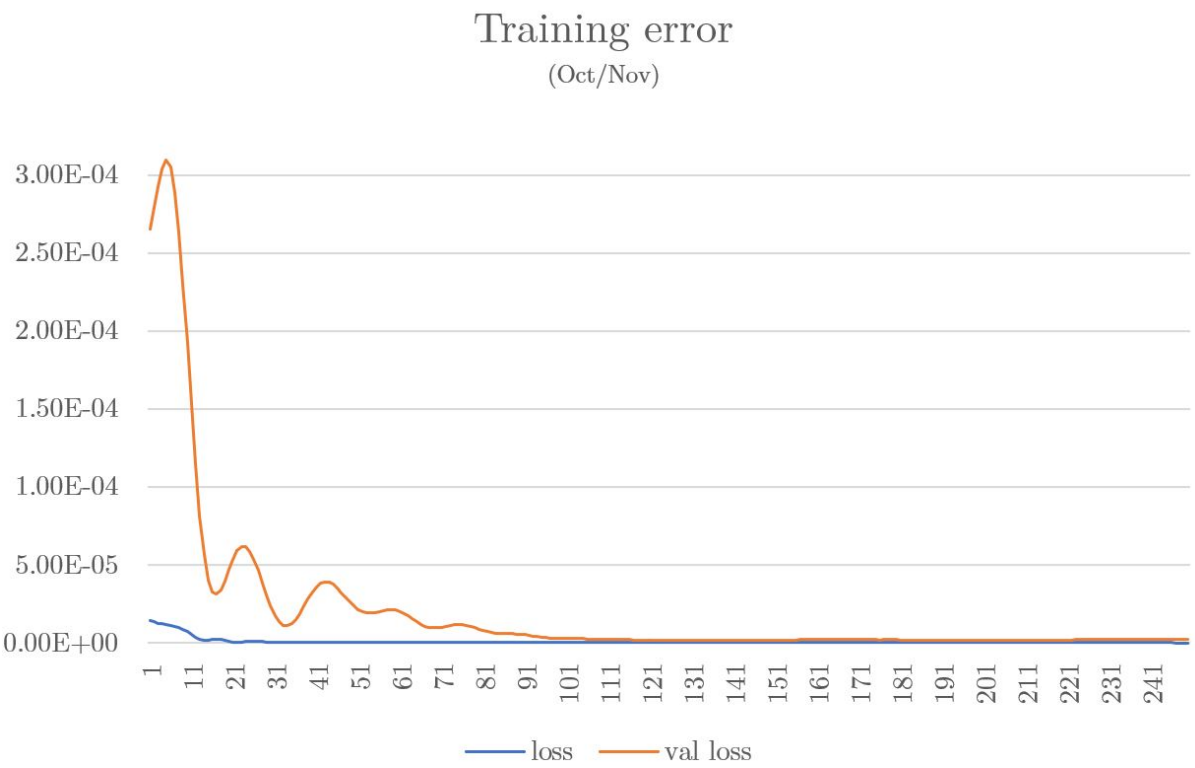


Figure 17: PyTorch training results for Oct/Nov model.

a -5.90%. That been said, our hypothesis can be resolved from two different perspectives:

- From the point of view of the accuracy, the TF model trained with the whole dataset performs better. It is quite superior when predicting exact prices and the direction of the market movement. It has sense since the model is more mature; it has trained with a lot more data.
- From the point of view of the general performance, the lightweight models seem to be more robust. As they are trained to only make predictions in a very reduced context, their adaptability to potential changes in the price is better.

Apart from this, the idea of working with "use and throwaway" models has the advantage of using less data and reducing training time. In fact, the training time has been reduced around -98.61%. Due to this significant training cost difference, lightweight models could be interesting in some cases in which precision is not the unique issue to take into account. Furthermore, we can appreciate how these models are able to better withstand the sudden increases of the prices without distancing itself as much as the TF one. In other words, these sudden increases that killed the TF model do not suppose as much variation for these lightweight models since they have been trained with closer data to that variation. For all this, although accuracy is not as higher, they respond quite well and have their own advantages that should be taken into account when making a choice of which model or approach to use.

On the other hand, despite the results, sudden market movements are still not precisely captured by these alternative models. This problem, whose nature stems from the volatility and irregularity of the market, shows us the complexity of finding firm patterns. Perhaps with a more in-depth investigation a solution that would soften these types of situations could be found. One of the main hypotheses that has been thought, is that the nature of the LSTM layers, whose activation functions are *tanh* and *sigmoid*, can limit the outputs when they are high. Based on how *sigmoid* and *tanh* functions are defined, no matter how much the input grows, the activation will be the same. That may be solved by changing

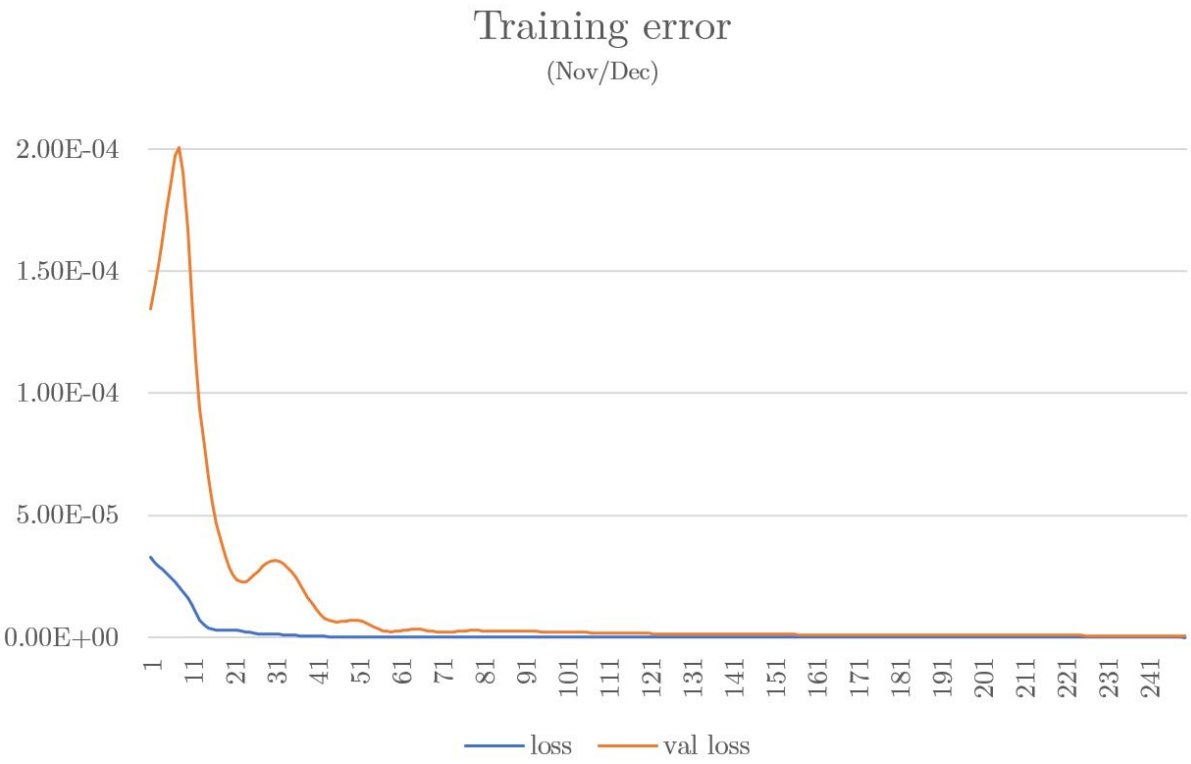
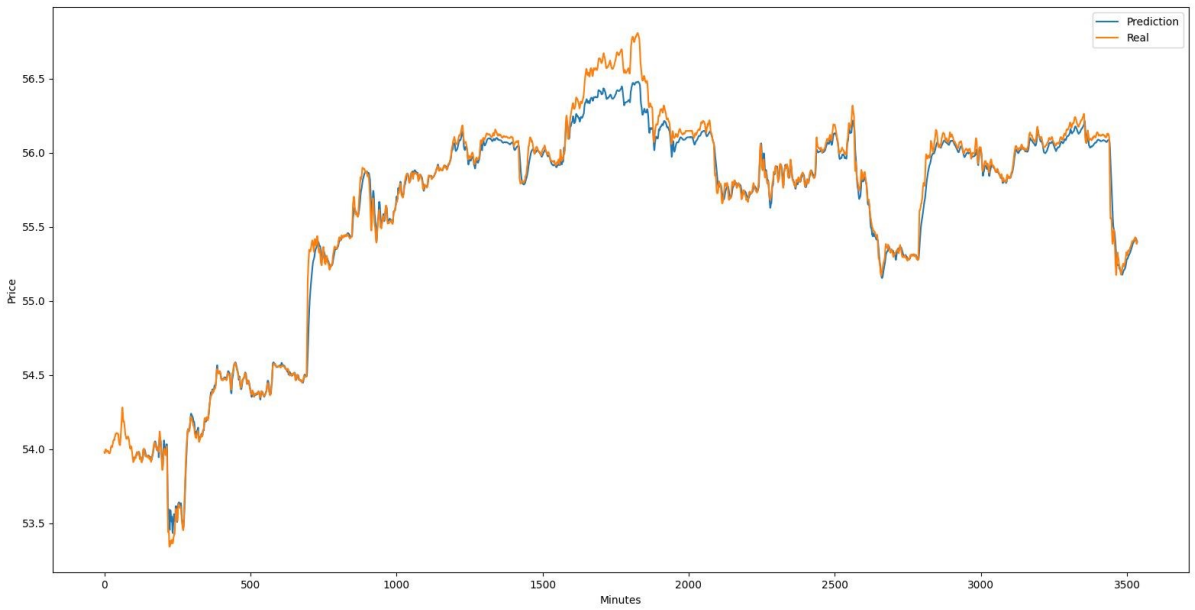
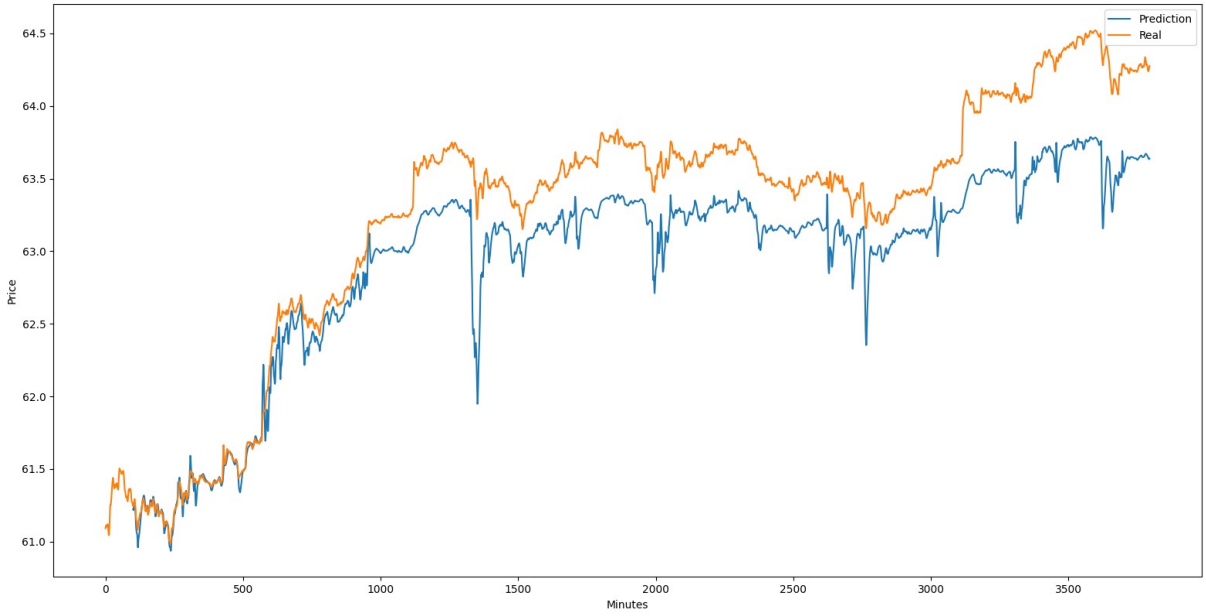


