Bachelor's Thesis
Final Report

Real-Time Stock predictions with Deep Learning and News Scraping

David Álvarez de la Torre
supervised by Dr. José Adrián Rodríguez Fonollosa

Polytechnic University of Catalonia
Submitted to the Faculty of Telecommunications Engineering of Barcelona in partial fulfillment of the requirements for the degree in
Telecommunications Technologies and Services Engineering

June 2018

"Your time is limited, so don't waste it living someone else's life. Don't be trapped by dogma — which is living with the results of other people's thinking. Don't let the noise of others' opinions drown out your own inner voice. And most important, have the courage to follow your heart and intuition."

– Steve Jobs, on Stanford's Commencement Ceremony of 2015

Abstract

Predict the stock market has always been one of the most challenging problems of the world. It is known that the market is influenced by uncountable things, however intuition says that one of the most correlated information is in fact public and accessible: the news.

The goal of this thesis is to use neural networks to check if it is possible to predict the stock market using only news previous to the opening of the session. To deal with the problem first we acquire the data, pre-process it and then we test different models to check if we can improve a random predictor.

Finally, the different models are compared and conclusions related with the correlation between news and stock market variations are exposed.

Resumen

Predecir el mercado siempre ha sido uno de los problemas más desafiantes del mundo. Se sabe que el mercado está influenciado por un gran número de factores, no obstante la intuición nos dice que uno de los tipos de información más correlada es pública y accesible: las noticias.

El objetivo de esta tesis consiste en utilizar redes neuronales para ver si es posible predecir el mercado bursátil utilizando únicamente noticias anteriores a la apertura de la sesión. Para afrontar este problema en primer lugar adquirimos la información, la pre procesamos y utilizamos diferentes arquitecturas para comprobar si podemos mejorar un predictor aleatorio.

Finalmente, se comparan las diferentes arquitecturas y se exponen las conclusiones a las que se ha llegado con respecto a la relación entre noticias y variaciones de bolsa.

Resum

Predir el mercat sempre ha estat un dels problemes més desafiadors del món. Se sap que el mercat està influenciat per un gran nombre de factors, però la intuïció ens diu que un dels tipus d'informació més correlada és pública i accessible: les notícies.

L'objectiu d'aquesta tesi consisteix a utilitzar xarxes neuronals per veure si és possible predir el mercat borsari utilitzant únicament notícies anteriors a l'obertura de la sessió. Per afrontar aquest problema en primer lloc adquirim la informació, la pre processem i utilitzem diferents arquitectures per comprovar si podem millorar un predictor aleatori.

Finalment, es comparen les diferents arquitectures i s'exposen les conclusions a les que s'ha arribat pel que fa a la relació entre notícies i les variacions de borsa.

Acknowledgements

I would like to dedicate this section not only to the people related with this thesis, but also to those who have helped me get where I am now.

The firsts of the list are, of course, my family. Without their facilities I could not have focus all my efforts on my studies to learn all the things I have learned during this past four years. Specially to my mum, who has supported me even when I was unbearable during exams period.

This bachelor's degree would not have been the same with good partners in it. I have had the luck to meet so many amazing people who I will remember for the rest of my life. I want to give a special mention to my comrade Oriol Barbany, the person who I have spent more time with.

The heart of a university are professors, and the Polytechnic University of Catalonia has a big and strong one. I do not want to be unfair with anyone so I would not mention any professor in particular, so I prefer to send a global gratitude to all of them.

To finish this section I want to give a special thank you to my supervisor, Dr. José Adrián Rodríguez Fonollosa, who has helped me during all the process of this thesis without complaints. If some day I manage to make profit with the stock market, you will be the first to know.

Revision History and Approval Record

Revision	Date	Purpose
0	27/05/2018	Document Creation
1	28/05/2018	Completion of the Introduction
2	01/06/2018	Completion of Appendices and Acknowledgments
3	04/06/2018	Completion of Budget
4	14/06/2018	Completion of State of the Art
5	19/06/2018	Completion of Methodology
6	25/06/2018	Completion of Results
7	28/06/2018	Completion of Conclusions and Abstract
8	29/06/2018	Revision and Correction

Document Distribution List

Name	Email
David Álvarez de la Torre	david.alvarez.de.la.torre@alu-etsetb.upc.edu
José Adrián Rodríguez Fonollosa	jose.fonollosa@upc.edu

Written by:		**Reviewed and Approved by**:	
Name:	David Álvarez de la Torre	**Name**:	José Adrián Rodríguez Fonollosa
Date:	28/06/2018	**Date**:	29/06/2018
Position:	Project Author	**Position**:	Project Supervisor

Contents

List of Figures **5**

List of Tables **6**

1 Introduction **7**

1.1 Statement of Purpose . . . 7

1.2 Requirements and Specifications . . . 8

1.3 Methods and Procedures . . . 8

1.4 Work Plan . . . 9

1.5 Incidences . . . 10

2 State of the Art of Deep Learning applied to the Stock Market **11**

2.1 Structured Events and Event Embeddings . . . 11

2.2 Sentence Embeddings with a RCNN . . . 13

2.3 Hierarchical Attention Networks . . . 14

3 Methodology **17**

3.1 Data Acquisition . . . 17

3.1.1 Database and ORM . . . 17

3.1.2 Scrapers . . . 18

3.2 Data Modeling . . . 18

3.2.1 Data Mining . . . 18

3.2.2 Experiments . . . 20

3.2.3 Loss Function and Optimizer . . . 21

3.2.4 Categorization and Error Measurements . . . 21

3.2.5 Profitability Test . . . 22

4 Results **23**

4.1 Convolutional Architecture . . . 24

4.2 Recurrent Architecture 26
4.3 Attention Architecture 27

5 Budget **29**

6 Conclusions **30**

Bibliography **32**

Appendices **33**

A Database Design **34**

B Recurrent Units **36**

B.1 GRU (Gated Recurrent Unit) 37
B.2 LSTM (Long Short-Term Memory) 37

Glossary **38**

List of Figures

1.1 Iterative process of Deep Learning 8

1.2 Gantt Diagram of the Project 10

2.1 Model to train Event Embeddings 12

2.2 Neural Network that relates Event Embeddings with Stock Variations 13

2.3 Sentence Embeddings with a RCNN model 14

2.4 HAN model 15

3.1 Convolutional Architecture (Conv1D + Max-Pooling) blocks diagram 20

3.2 Recurrent Architecture (LSTM) blocks diagram 20

3.3 Attention Architecture (Simplified HAN) blocks diagram 21

4.1 Accuracies of the Convolutional Architecture using the 5n_DS dataset 24

4.2 Accuracies of the Convolutional Architecture using the 50n_DS dataset 24

4.3 Losses of the Convolutional Architecture for the `GOOGL` stock 25

4.4 Benefits of the Convolutional Architecture using the 5n_DS dataset 25

4.5 Results of the Recurrent Architecture using the 5n_DS dataset 26

4.6 Accuracies of the Recurrent Architecture using the 50n_DS dataset 26

4.7 Accuracies of the Attention Architecture using the 5n_DS dataset 27

4.8 Benefits of the Attention Architecture using the 5n_DS dataset 28

A.1 Design of the Database 34

B.1 Structure of a RNN 36

List of Tables

1.1 Work Package 1 summary 9

1.2 Work Package 2 summary 9

3.1 Examples of sanitized new titles 19

4.1 Datasets used in the thesis 23

5.1 Budget of the Project 29

Chapter 1

Introduction

Exchange of goods has been present in humanity since the beginning of our race. Back in prehistoric times, people coordinate with their equals so they could survive by exchanging food or whatever material they could provide.

With the past of time these transactions became more and more sophisticated until we had to create the money as a symbol of debt, that is to say, money was the prove that you gave something to someone and now society owned you something in return.

Modern stock markets are the center of goods exchanges in the current times. Every day millions of people trade equities, which are portions of a company. The prices at which people buy and sell their equities define the current price of them, and so the value of a company.

The goal of the people that buy equities is to sell them for a higher price some day. Several different techniques have been used to accomplish that goal, all of them can be divided in two different approaches:

- **Fundamental Analysis**: concretes on financial and economic reports to determine the value of a company and detect undervalued equities.
- **Technical Analysis**: studies past market actions to predict future movements. Best known concepts of this type of analysis are support and resistance levels, which indicates bearish and bullish trends.

It has also been proved that the stock market is "informationally efficient" [3] - stock prices reflect all known information. In this thesis our goal is to include a new type of information that is not directly used in classical analysis methods: the news.

1.1 Statement of Purpose

The thesis has one primary goal: to predict the stock market using news as an only source of information. However, it has already been proved that titles are more efficient than the content to predict the stock market [10], so we will focus on using only news titles.

The following list summarizes all the goals of the thesis:

- Design a relational database with the analyzed markets, stocks, prices, news and related information.
- Code a script capable of scraping the value of a stock every $\Delta t = 1$ day.
- Code a script capable of scraping the news published in an (almost) real-time way.
- Code a script capable of acquire social information of the news to determine which ones are more important.
- Test and design different neural network architectures and compare the accuracy obtained with each one.
- Design a set of stock market trading rules and compute the daily benefit for each architecture.

1.2 Requirements and Specifications

Although the goal of this thesis is to predict the stock market, the objective is not to make money but to create a first prototype capable of improving the random accuracy. Even though, there are a few requirements that should be accomplished if we would like to use our software for that purpose.

All of these requirements are related with processing speed, because in the stock market every second counts. For this project **we consider those production requirements without the speed limitations**:

- Recent news should be scraped by the system ~~with a delay smaller than a minute~~.
- The model input has to follow a given format and the execution time should be small.
- All the trading rules established should be possible to follow in a production environment.