Figure 18: PyTorch training results for Nov/Dec model.



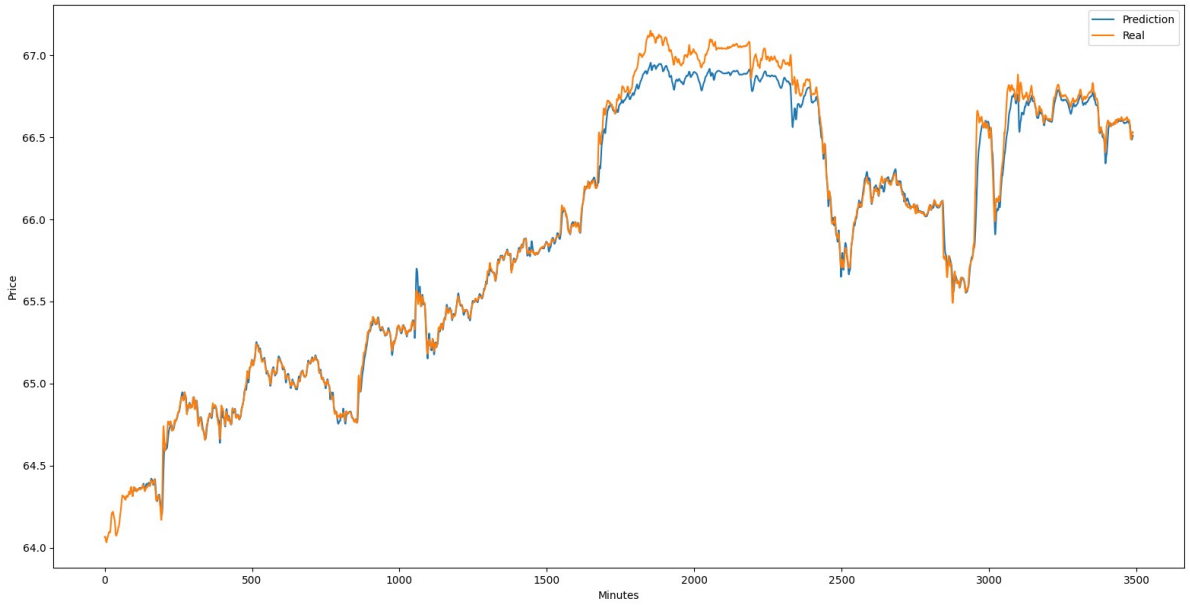
RMSE	Direction Errors	Direction Accuracy
0.0782897101612149	948/3435	72.40174672489083 %

Figure 19: PyTorch performance on validation dataset (1st week of October) for Sept/Oct model.



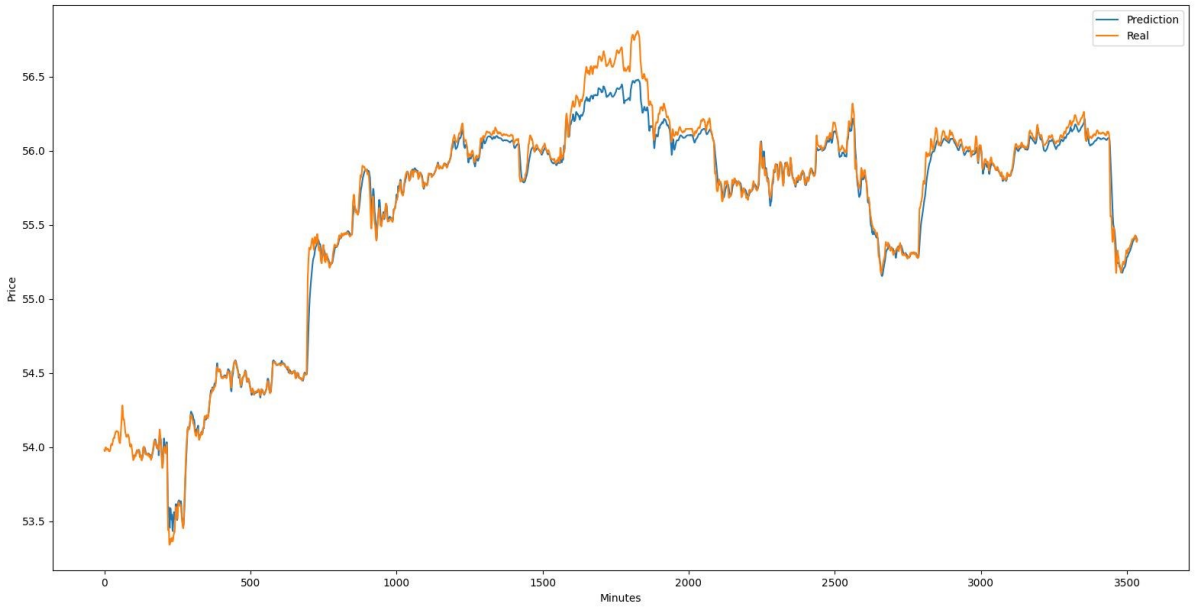
RMSE	Direction Errors	Direction Accuracy
0.40694093763175887	1078/3695	70.8254397834912 %

Figure 20: PyTorch performance on validation dataset (1st week of November) for Oct/Nov model.



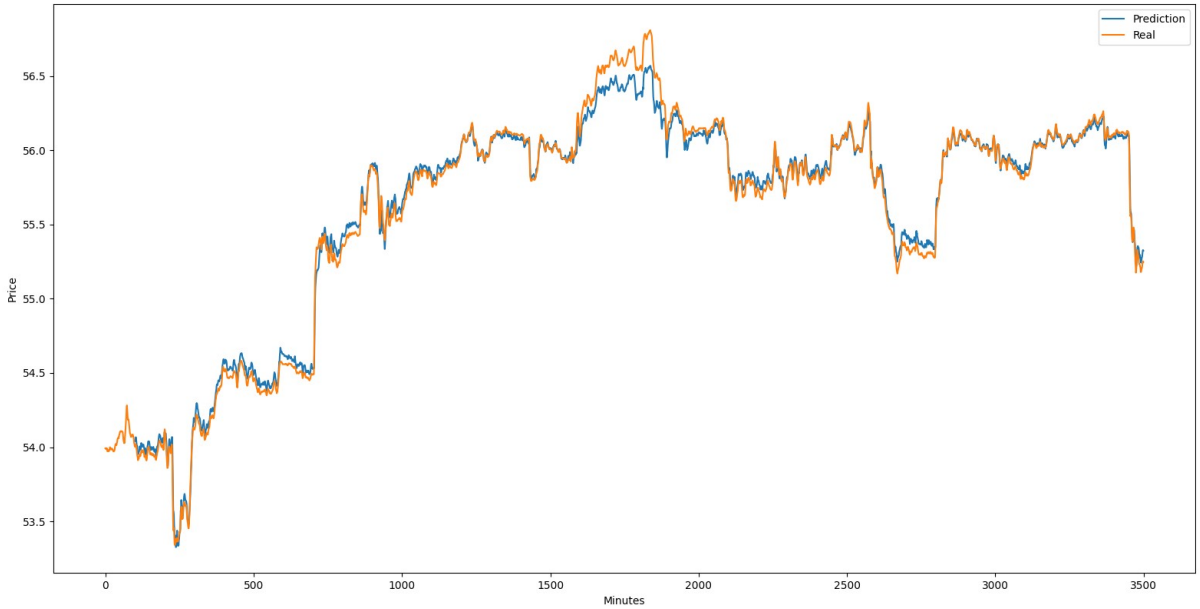
RMSE	Direction Errors	Direction Accuracy
0.07433484873319383	930/3388	72.55017709563164 %

Figure 21: PyTorch performance on validation dataset (1st week of December) for Nov/Dec model.



RMSE	Direction Errors	Direction Accuracy
0.0782897101612149	948/3435	72.40174672489083 %

(a) PyTorch Sept/Oct model



RMSE	Direction Errors	Direction Accuracy
0.06562594422868856	792/3435	76.94323144104804 %

(b) TensorFlow model

Figure 22: Performance comparison between TF model and PyTorch Sept/Oct model for the same validation data.

the LSTM architecture and use another activation function. *ReLu* for example, behaves linearly, this means that at a higher input value the activation will also be higher. However, investigating this issue was considered out of the scope of the project.

The results of the models, in general lines, have been satisfactory. In fact, for the prediction of 1 minute ahead, although the predicted prices have a certain margin of error, the correctness in direction of the predictions is more than 70%, which is far better than tossing a coin. If we take these results literally, we have a great statistical advantage over the market. However, the utility of these models on real market operations needs to be questioned, as we are predicting 1 minute ahead; which is a very short period of time. As previously mentioned, other models regarding a greater number of predictions, like trying to predict 10 minutes ahead, were not so good. In fact, the complexity and the randomness of the market, makes that those models provided just a little percentage of advantage against someone that is tossing a coin. In any case, the volatility of the market and the influence of external factors, make the results to be taken with caution. As we have seen, when the model collides with a section with very abrupt changes, it is not able to react adequately. That been said, we can conclude the results are satisfactory but not strictly representative.

8.3 Results improvement

With the intention of improving the results, multiple hypotheses were tested. One of the most relevant ones, although rejected, is going to be explained below as it consumed a significant part of the development effort. Others, like the already mentioned possible change of activation functions or the error function customization were out of the scope of the project but considered as potential improvements of the models results (Section 10.1).

Let us suppose a model that, from a 100 minutes time step, generates an output prediction of 10 minutes. Based on this and following a normal prediction methodology, our prediction will be generated from the last 100 minutes data sequence. That is, we are limiting our results to the prediction that the model makes for the last data sequence. After analysing this situation, it was concluded that this way of predicting could be improved, since certain predictions that could support the final prediction are being wasted. Just as the president of a country does not make decisions by just himself, it is preferable that our system collects the "opinions" of the other sequences in order to make a decision. To explain this system, it is essential to refer to the time-series format on which the operation of our models is based. In Figure 23, we can appreciate the superposition of the predictions in which our methodology is based. The idea is to take the last 10 (in this case is 10, but it refers to the number of predictions of the model) predictions and use the “portion” of the prediction that superposes with the last prediction (the one we are interested in) in order to generate the final “collaborative prediction”. That is, instead of just taking the prediction of the last time sequence, we generate a prediction based on the participation of all the previous predictions that overlap the final one. In fact, the participation is not made in equal parts, but each overlapping portion has a weight. The weight changes based on the proximity to the final sequence, which would be the most relevant. To sum up, the idea of this methodology is to reinforce the final prediction with the collaboration of the neighbouring predictions.

Although the idea seemed to make sense, results were not as expected. Not only it did not improve the predictions, but it slightly worsened them. Furthermore, assuming a higher performance cost since it requires a greater number of predictions, the idea was rejected.

9 Web application

In addition to the models and frameworks analysis, a simple web application prototype will be developed. The idea is to create something visual that can show the application of what has been done previously. That is, create a web application prototype that uses any of the previously developed models, in such a way that the user can interact with them and simulate a real prediction system. The prototype will consist of a simulation of a prediction system in real-time. The idea is that when users open the web application, they can view the market data at a certain time, and make a prediction for them.

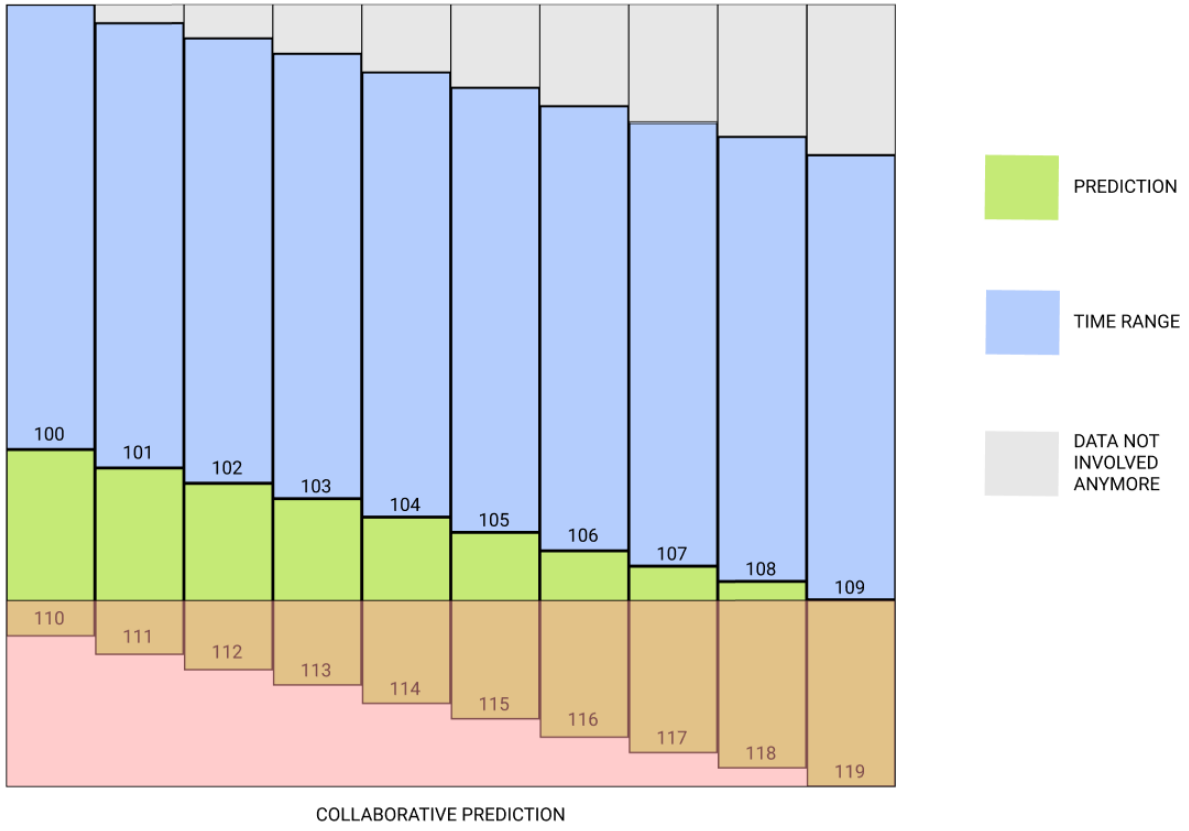


Figure 23: Collaborative prediction representation.

9.1 Analysis

9.1.1 Scope

The system will be called PredictionApp and so it will be referred to throughout the analysis.

PredictionApp is a prototype of a web system for making predictions based on stock market data. The system will consist of the interaction of a REST API, that will be referred as StockAPI, an external database and a web application, that will be referred as WebApp.

StockAPI will provide random dates and stock price data. On the other hand, WebApp will show the data provided and allow the user to make predictions on that simulated environment. With the use of PredictionApp, users will be able to interact with a web application that will simulate a real-time stock price prediction system.

9.1.2 Requirements

StockAPI

1. External interfaces
   - 1.1. Communication interface
     - RCI.1 The system must communicate with an external database to get market data.
     - RCI.2 The system must send all data in JSON[12] format.
2. Functional
   - 2.1. Get random date
     - RFDate.1 The system must allow users to retrieve a date.
       - RFDate.1.1 Date must be in the database.
       - RFDate.1.2 Date must be random.
       - RFDate.1.3 Date must be after the first $MARGIN\_FOR\_PREDICTION$ entries temporarily ordered.
         - RFDate.1.3.1 The constant $MARGIN\_FOR\_PREDICTION$ must be modifiable.
         - RFDate.1.3.2 The initial value for $MARGIN\_FOR\_PREDICTION$ is 1000.
       - RFDate.1.4 Date must follow MySQL $DATETIME$ format[13].
   - 2.2. Get prices
     - RFPrices.1 The system must allow users to retrieve price data.
       - RFPrices.1.1 The user must specify a date.
         - RFPrices.1.1.1 Date must follow RFDate.1.3 (Section 2.1.)
         - RFPrices.1.1.2 Date must follow RFDate.1.4 (Section 2.1.)
       - RFPrices.1.2 If RFPrices.1.1 (Section 2.2.) is fulfilled, data from the database must be returned to the user.
         - RFPrices.1.2.1 Data must be composed of $NUMBER\_OF\_MINUTES$ entries.
           - RFPrices.1.2.1.1 The constant $NUMBER\_OF\_MINUTES$ must be modifiable.
           - RFPrices.1.2.1.2 The initial value for $NUMBER\_OF\_MINUTES$ is 1000.
         - RFPrices.1.2.2 Data must start on date from RFPrices.1.1 (Section 2.2.) backwards.
3. Non-functional

[12]JSON: https://www.json.org/json-es.html
[13]MySQL Datetime Format: https://dev.mysql.com/doc/refman/5.7/en/datetime.html

3.1. Technological requirements
RNF-T.1 System must be implemented using Flask[14].
RNF-T.2 System must be containerized using Docker[15].