In terms of accuracy, our goal is to overcome the random accuracy to obtain a advantage against the market.

1.3 Methods and Procedures

This thesis does not follow the work of any previous project, so the methods and procedures are based on intuition and good practices. As every Deep Learning project, the workflow is the following:

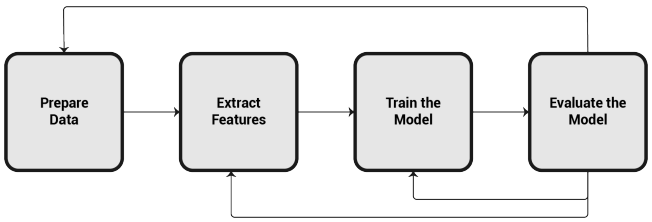


Figure 1.1: Iterative process of Deep Learning

We can not avoid any of the blocks, as it is a sequential and required process. In details, the general description of each block and the application to this project is the following:

- **Prepare Data**: given a set of data in an arbitrary format, in this block we must transform this data (or part of it) so it can be processed by our neural network later. In our project our data is stored in a MySQL database, which is not accessible by the neural network. We export this data to a `.json` file including only relevant title news, reducing the file size from a bunch of gigabytes to only a few megabytes.
- **Extract Features**: in most cases raw data is not used as an input of a neural network, but only a representation of it. In this case, we transform each word to a word embedding.
- **Train the Model**: once the data has been prepared, we must run the model so it propagates and back-propagates through the layers using the batch of samples.
- **Evaluate the Model**: every time a mini-batch has gone through the neural network, a set of quality measures is computed.

The iterative process is simple: we extract a subset of the database data, transform it and train a model. If the model does not converge, in the next iteration we must change the subset selection criteria and/or change the architecture of the model. Our goal is to complete this process as fast as possible so we can perform more experiments and archive better results.

1.4 Work Plan

The thesis has been divided in two big work packages:

WP1: Data Acquisition	
Time Period	26/02/2018 - 15/04/2018 (7 weeks)
Description	In this work package the scripts to retrieve stock prices and scrap news will be coded.
Tasks	• Design the relational database. • Use an ORM to map the database with the scripts. • Program a script that retrieves stock values every day. Take into account exceptional scenarios like market closing or server breakdowns. • Program one script to scrap news from Reuters. • Program a script that scraps social signals from news. • Schedule a cron job that executes both scripts every few seconds using parallel computing.
Deliverables	Source code of each script hosted at GitHub.

Table 1.1: Work Package 1 summary

WP2: Data Modeling	
Time Period	16/04/2018 - 24/06/2018 (10 weeks)
Description	Use neural networks to model the acquired data.
Tasks	• Transform the acquired data so it can be the input of a neural network as a classification problem. • Design an evaluation test for the modelx. • Try different neural network architectures and evaluate the obtained accuracy. • Design a set of trading rules and estimate the benefits of each model.
Deliverables	Source code of the model hosted at GitHub.

Table 1.2: Work Package 2 summary

At the beginning of the project we planned to include another work package related with the visualization of the data. However, as the WP1 difficulty was much higher than the expected and there exist services which do precisely that, we decided to suppress it. Basically, the idea was to represent all the price data points in a plot and associate them with the news that are used for they predictions.

To organize all the tasks of the thesis a Gantt diagram was created the first week of the course. This diagram has changed every since, with its last version showed here:

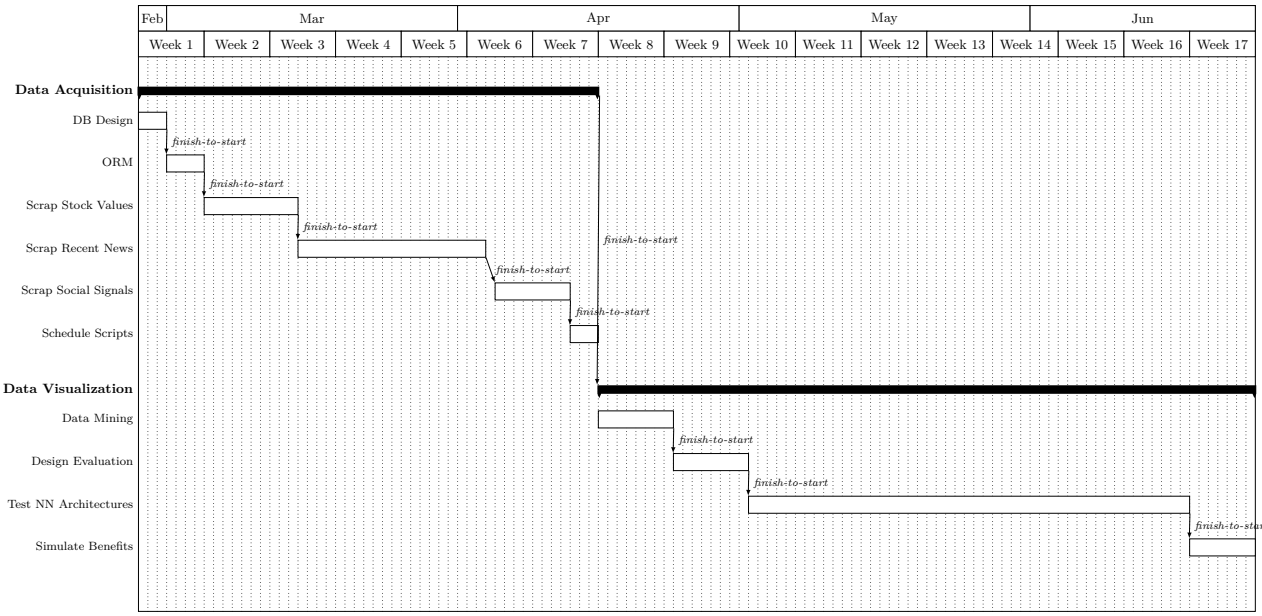


Figure 1.2: Gantt Diagram of the Project

This diagram is just an approximation of the time dedicated to every task and it does not reflect small setbacks or bug patches.

1.5 Incidences

During the thesis there have been several small and big incidences. In this chapter we expose the three biggest difficulties that we have encountered in the project and the solutions given to them:

- In an application like the stock market it is mandatory to verify the integrity of the data. For this reason it is essential to have a perfectly designed database, so if a program has an error it can be easily identified. When coding the stock prices module we realized that there are days when certain market does not open; or even worse, days when a stock does not open. All this casuistry made the module much more difficult to code.

  To solve this problem we decided to create auxiliary tables to store these special dates so when can store the exact information we want.

- The initial idea of the project was to scrap news from different newspapers. However, we quickly realized that modern websites include several security mechanism to avoid precisely that. For example, by limiting the requests that a user can make every minute.

  However, Reuters.com, one of the best sources of market news does not include any of these mechanisms. We decided to focus on this newspaper as it produces enough news every day. Besides, using only one newspaper removes the content clustering that we have should used to avoid repeated content.

- When modeling the news titles we realized that a big portion of them were not directly related with the stock market. It is known that neural networks should be able to learn by themselves which data is relevant and which one is not, however this idea is related with information in the big data scale.

  As we do not have that much information, we had to somehow filter this uncorrelated news before feeding the neural networks. The first idea was to add social media signs to each new, so we could use most shared news. This approach did not improve the results as we expected, so finally we decided to categorize each entry and use only news directly related with the stock market.

Chapter 2

State of the Art of Deep Learning applied to the Stock Market

Predict the stock market has always been seen as a difficult field of research. Back in the seventies, there was the theory that the market followed a random walk and it was impossible to predict [7], however with the arrival of Artificial Intelligence (AI) it has been proved that the stock market is not a random process [2, 9, 12].

Most of the research in the stock market field has been focused on predicting future values using previous ones, or as we said in the introduction, creating some type of technical analysis predictor. For example, Althelaya and Mohammed [1] used a Long Short-Term Memory (LSTM) based neural network to make short and long term predictions.

However, the focus of this thesis is to predict the stock market using news as the only source of data, which implies a Natural Language Processing (NLP) task. Most of the old research is based on linear classifiers using bag-of-words or even known sentences, however in the last few years a couple of novel approaches have overcome these old methodologies using state of the art neural network architectures.

2.1 Structured Events and Event Embeddings

The idea of extract events from sentences was introduced by Kim [6] and used by Ding et al. in his two papers [2, 9]. The procedure is simple: given a sentence the structured representation is composed by an actor, an action and an object. The resulting tuple is written as (O_1, P, O_2). For example:

$$\text{Apple sues Microsoft} \longrightarrow (O_1 = \text{'Apple'}, P = \text{'sues'}, O_2 = \text{'Microsoft'})$$

The justification of these papers about this approach is that unstructured text is too sparse to be used in this type of predictive models. Once done this transformation, the first paper [9] compares the performance of using a Support Vector Machine (SVM) against a basic LSTM architecture using both bag-of-words and structured events. The result is that the best performance is archived using the LSTM architecture in combination with structured events.

In the updated paper [2], they propose an even better approach to use structured events to predict the stock market. The problem is that this tuples are still too sparse, so they should find a way to represent similar actions with similar input vectors. They create event embeddings, which is the equivalent of word embeddings but applied to this structured events.