WebApp

1. External interfaces
   1.1. User interface
   RUI.1 The user interface must be displayed in English.
   1.2. Communication interface
   RCI.1 The system must communicate with an external API to get market data.
   RCI.1.1 The system must communicate through the HTTP protocol.
2. Functional
   2.1. Visualize data
   RFVisual.1 The system must allow users to visualize data.
   RFVisual.1.1 Date.
   RFVisual.1.1.1 Date must follow MySQL DATETIME format[16].
   RFVisual.1.2 Stock prices until the selected date on RFVisual.1.1 (Section 2.1.).
   RFVisual.1.2.1 Prices must be shown in dollars.
   2.2. Make predictions
   RPred.1 The system must allow users to make predictions.
   RFPred.1.1 The system must allow to make a 10-minutes prediction.
   RFPred.1.1.1 The system must show the prediction to the user by using a graph.
   RFPred.1.1.2 The system must show the direction of the prediction.
   RFPred.1.1.3 The system must show the final price of the prediction.
   RFPred.1.2 The system must allow to make a 1-minute prediction.
   RFPred.1.2.1 The system must show the direction of the prediction.
   RFPred.1.2.2 The system must show the direction of the prediction.
   RFPred.1.3 The system must allow to make a 60-minutes prediction.
   RFPred.1.3.1 The system must show the direction of the prediction.
   RFPred.1.4 The system should inform the user when the prediction process begins.
3. Non-functional
   3.1. Technological requirements
   RNF-T.1 System must be implemented using Flask[14].
   RNF-T.2 System must be containerized using Docker[15].

9.1.3 Diagrams

StockAPI For the use case diagram in Figure 24 we define two different use cases:

- Get random date: The user asks for a date to the system. The system picks a random date from the database. The system returns the date to the user.
- Get data: The user asks for price data. For that, the user must have introduced a specific date. If the date is correct, the system will return to the user a set of price data previous to that date.

Please, find the correspondent context diagram in Figure 25.

[14]Flask: https://flask.palletsprojects.com/en/1.1.x/
[15]Docker: https://www.docker.com/
[16]MySQL Datetime Format: https://dev.mysql.com/doc/refman/5.7/en/datetime.html

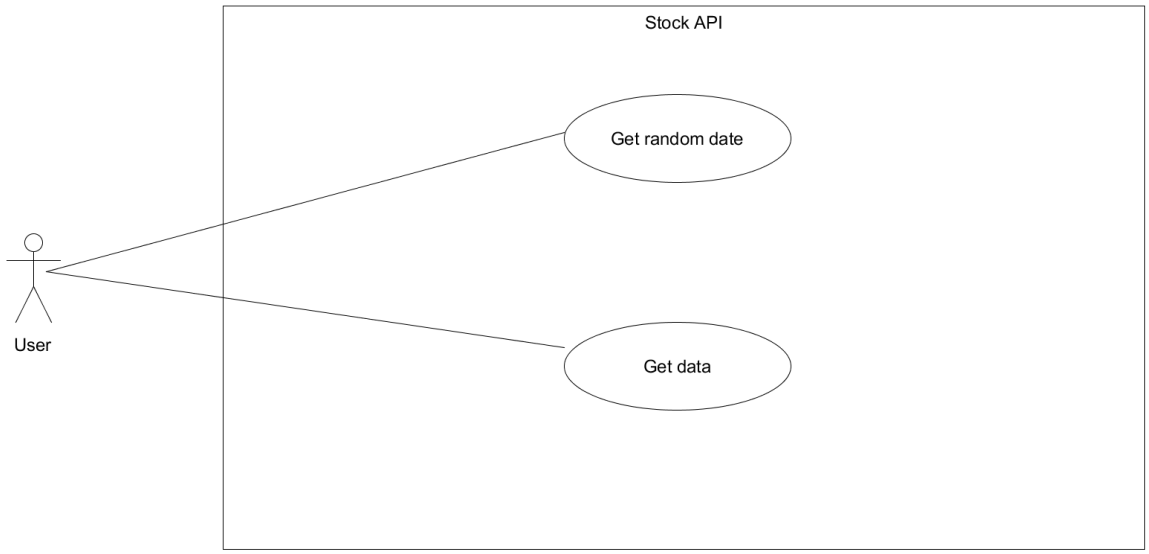


Figure 24: Use case diagram of StockAPI.

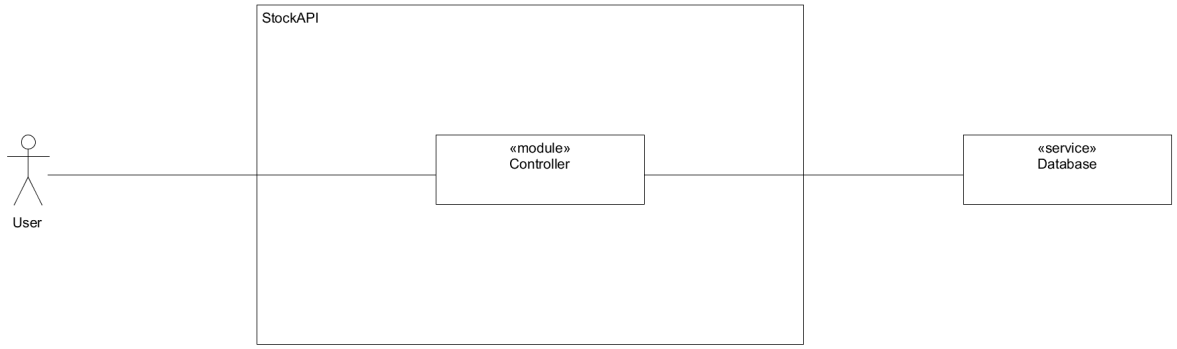


Figure 25: Context diagram of StockAPI.

WebApp For the use case diagram in Figure 26 we define two different use cases:

- Visualize data: The system will show some data to the user. Data includes the date and price data.
- Request prediction: The system will allow the user to make predictions.

Please, find the correspondent context diagram in Figure 27.

9.2 Design

In this chapter, the design of the PredictionApp will be considered.

9.2.1 Architecture

PredictionApp As we can see on the component diagram (Figure 28), the architecture of the system is composed of three main units:

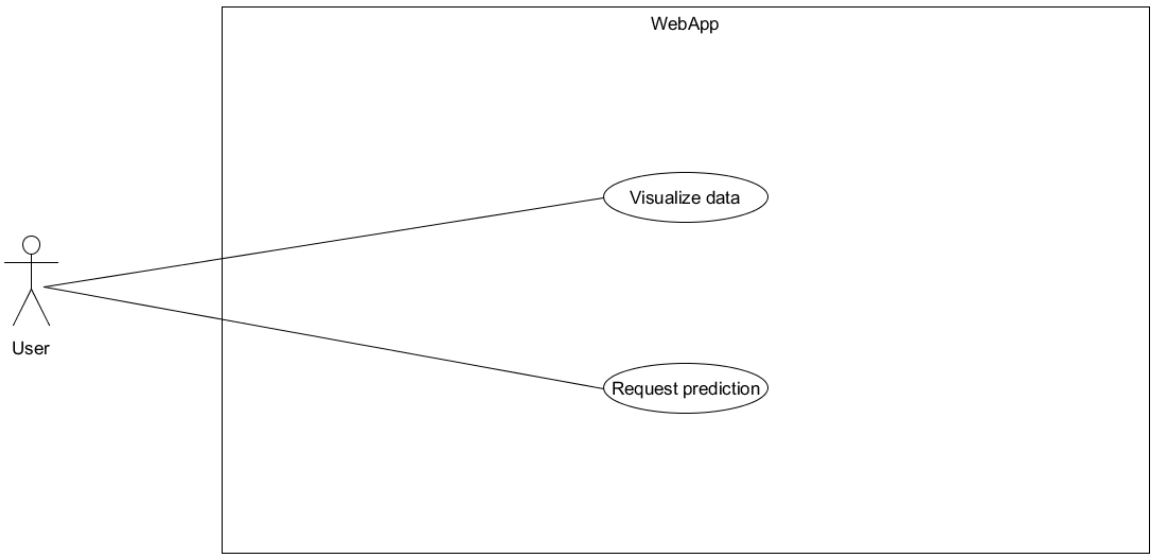


Figure 26: Use case diagram of WebApp.

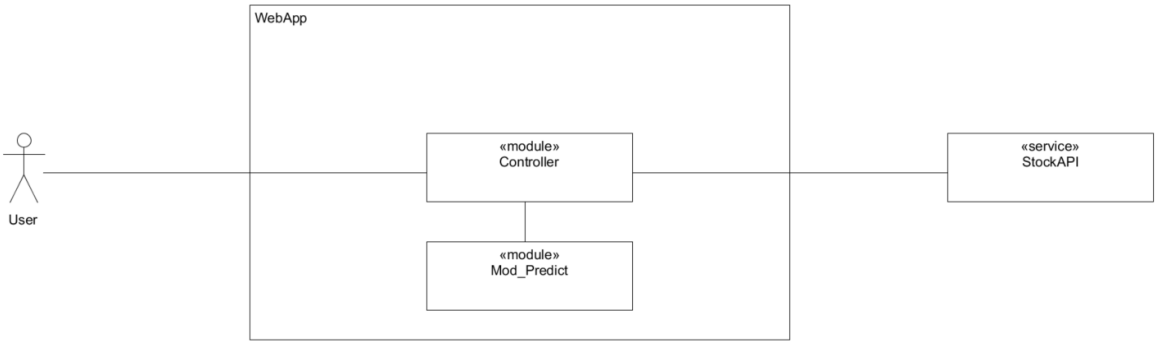


Figure 27: Context diagram of WebApp.

- WebApp: Represents the WebApp analysed in the requirements (Section 9.1.2). It interacts with the user (it is the system face for the user) and with the StockAPI component in order to request data.
- StockAPI: Represents the StockAPI analysed in the requirements (Section 9.1.2). It provides an API for accessing stock market data. It accesses a database service from AWS in order to get the data.
- AWS RelationalDatabaseSystem: It is the storage for the stock market data. It is a relational database part of Amazon Web Services.

In relation to the deployment diagram (Figure 29) we can highlight the following aspects:

- The StockAPI component will be on its own Docker container.
- The WebApp component will be on its own Docker container.
- Both, StockAPI and WebApp, are deployed on the same Universidad de Oviedo server.

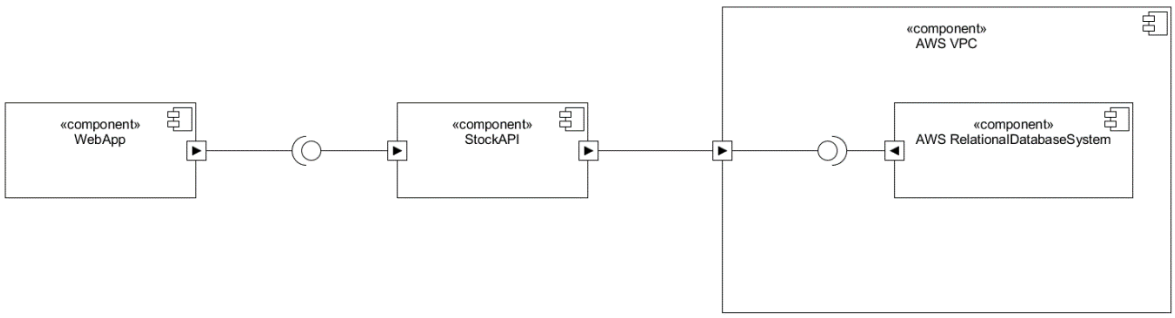


Figure 28: Component diagram of PredictionApp.

- Both, StockAPI and WebApp are able to communicate among them. Likewise, StockAPI will be able to communicate with the AWS service.
- The user will be able to access the PredictionApp system through the WebApp component. For that, the user (that should be inside the Universidad de Oviedo network) can use any browser to access the server IP (156.35.163.139) with the port 7000.

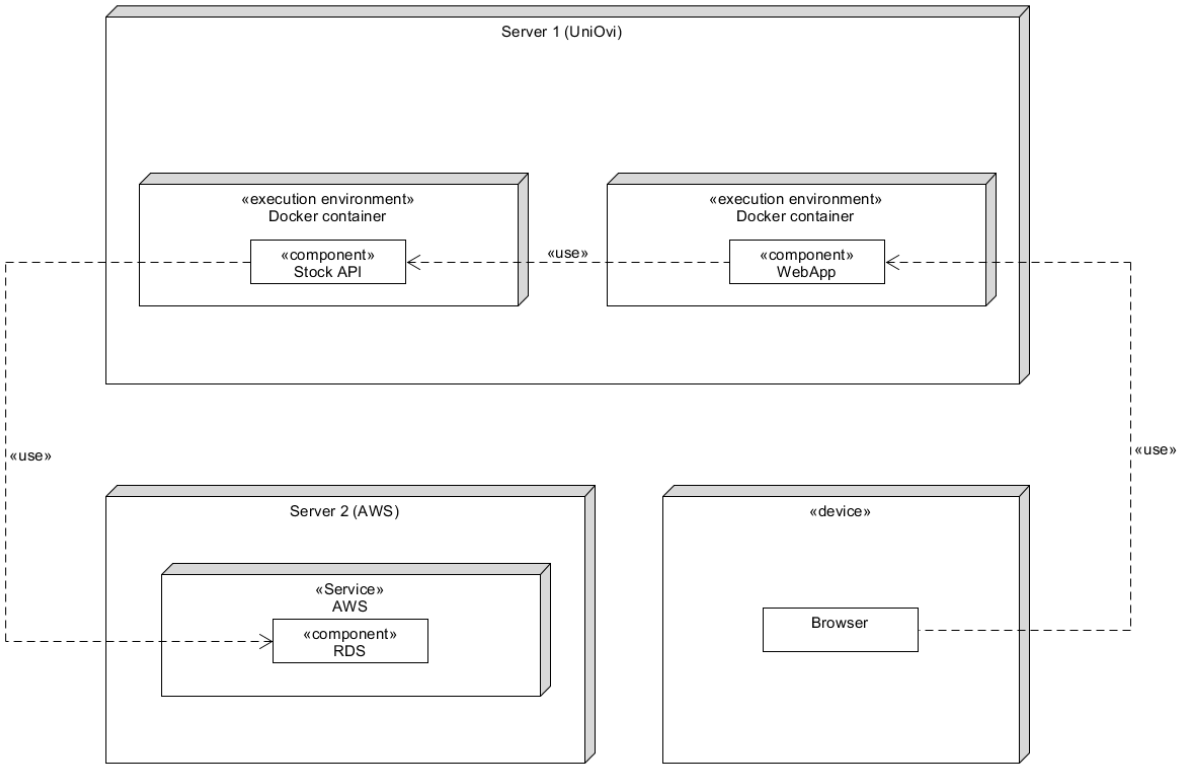


Figure 29: Deployment diagram of PredictionApp.

9.2.2 Class diagrams

StockAPI The StockAPI class diagram (Figure 30) is composed of the following elements:

- *controller*: It is the main unit StockAPI system, it acts as the Controller GRASP pattern and handles the two use cases of the system (Figure 24).
  - *stock_price_dao*: It regards the managements of stock price data received from the external database. It acts as a Data Access Object.
  - *input_validator*: It is an auxiliary component whose objective is to validate the input received in the controller.

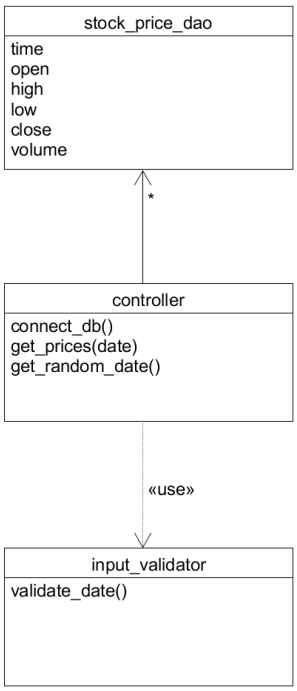


Figure 30: Class diagram of StockAPI.

WebApp The WebApp class diagram (Figure 31) is composed of the following elements:

- *controller*: It is the Controller (GRASP pattern) that handles the "Visualize data" use case (Figure 26).
- *controller_predict*: It is the Controller that handles the "Make prediction" use case (Figure 26).
  - *input_validator*: It is an auxiliary component whose objective is to validate the input received from the user.
  - *prediction_manager*: It provides the functionality to the *controller_predict* in order to make the prediction. It interacts with *prediction_data* to get the data to pass to the *Model* specific instance.
    - *prediction_data*: It is the responsible of getting and pre-processing the data from the specified date (that comes from the user).
      - *ta_indicators*: It is an auxiliary component that helps pre-processing the data by adding the technical indicators.

∗ *Model*: It is an abstract class that represents the models and provides all the necessary methods to make the desired prediction.

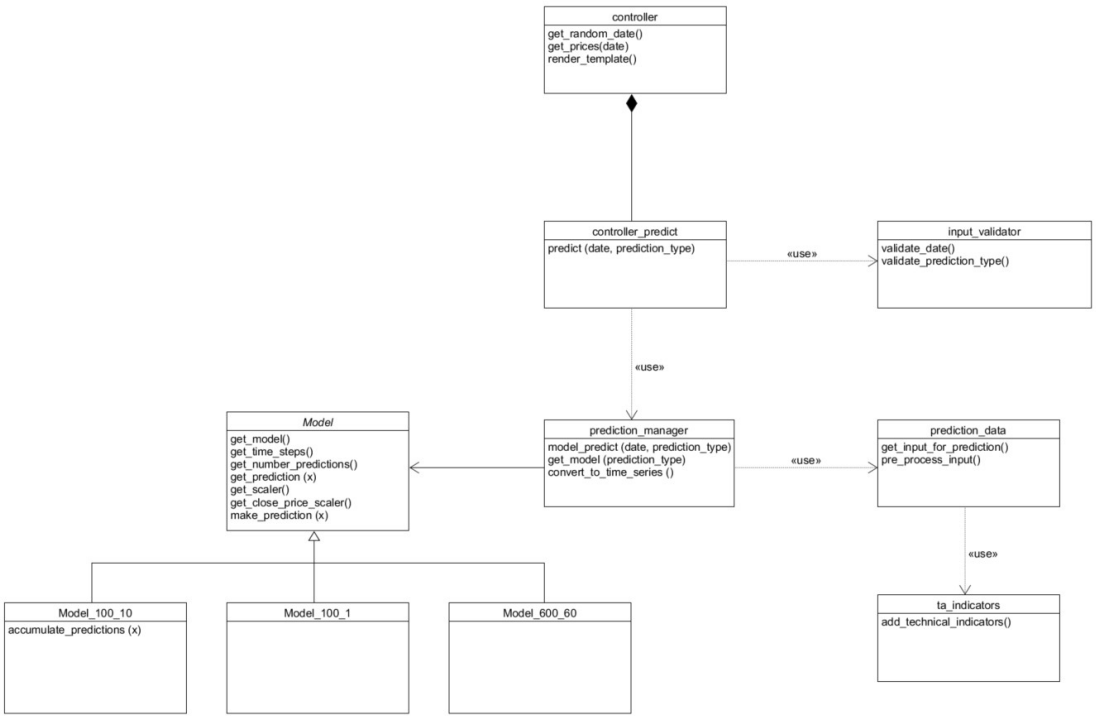


Figure 31: Class diagram of WebApp.

In order to manage the creation of models, the Factory design pattern is used (Figure 32). As a creational pattern, it provides one of the best ways to handle the logic for the instantiation of these objects.

The *ModelFactory* is in charge of choosing which object to instantiate from the following ones:

- *Model*_100_10: Class that represents the model created with time step = 100 and number of minutes predicted = 10. It uses an additional method *accumulate_predictions* as it shows the whole set of predictions of a data range, not a single prediction.
- *Model*_100_1: Class that represents the model created with time step = 100 and number of minutes predicted = 1.
- *Model*_600_60: Class that represents the model created with time step = 600 and number of minutes predicted = 60.

However, if needed, new models (new classes) could be added to the system without any impact. The design is made in such a way that the system can plug-in new models as desired.