The procedure to transform a tuple of word embeddings into a single event embedding uses a standard fully-connected neural network design:

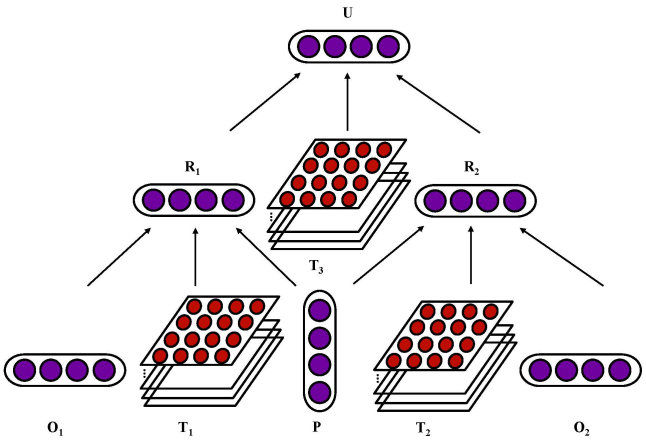


Figure 2.1: Model to train Event Embeddings

Where $\boldsymbol{T_1} \in \mathbb{R}^{d \times d \times k}$, $\boldsymbol{T_2} \in \mathbb{R}^{d \times d \times k}$ and $\boldsymbol{T_3} \in \mathbb{R}^{d \times d \times k}$ are the trainable weights and $\boldsymbol{o_1} \in \mathbb{R}^d$, $\boldsymbol{p} \in \mathbb{R}^d$ and $\boldsymbol{o_2} \in \mathbb{R}^d$ are initialized as the average of the word embeddings of the actor, action and object, respectively.

To compute $\boldsymbol{r_1} \in \mathbb{R}^d$, $\boldsymbol{r_2} \in \mathbb{R}^d$ and $\boldsymbol{u} \in \mathbb{R}^d$ they include another two parameters, $\boldsymbol{W} \in \mathbb{R}^{k \times 2d}$ and $\boldsymbol{b} \in \mathbb{R}^k$:

$$\boldsymbol{r_1} = f(\boldsymbol{o_1}^T \boldsymbol{T_1}^{[:,:,i]} \boldsymbol{p} + \boldsymbol{W} \begin{bmatrix} \boldsymbol{o_1} \\ \boldsymbol{p} \end{bmatrix} + \boldsymbol{b}) \qquad i = 1, ..., k$$

$$\boldsymbol{r_2} = f(\boldsymbol{o_2}^T \boldsymbol{T_2}^{[:,:,i]} \boldsymbol{p} + \boldsymbol{W} \begin{bmatrix} \boldsymbol{o_2} \\ \boldsymbol{p} \end{bmatrix} + \boldsymbol{b}) \qquad i = 1, ..., k$$

$$\boldsymbol{u} = f(\boldsymbol{o_1}^T \boldsymbol{T_3}^{[:,:,i]} \boldsymbol{o_2} + \boldsymbol{W} \begin{bmatrix} \boldsymbol{o_1} \\ \boldsymbol{o_2} \end{bmatrix} + \boldsymbol{b}) \qquad i = 1, ..., k$$

To train the parameters $\boldsymbol{\Phi} = (\boldsymbol{T_1}, \boldsymbol{T_2}, \boldsymbol{T_3}, \boldsymbol{W}, \boldsymbol{b})$ we create a set of corrupted events ξ^r by replacing one of the three elements of the tuple by a random element of another tuple. Then, the algorithm works as follows:

Data: $\xi = (E_1, E_2, ..., E_n)$, $\xi^r = (E_1^r, E_2^r, ..., E_n^r)$ and the initialized *EELM* model
Result: Updated *EELM* model
while $\xi \neq []$ **do**
  loss $\longleftarrow \max\left(0, 1 - f(E_i) + f(E_i^r) + \lambda \left\|\boldsymbol{\Phi}\right\|_2^2\right)$;
  if $loss > 0$ **then**
    Update($\boldsymbol{\Phi}$);
  else
    $\xi \longleftarrow \xi \setminus \{E_i\}$;
  end
end
return *EELM*

Algorithm 1: Events Embedding algorithm

Where $f = tanh$, the regularization parameter λ is set to 0.0001 and the update is done using the standard back propagation algorithm.

Once we have the event tuples transformed into event embeddings, the neural network that relates these with stock variations is the one showed in this figure:

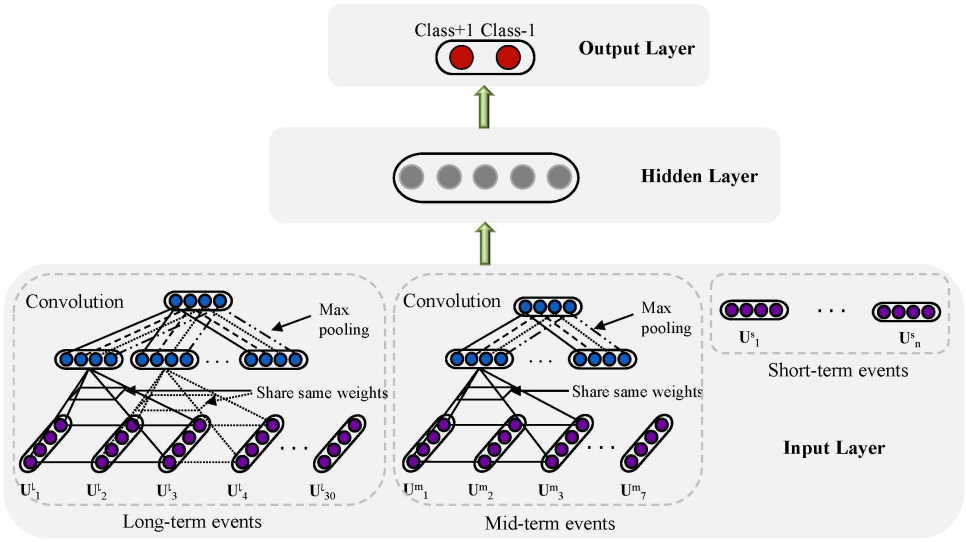


Figure 2.2: Neural Network that relates Event Embeddings with Stock Variations

As we can see the model is divided in three blocks: one for long-term events, one for mid-term events and one for short-term events. We denote $\boldsymbol{u_n}$ as the average of the event embeddings of day n. Then, for the long-term block we select the past 30 days vectors; for the mid-term block, the previous 7 days; and for the short-term, just the day before the prediction.

To combine the event embeddings of the long and mid-term blocks a convolution with $l = 3$ is applied along with a max-pool layer. Formally, given the matrix of events $\boldsymbol{U_{lt}} \in \mathbb{R}^{d \times 30}$ and a vector of weights $\boldsymbol{w} \in \mathbb{R}^l$, we obtain $\boldsymbol{Q_{lt}} \in \mathbb{R}^{d \times 30-l+1}$ and $\boldsymbol{V_{lt}} \in \mathbb{R}^d$ as:

$$\boldsymbol{Q_{lt}}^{[:,j]} = \boldsymbol{w}^T \cdot \boldsymbol{U_{lt}}^{[:,j-2:j]} \qquad j = 3, ..., 30$$

$$\boldsymbol{V_{lt}}^{[k]} = \max\left(\boldsymbol{Q_{lt}}^{[k,:]}\right) \qquad k = 1, ..., d$$

Note: The symbol $\cdot$ represents a dot product between the vector and all the rows of the matrix.

The same process is applied to the mid-term block, obtaining $\boldsymbol{Q_{mt}} \in \mathbb{R}^{d \times 7-l+1}$ and $\boldsymbol{V_{mt}} \in \mathbb{R}^d$. In the case of the short-term block, $\boldsymbol{V_{st}} \in \mathbb{R}^d$ is the average of the event embeddings directly.

Finally, we combine the three vectors to create $\boldsymbol{V} \in \mathbb{R}^{3d}$ and we use a fully-connected layer in combination with a softmax layer to output the correct class.

2.2 Sentence Embeddings with a RCNN

It is known that the importance of news in the stock market has a small temporal effect, having his real impact in the closest market session. Following this idea, Vargas et al. [12] propose a model where only data from the previous day is used. Specifically, the data used are the news and seven stock indicators, which are not detailed in the paper.

The authors propose a method combining a Convolutional Neural Network (CNN) and a Recurrent Neural Network (RNN), arguing that CNNs are better extracting semantic information while RNNs are better catching context information and complicated temporal references. The final model is represented in the following figure:

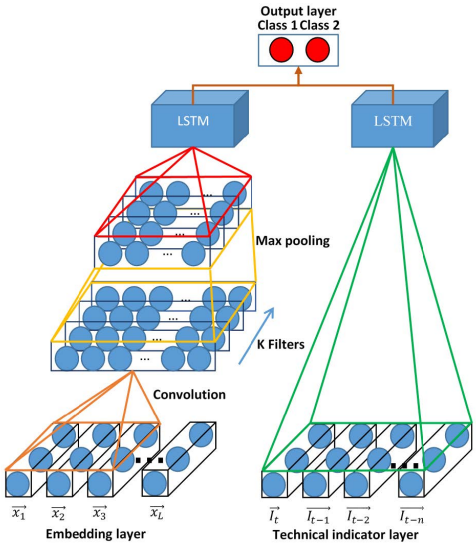


Figure 2.3: Sentence Embeddings with a RCNN model

As we can see, the model is divided in two blocks: one regarding to the news titles and the other with the technical indicators. As in this thesis we focus on news, the second block will not be explained. However, the structure is identical to the first block.

The first step is to prepare the sentence embeddings, which are the average of the words of the titles. For each day, we have a set of L new titles denoted as $\boldsymbol{x}_l \in \mathbb{R}^d$. We can create the matrix of titles $\boldsymbol{X} \in \mathbb{R}^{d \times L}$.