9.3 Development

9.3.1 Programming languages

For the development of the system the following programming languages and libraries have been used:

- Python3

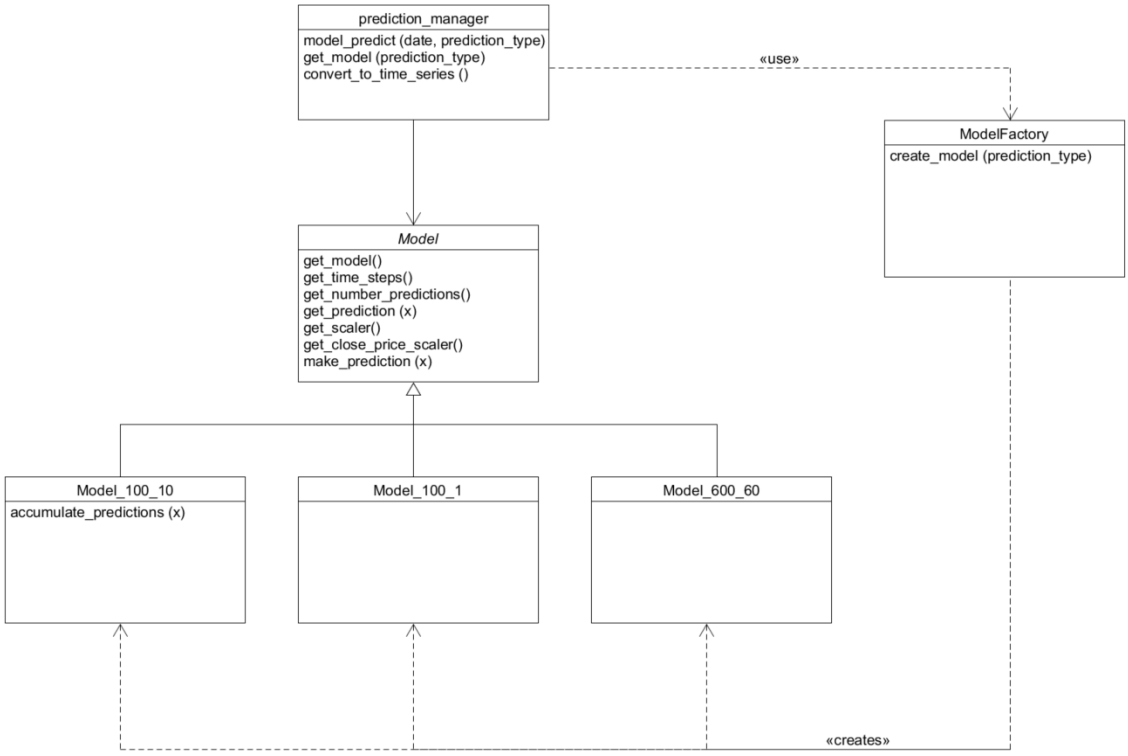


Figure 32: Class diagram of WebApp (Factory design pattern view).

  - Version: Python 3.7.9
  - Distribution: Anaconda
  - Virtual Environment: 20.2.2
  - Libraries:
    - Cryptography 3.3.1
    - Flask 1.1.2
    - Jinja2 2.11.2
    - Joblib 0.17.0
    - Numpy 1.18.5
    - Pandas 1.1.4
    - PyMySQL 0.10.1
    - Requests 2.25.0
    - Sklearn 0.0
    - Ta 0.7.0
    - TensorFlow 2.3.1 (Models used for the PredictionApp were TF models)
- JavaScript
  - Libraries:
    - jQuery 3.5.1
    - ChartJs 2.9.4
- HTML
- CSS

9.3.2 Tools

Next, the set of tools used for the development of the system:

- Visual Studio Code
  - Version: 1.52.1
  - Website: https://visualstudio.microsoft.com
  - Brief description: It is an integrated development environment (IDE) from Microsoft.
- Postman
  - Version: 7.36.1
  - Website: https://www.postman.com/
  - Brief description: Postman is a software development tool that enables to test calls to APIs.
- MySQL Workbench
  - Version: 8.0.17
  - Website: https://www.mysql.com/products/workbench/
  - Brief description: MySQL Workbench is a visual database design tool that integrates SQL development, administration, database design, creation and maintenance into a single integrated development environment for the MySQL database system.
- Git
  - Version: 2.29.2.windows.2
  - Website: https://git-scm.com/
  - Brief description: Git is a free and open source distributed version control system.
- GitHub
  - Website: https://github.com/
  - Brief description: GitHub is a code hosting platform for version control and collaboration.
- AWS Console
  - Website: https://aws.amazon.com/console/
  - Brief description: AWS Console is a web application that comprises and refers to a broad collection of service consoles for managing Amazon Web Services.
- Docker
  - Version: 20.10.2
  - Website: https://www.docker.com/
  - Brief description: Docker is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files.

9.3.3 Deployment

For the deployment of the application, it has been decided to use Docker. Docker uses OS-level virtualization to deliver software in packages called containers. These containers, as they contain the libraries and software already installed, are ideal for deploying the application anywhere and not having any kind of problem with software versions. They can be seen as a much lighter version of a virtual machine.

For our system we have defined two different images, one for StockAPI and another for WebApp. Both must be able to interact with each other, so *dockercompose* has been used for this purpose. Docker Compose is a tool for defining and running multi-container Docker applications. We use a YAML file to configure our application's containers and then, with a single command, we create and start all the services from our configuration. One of the issues to take into account when using containers could have been affected by the containerization of the REST API, which works with a database to provide the data to the requests. Containers, due to their stateless nature, do not allow us to save the contents of the database without recurring to an external storage. That is, the content of the database, disappears every time the container is "turned off". It is because of this issue, that during the design it was decided to resort to a database hosted on some server accessible through Internet.

10 Conclusions and possible extensions

Throughout the project we have worked with two of the most popular libraries in the field of deep learning. Not in vain they are recognized in this way, since they have provided us with an immense work capacity at a relatively low learning cost.

Regarding TensorFlow, the incorporation of Keras high-level API could not have been a more successful decision. Nowadays, from TensorFlow2, it is common to talk about Keras and TensorFlow indistinctly because of their effective collaboration for building such an intuitive framework. In fact, as we have seen during the project, it can be so high-level that we could build a stacked LSTM model without even needing to understand its internal mechanics; not as in the case of PyTorch. However, on the other side of the coin, far from the pythonic high-level approach, TF stills keeps the possibility of building the computational graph statically, as it was in TF1. Some may prefer this approach because it can provide some advantages like performing transformation operations directly on the graph, independence of the programming language or reinforcement on the design. This may come with the cost of a more complex process, specially if the developer is not familiarized with graphs, but it can also be beneficial.

In addition to the model design/implementation features, TensorFlow comes with a lot of tools that help developers with integration and development issues. In fact, there are integrated services on AWS, like Amazon Sagemaker,[17] used for creating scaled TensorFlow models in AWS platform; or in the same Google, like TensorFlow Cloud[18], that helps the integration of the local environment in Google Cloud. All these extra features contribute to the power and versatility of TensorFlow and prove its well-known dominance in production environments.

On the other side, PyTorch provides a quite different approach specially for the development of the model API. Even though both libraries share a lot of features, as they both are working hard to adapt to the market needs, PyTorch is slightly falling into a different public than TF. As we analysed in Section 7.2, PyTorch is more attractive to the research field as it provides more flexibility, among other things. That flexibility is due to a different design which does not bet for a very high-level API that hides a lot of functionality, but for a less high-level approach that supply a very powerful tool that just help users exploit their knowledge. As we could see during the project, for the PyTorch stacked LSTM model, it was strictly necessary to know the internal behaviour of an LSTM network in order to successfully build it. Apart from this, PyTorch also provides some tools for production, but not in the same level as TF. That is why, its less mature state for production uses, makes it less preferable for big deployments in the industry.

[17] AWS TensorFlow: https://aws.amazon.com/tensorflow/
[18] Google Cloud: https://www.tensorflow.org/cloud

To sum up, TensorFlow stands out for its immensity and its ability to create models with a very simple and intuitive syntax. On the other hand, PyTorch provides a service with a slightly higher initial learning curve but with a flexibility that captures most researchers. Although each library has its advantages and disadvantages, both struggle to adapt to a community with similar interests. In fact, we saw how PyTorch was a pioneer compared to TensorFlow in the use of dynamic state graphs, achieving, above all, certain usability advantages. This was later added by TensorFlow, adapting to the needs of the community; who welcomed this improvement with open arms. Therefore, although both have their differences in certain approaches, both fight for a common goal: to bring the world of machine learning closer to the community.

Regarding the field in which the models were applied, the complexity surrounding the stock market was demonstrated. Right from the start there was awareness of the difficulty of achieving optimal results. In any case, without professional resources, time to carry out a deep investigation and not being the only objective of the project, the results achieved are satisfactory.

The first approach, the whole 2019 dataset with TF, showed good results for the first section of the validation data. However, it was severely cancelled as soon as a sudden strong rise in the price appeared. This may be for multiple reasons that have been already analysed, but it was an interesting question to experiment with. That is why, the approach for the PyTorch models was totally different.

The results of both solutions (ignoring the cancelled section for TF) provided more than 70% of accuracy in predicting the direction of the next minute price. The TF model was even near to the 80% with an accuracy of 76.30%. However, they were not so good in the precision of predicting the exact price, whose RMSEs were less impressive. Regarding the lightweight models (the PyTorch ones) approach, it was concluded that they provide a greater capacity of overcoming with possible sudden changes in the price, but at the cost of losing some precision. However, they kept the accuracy in direction above the 70%.

Having said that, and according to the premise of the complexity of the market, all models coincide in their weakness in the face of very abrupt movements in the price. Due to the lack of this ability to react to all types of external factors, and above all, due to the intrinsic stochasticity of the market, achieving a model that defines accurately market movements is really challenging, if possible. However, as we have experienced in this project, deep learning guided by powerful libraries like TensorFlow or PyTorch, provides a very good starting point.