Once the data is prepared, we apply a single filter of window size R with a weights matrix $\boldsymbol{W}_k \in \mathbb{R}^{d \times R}$ and bias $\boldsymbol{b}_k \in \mathbb{R}^R$. Then, we compute $\boldsymbol{Q}_k \in \mathbb{R}^{R \times L-R+1}$ as:

$$\boldsymbol{Q}_k^{[:,j]} = (\boldsymbol{W}_k \cdot \boldsymbol{X}^{[:,j:j+R-1]})^T + \boldsymbol{b}_k \qquad j = 1, ..., L-R+1$$

Note: In this case, symbol $\cdot$ represents the dot product between the matrices and the sum of the elements of each column. This is not a traditional convolution, but the one explained in the paper.

The next step of the chain is to apply a max pooling layer so we obtain $\boldsymbol{V} \in \mathbb{R}^{L-R+1}$:

$$\boldsymbol{V}_k^{[i]} = \max\left(\boldsymbol{Q}^{[:,i]}\right) \qquad i = 1, ..., L-R+1$$

Finally, we apply the ReLU activation function to $\boldsymbol{V}$ and a Dropout layer [11] and pass the result though a LSTM layer to obtain the final result.

Notice that the procedure explained before is applied to each one of the K filters, so we can create a matrix $\boldsymbol{H} \in \mathbb{R}^{L-R+1 \times K}$ by concatenating the vectors $\boldsymbol{V}_k$.

2.3 Hierarchical Attention Networks

One of the state of the art architectures of Deep Learning are attention networks. These type of networks use attention vectors to detect important fragments of the input according to the context.

In the paper published by Yang et al. [13], a two levels attention network to classify documents called Hiererchical Attention Network (HAN) is proposed. The first level receives the word embeddings of a sentence as input and returns a representative vector; the second level receives all these representative vectors and returns a document vector.

The model is divided in four layers, however the first two are identical to the last two with different input lengths. Given the matrix of word embeddings $\boldsymbol{X_i} \in \mathbb{R}^{T \times d}$ of the sentence i, we apply a bidirectional Gated

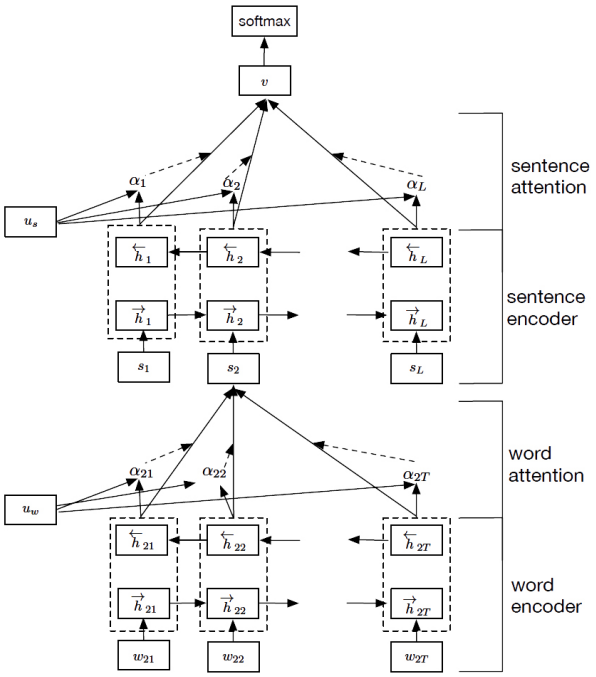


Figure 2.4: HAN model

Recurrent Unit (GRU) to obtain the hidden state of size z of the word t ($\boldsymbol{h_{it}} \in \mathbb{R}^{2z}$) as:

$$\overrightarrow{\boldsymbol{h_{it}}} = \text{GRU}(\overrightarrow{x_{it}})$$
$$\overleftarrow{\boldsymbol{h_{it}}} = \text{GRU}(\overleftarrow{x_{it}})$$
$$\boldsymbol{h_{it}} = [\overrightarrow{\boldsymbol{h_{it}}}, \overleftarrow{\boldsymbol{h_{it}}}]$$

Once the words are encoded, the next step is to apply the attention layer to detect words that are important to the sentence. Firstly, we obtain the hidden representation $\boldsymbol{u_{it}} \in \mathbb{R}^{2z}$ using a weights matrix $\boldsymbol{W_w} \in \mathbb{R}^{2z \times 2z}$ and a bias $\boldsymbol{b_w} \in \mathbb{R}^{2z}$:

$$u_{it} = \tanh\left(\boldsymbol{W_w h_{it}} + \boldsymbol{b_w}\right)$$

Then, we measure the importance of the word as the similarity of $\boldsymbol{u_{it}}$ with a word level context vector $\boldsymbol{u_w} \in \mathbb{R}^{2z}$ to get the normalized weight $\alpha_{it} \in \mathbb{R}$:

$$\alpha_{it} = \frac{\exp\left(\boldsymbol{u_{it}}^T \boldsymbol{u_w}\right)}{\sum_t \exp\left(\boldsymbol{u_{it}}^T \boldsymbol{u_w}\right)}$$