10.1 Possible extensions

Activation functions on LSTM layers As we have seen throughout the development of the models, when the price suddenly reached historical highs, the model was not able to make the correct predictions. One of the possible proposals is to study the change of the LSTM activation functions by default, which are *sigmoid* and *tanh*, for others that are not saturated as easily. One of the candidates could be *ReLU*, which keeps the non-linearity, but provides a linear behaviour. This possible change, with its consequences and difficulties, would be worth evaluating if we wanted to improve the system.

Adjustment of number of layers and nodes In order to optimize and obtain more robust models, the number of layers and nodes should be adjusted. For that reason, it is proposed that a future extension makes a deeper investigation so as to achieve the improvement of the results by, in this case, reducing the number of parameters of the network.

Custom loss function In order to improve the performance of the models when predicting prices, one of the possible improvements would be to create a custom loss function that take into account more factors than the difference of value between prices. The idea is that the custom loss function evaluates the hit rate in the direction of the prediction; as in the end, it is the most important factor.

Greater investment in web prototype The web system that has been developed has been one more part of the project, not the main one. Therefore, the dedication that has been devoted to its analysis,

design and development has been reduced. Both the improvement of the design at an architectural level, the development of tests and the creation of new functionalities, would be necessary if we wanted to carry out a web engineering project.

11 Final project planning and budget

The project was finally completed after 350 hours of work; 50 hours more than expected. That imprecision in the estimation affected the proportions of work of each of the high-level tasks, as we can see in Table 12.

Task	Planned hours	%	Real hours	%
Initial studies	20	6.67%	20	5.71%
Models development	177	59.00%	152	43.43%
Web application prototype building	53	17.67%	78	22.29%
Project documentation development	50	16.67%	100	28.57%

Table 12: Comparison between estimated and real hours of work to complete each task of the project.

These extra hours of work caused that instead of completing the project for the expected end date, which was the 4th June of 2021, it was completed the 12th June of 2021. That last period of the project required to double the estimated weekly hours of work in order to have it for the goal date.

11.1 Summarized budget

The summarized final budget can be found in Table 13. For a more detailed version of the final budget, please check Appendix 12.3.

Item	Description	Total
01	Initial studies	899.35 €
02	Models development	6,835.03 €
03	Web application prototype building	3,507.45 €
04	Project documentation development	4,496.73 €
TOTAL		**15,738.55 €**

Table 13: Summarized final budget.

12 Appendix

12.1 Glossary and abbreviations

Below, all the terms (in alphabetical order) whose clarification may be of help for a better understanding of the document:

- ANN: Artificial Neural Network.
- API: Application Programming Interface.
- AWS: Amazon Web Services.
- CNN: Convolutional Neural Network.
- CPU: Central Processing Unit.
- GPU: Graphics Processing Unit.
- GRASP: General Responsibility Assignment Software Patterns.
- IDE: Integrated Development Environment.
- LSTM: Long-Short Term Memory.
- MSE: Mean Square Error.
- PBS: Product Breakdown Structure.
- RMSE: Root Mean Square Error.
- RNN: Recurrent Neural Network.
- TF: TensorFlow.
- WBS: Work Breakdown Structure.

12.2 Initial budget

In this appendix, the estimated budget for the realization of the project is presented. We begin with a study of the context of the company to obtain certain values such as billing needs or workforce price/hour. After that, we proceed with the cost budget and its respective client budget.

12.2.1 Company context

Staff cost The salary cost for each company profile is presented below (Table 14). To calculate the salary cost per year, the social security contributions have been added to the annual gross salary. That contribution percentage has been established as a 31% and it has been calculated based on the rate for the contribution by accident work adapted for computer scientists.

Staff	Annual Gross Salary (€)	Annual Salary Cost	TOTAL
Software Engineer	33,000.00 €	43,362.00 €	43,362.00 €
TOTAL			**43,362.00 €**

Table 14: Staff cost.

Staff	Annual Salary Cost	Prod (%)	Direct Cost	IC (%)	Indirect Cost
Software Engineer	43,362.00 €	90.00%	39,025.80 €	10.00%	4,336.20 €
TOTAL	43,362.00 €		**39,025.80 €**		**4,336.20 €**

Table 15: Staff productivity.

Staff productivity Once we have calculated the expenses of each company profile, we proceed to calculate their productivity (Table 15), to differentiate between direct and indirect costs. The productivity percentages have been established considering the involvement in the project of each of the profiles.

Indirect company costs In addition to the indirect costs of the staff, the indirect costs of the company must be taken into account, together with the indirect costs of production (Table 16 and 17).

Service	Monthly cost	Annual cost
Cleaning	100.00 €	1,200.00 €
Advisory	300.00 €	3,600.00 €
Rental of real estate (office)	300.00 €	3,600.00 €
Maintenance, repair and conservation expenses	50.00 €	600.00 €
Electricity consumption	40.00 €	480.00 €
Water consumption	20.00 €	240.00 €
Travel expenses	100.00 €	1,200.00 €
Public relationships and adds expenses	100.00 €	1,200.00 €
Office equipment expenses	30.00 €	360.00 €
Communications and postal expenses	10.00 €	120.00 €
Financial expenses	50.00 €	600.00 €
TOTAL		**13,200.00 €**

Table 16: Indirect company costs.

Device/License	Units	Price	Annual Cost	Type	Term (years)
Laptop MSI GL62M	1	850.00 €	170.00 €	Amortization	5
Windows License	1	10.00 €	2.00 €	Rent	5
Microsoft Office 365 License	1	6.96€/month	83.52 €	Rent	-
Microsoft Project Plan 3 License	1	30.62€/month	367.44 €	Rent	-
Training server (32GB RAM 8 cores)	1	102.85€/month	1,234.20 €	Rent	-
TOTAL			**1,857.16 €**		

Table 17: Production assets costs.

Turnover needs After adding the indirect costs (both staff and company) and direct costs, we apply the 25% profit. Now, if we add the benefit to the costs, we obtain the company's turnover needs (Table 18).

Price/Hour To calculate the price/hour of the company staff, the gross hourly wage needs to be multiplied by a factor until turnover needs are met. In our case, we have multiplied the salary by a factor of 2.5 to obtain the final price/hour that covers the company's needs. Subsequently, a 25% discount is applied on the price/hour to calculate the one without benefits, which will be the prices that are used for the staff in the cost budget.

Summary In Table 20, we can find a summary of the company situation.

Direct costs	39,025.80 €
Indirect costs	19,393.36 €
Earnings (25%)	14,604.79 €
TOTAL	**73,023.95 €**

Table 18: Company turnover needs.

Staff	Price/Hour (Client)	Productive hours (Company total)	Turnover
Software Engineer	47.97 €	1548	74,250.00 €
TOTAL		1548	**74,250.00 €**

Table 19: Client Price/Hour.

Nº	Concept	Amount
1	Total direct costs	39,025.80 €
2	Total indirect costs	19,393.36 €
3	Direct and indirect costs sum	58,419.16 €
4	Desired earnings (25%)	14,604.79 €
5	Total cost	73,023.95 €
6	Expected turnover	74,250.00 €
7	Margin between cost and turnover	1.65%

Table 20: Company situation summary.

12.2.2 Costs budget

Next, all the items for the cost budget are presented.

Item 1 Initial studies (Table 21).

L1	L2	Description	Amount	Units	Price	Subtotal (2)	Total
01		Deep learning study					215.84 €
	001	Software Engineer	6	hours	35.97 €	215.84 €	
02		Neural networks study					251.82 €
	002	Software Engineer	7	hours	35.97 €	251.82 €	
03		Deep learning applied to stock market study					251.82 €
	003	Software Engineer	7	hours	35.97 €	251.82 €	
TOTAL							**719.48 €**

Table 21: "Initial studies" item costs.

Item 2 Models development (Table 22).

Item 3 Web application prototype building (Table 23).

Item 4 Project documentation development (Table 24).

Cost budget Grouping all budget items, we make up the cost budget (Table 25).

L1	L2	L3	Description	Amount	Units	Price	Subtotal (2)	Subtotal (3)	Total
01	001		TensorFlow and PyTorch study					1,079.22 €	1,079.22 €
		01	Software Engineer	30	hours	35.97 €	1,079.22 €		
02			Dataset preparation						899.35 €
	001		Dataset research					179.87 €	
		01	Software Engineer	5	hours	35.97 €	179.87 €		
	002		Cleaning and improvement of dataset					719.48 €	
		01	Software Engineer	20	hours	35.97 €	719.48 €		
03	001		Neural network configuration study					1,079.22 €	1,079.22 €
		01	Software Engineer	30	hours	35.97 €	1,079.22 €		
04			Models implementation						3,309.59 €
	001		TensorFlow model implementation					719.48 €	
		01	Software Engineer	20	hours	35.97 €	719.48 €		
	002		TensorFlow model testing and improvement					935.32 €	
		01	Software Engineer	26	hours	35.97 €	935.32 €		
	003		PyTorch model implementation					719.48 €	
		01	Software Engineer	20	hours	35.97 €	719.48 €		
	004		PyTorch model testing and improvement					935.32 €	
		01	Software Engineer	26	hours	35.97 €	935.32 €		
TOTAL									**6,367.37 €**

Table 22: "Models development" item costs.

L1	L2	L3	Description	Amount	Units	Price	Subtotal (2)	Subtotal (3)	Total
01			Web application analysis						143.90 €
	001		Requirements specification					71.95 €	
		01	Software Engineer	2	hours	35.97 €	71.95 €		
	002		Use case specification					71.95 €	
		01	Software Engineer	2	hours	35.97 €	71.95 €		
02			Web application design						323.76 €
	001		Architecture definition					179.87 €	
		01	Software Engineer	5	hours	35.97 €	179.87 €		
	002		Class diagrams definition					143.90 €	
		01	Software Engineer	4	hours	35.97 €	143.90 €		
03			Web application development						1,438.95 €
	001		Backend development					1,079.22 €	
		01	Software Engineer	30	hours	35.97 €	1,079.22 €		
	002		Frontend development					359.74 €	
		01	Software Engineer	10	hours	35.97 €	359.74 €		
TOTAL									**1,906.61 €**

Table 23: "Web application prototype building" item costs.

L1	L2	Description	Amount	Units	Price	Subtotal (2)	Total
01		Project documentation development					1,798.69 €
	001	Software Engineer	50	hours	35.97 €	1,798.69 €	
TOTAL							**1,798.69 €**

Table 24: "Project documentation development" item costs.

Item	Activity	Description	Subtotal (2)	Total
01		Initial studies		719,48 €
	01	Deep learning study	215,84 €	
	02	Neural network study	251,82 €	
	03	Deep learning applied to stock market study	251,82 €	
02		Models development		6.367,37 €
	01	TensorFlow and PyTorch study	1.079,22 €	
	02	Dataset preparation	899,35 €	
	03	Neural network configuration study	1.079,22 €	
	04	Models implementation	3.309,59 €	
03		Web application prototype building		1.906,61 €
	01	Web application analysis	143,90 €	
	02	Web application design	323,76 €	
	03	Web application development	1.438,95 €	
04		Project documentation development		1.798,69 €
	01	Project documentation development	1.798,69 €	
TOTAL				**10.792,15 €**

Table 25: Estimated cost budget.

12.2.3 Client budget

Client detailed budget Table 26.

Item	Activity	Description	Subtotal (2)	Total
01		Initial studies		899.35 €
	01	Deep learning study	269.80 €	
	02	Neural network study	314.77 €	
	03	Deep learning applied to stock market study	314.77 €	
02		Models development		7,959.21 €
	01	TensorFlow and PyTorch study	1,349.02 €	
	02	Dataset preparation	1,124.18 €	
	03	Neural network configuration study	1,349.02 €	
	04	Models implementation	4,136.99 €	
03		Web application prototype building		2,383.27 €
	01	Web application analysis	179.87 €	
	02	Web application design	404.71 €	
	03	Web application development	1,798.69 €	
04		Project documentation development		2,248.36 €
	01	Project documentation development	2,248.36 €	
TOTAL				**13,490.19 €**

Table 26: Client detailed budget.

Client summarized budget Table 27.

Item	Description	Total
01	Initial studies	899.35 €
02	Models development	7,959.21 €
03	Web application prototype building	2,383.27 €
04	Project documentation development	2,248.36 €
TOTAL		**13,490.19 €**

Table 27: Client summarized budget.

12.3 Final budget

In this appendix, the final budget for the realization of the project is presented. It shares the same company context as the Initial budget (Section 12.2), so it will not be repeated. That been said, let us proceed with the cost budget and its respective client budget.

12.3.1 Costs budget

Next, all the items for the cost budget are presented.

Item 1 Initial studies (Table 28).

L1	L2	Description	Amount	Units	Price	Subtotal (2)	Total
01		Deep learning study					215.84 €
	001	Software Engineer	6	hours	35.97 €	215.84 €	
02		Neural networks study					251.82 €
	002	Software Engineer	7	hours	35.97 €	251.82 €	
03		Deep learning applied to stock market study					251.82 €
	003	Software Engineer	7	hours	35.97 €	251.82 €	
TOTAL							**719.48 €**

Table 28: "Initial studies" final budget item costs.

Item 2 Models development (Table 29).

Item 3 Web application prototype building (Table 30).

Item 4 Project documentation development (Table 31).

Cost budget Grouping all budget items, we make up the cost budget (Table 32).

L1	L2	L3	Description	Amount	Units	Price	Subtotal (2)	Subtotal (3)	Total
01	001		TensorFlow and PyTorch study					359.74 €	359.74 €
		01	Software Engineer	10	hours	35.97 €	359.74 €		
02			Dataset preparation						575.58 €
	001		Dataset research					323.76 €	
		01	Software Engineer	9	hours	35.97 €	323.76 €		
	002		Cleaning and improvement of dataset					251.82 €	
		01	Software Engineer	7	hours	35.97 €	251.82 €		
03	001		Neural network configuration study					683.50 €	683.50 €
		01	Software Engineer	19	hours	35.97 €	683.50 €		
04			Models implementation						3,849.20 €
	001		TensorFlow model implementation					719.48 €	
		01	Software Engineer	20	hours	35.97 €	719.48 €		
	002		TensorFlow model testing and improvement					1,618.82 €	
		01	Software Engineer	45	hours	35.97 €	1,618.82 €		
	003		PyTorch model implementation					359.74 €	
		01	Software Engineer	10	hours	35.97 €	359.74 €		
	004		PyTorch model testing and improvement					1,151.16 €	
		01	Software Engineer	32	hours	35.97 €	1,151.16 €		
TOTAL									**5,468.02 €**

Table 29: "Models development" final budget item costs.

L1	L2	L3	Description	Amount	Units	Price	Subtotal (2)	Subtotal (3)	Total
01			Web application analysis						143.90 €
	001		Requirements specification					71.95 €	
		01	Software Engineer	2	hours	35.97 €	71.95 €		
	002		Use case specification					71.95 €	
		01	Software Engineer	2	hours	35.97 €	71.95 €		
02			Web application design						323.76 €
	001		Architecture definition					179.87 €	
		01	Software Engineer	5	hours	35.97 €	179.87 €		
	002		Class diagrams definition					143.90 €	
		01	Software Engineer	4	hours	35.97 €	143.90 €		
03			Web application development						2,338.30 €
	001		Backend development					1,618.82 €	
		01	Software Engineer	45	hours	35.97 €	1,618.82 €		
	002		Frontend development					719.48 €	
		01	Software Engineer	20	hours	35.97 €	719.48 €		
TOTAL									**2,805.96 €**

Table 30: "Web application prototype building" final budget item costs.

L1	L2	Description	Amount	Units	Price	Subtotal (2)	Total
01		Project documentation development					3,597.38 €
	001	Software Engineer	100	hours	35.97 €	3,597.38 €	
TOTAL							**3,597.38 €**

Table 31: "Project documentation development" final budget item costs.

Item	Activity	Description	Subtotal (2)	Total
01		Initial studies		719,48 €
	01	Deep learning study	215,84 €	
	02	Neural network study	251,82 €	
	03	Deep learning applied to stock market study	251,82 €	
02		Models development		5.468,02 €
	01	TensorFlow and PyTorch study	359,74 €	
	02	Dataset preparation	575,58 €	
	03	Neural network configuration study	683,50 €	
	04	Models implementation	3.849,20 €	
03		Web application prototype building		2.805,96 €
	01	Web application analysis	143,90 €	
	02	Web application design	323,76 €	
	03	Web application development	2.338,30 €	
04		Project documentation development		3.597,38 €
	01	Project documentation development	3.597,38 €	
TOTAL				**12.590,84 €**

Table 32: Final cost budget.

12.3.2 Client budget

Client detailed budget Table 33.

Item	Activity	Description	Subtotal (2)	Total
01		Initial studies		899.35 €
	01	Deep learning study	269.80 €	
	02	Neural network study	314.77 €	
	03	Deep learning applied to stock market study	314.77 €	
02		Models development		6,835.03 €
	01	TensorFlow and PyTorch study	449.67 €	
	02	Dataset preparation	719.48 €	
	03	Neural network configuration study	854.38 €	
	04	Models implementation	4,811.50 €	
03		Web application prototype building		3,507.45 €
	01	Web application analysis	179.87 €	
	02	Web application design	404.71 €	
	03	Web application development	2,922.87 €	
04		Project documentation development		4,496.73 €
	01	Project documentation development	4,496.73 €	
TOTAL				**15,738.55 €**

Table 33: Client detailed final budget.

Client summarized budget Table 34.

Item	Description	Total
01	Initial studies	899.35 €
02	Models development	6,835.03 €
03	Web application prototype building	3,507.45 €
04	Project documentation development	4,496.73 €
TOTAL		**15,738.55 €**

Table 34: Client summarized final budget.

12.4 Contents delivered

Below, a brief description of the structure of the content delivered. Please, note that all mentioned directories contain individual READMEs with indications. The highest level components of the content delivered are the following ones:

- models: Contains all the implementation regarding TensorFlow and PyTorch models.
- webapp_prototype: Contains all the implementation regarding PredictionApp (Section 9.1.1).
- README.txt: Contains a similar description as the current one.

References

[1] A. Amidi and S. Amidi, "Cs 230 - recurrent neural networks cheatsheet," *CS 230*, 2021.

[2] Hochreiter, "Understanding lstm networks – colah's blog," 2021.

[3] T. Zebin, N. Peek, A. Casson, and M. Sperrin, "Human activity recognition from inertial sensor time-series using batch normalized deep lstm recurrent networks," *Conference proceedings: ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*, vol. 2018, 07 2018.

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[5] H. He, "The state of machine learning frameworks in 2019," *The Gradient*, 2019.

[6] C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*. 01 2018.

[7] J. Chen, "Stock market definition," 2021.

[8] A. Hayes, "Technical analysis definition," 2021.

[9] M. Nabipour, P. Nayyeri, H. Jabani, and A. Mosavi, "Deep learning for stock market prediction," 03 2020.

[10] S. Nosratabadi, A. Mosavi, P. Duan, and P. Ghamisi, "Data science in economics," 03 2020.

[11] F. B. Oriani and G. P. Coelho, "Evaluating the impact of technical indicators on stock forecasting," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, 2016.

[12] R. Dash and P. K. Dash, "A hybrid stock trading framework integrating technical analysis with machine learning techniques," *The Journal of Finance and Data Science*, vol. 2, no. 1, pp. 42–57, 2016.

[13] F. Florencio, T. Silva, E. Ordonez, and M. Júnior, "Performance analysis of deep learning libraries: Tensorflow and pytorch," *Journal of Computer Science*, vol. 15, 05 2019.