Finally, we obtain the sentence vector $\boldsymbol{s_i} \in \mathbb{R}^{2z}$ as the weighted sum of the hidden states:

$$\boldsymbol{s_i} = \sum_t \alpha_{it} \boldsymbol{h_{it}}$$

The equations applied to the second levels are the same, but this time the inputs are $\boldsymbol{S} \in \mathbb{R}^{L \times 2z}$. We would first compute the hidden states of size s ($\boldsymbol{h_i} \in \mathbb{R}^{2s}$) as:

$$\overrightarrow{\boldsymbol{h_i}} = \text{GRU}(\overrightarrow{s_i})$$
$$\overleftarrow{\boldsymbol{h_i}} = \text{GRU}(\overleftarrow{s_i})$$
$$\boldsymbol{h_i} = [\overrightarrow{\boldsymbol{h_i}}, \overleftarrow{\boldsymbol{h_i}}]$$

Then, we would compute the hidden representation $\boldsymbol{u_i} \in \mathbb{R}^{2s}$ using a different $\boldsymbol{W}_s \in \mathbb{R}^{2s \times 2s}$ and $\boldsymbol{b}_s \in \mathbb{R}^{2s}$ parameters:

$$u_i = \tanh\left(\boldsymbol{W_s h_i} + \boldsymbol{b_s}\right)$$

The same idea is applied to obtain the weighting factors $\alpha_i \in \mathbb{R}$ using a context sentence vector $\boldsymbol{u_s} \in \mathbb{R}^{2s}$:

$$\alpha_i = \frac{\exp\left(\boldsymbol{u_i}^T \boldsymbol{u_s}\right)}{\sum_i \exp\left(\boldsymbol{u_i}^T \boldsymbol{u_s}\right)}$$

Finally, we obtain the vector representing a document $\boldsymbol{v} \in \mathbb{R}^{2s}$ as the weighted sum of all the sentence hidden states:

$$\boldsymbol{v} = \sum_i \alpha_i \boldsymbol{h_i}$$

The final step to relate this document vector with classes is to apply a fully-connected layer along with a softmax layer, as showed in 2.4.

Chapter 3

Methodology

The work of this thesis has been divided in two big parts: data acquisition and data modeling. In this chapter we will explain the technical details of the implementation of both packages as well as the technologies implied in each one. To begin, let's give a brief of them:

- **Data Acquisition**: in this package our goal is to obtain the data used to train the model. We could have used internet databases, however they were too old and we wanted to be able to obtain recent data. The types of data acquired are explained later.
- **Data Modeling**: once we have the data, the goal is to relate it with stock variations. To do that, a set of experiments will be performed and a benefit test will be executed on the resulting models.

3.1 Data Acquisition

The data acquisition packages covers all the process of finding useful data, retrieve it and store it so it can be easily accessed in the future. For this package and all the thesis, the programming language used is `Python 3.6`, which is a flexible and easy to learn language with beautiful syntax and thousands of libraries.

3.1.1 Database and ORM

In order to store big amounts of data the best option is always to use a database. In this thesis, the selected database is `MySQL 5.7`, running in a `CentOS 6.5` Virtual Private Server (VPS) of the company OVH.es. The choice of a relational database instead of a NoSQL database is because of the unmanaged uniqueness that the first one guarantees to certain data, and so an easier way to code the inserts and updates.

The tables of the database should store two different types of data: stock prices and news. However, creating one table for each type of data was not enough, as if we wanted the program to be robust different scenarios should be taken into account:

- What happens if we want to scrap a price of a stock in a holiday?
- At what time does the program has to scrap the stock price data? When does a certain market close? Are there any days that this closing time is not the usual one?
- How can we avoid trying to scrap prices of a stock that has just gone public?
- How can we handle similar news from different newspapers?

To give a programmatic solution to all this scenarios we must store auxiliary information related with stock markets, newspapers and stocks. Altogether, seven different relational tables have been created. The complete design and explanation of each table has been included in appendix A.

Finally, to avoid raw SQL queries we have added a Object Relational Model (ORM) between the database and the Python code. An ORM relates entries of a database with objects so data manipulation can be done directly with object methods. In this thesis, the library used has been `SQLAlchemy`, which is the preferred option for Python engineers. The explanation of the operation of an ORM is outside of the scope of this text.

3.1.2 Scrapers

To extract data from the internet and store it in the database we create pieces of software called scrapers, which are algorithms capable of extracting data from websites. In this thesis we call scrapers to those scripts that connect with Application Programming Interfaces (APIs), which are not scrapers in the original sense of the word.

In total, we have divided the scraping functionalities in five different scripts:

- **scrap_prices_archive**: retrieves all the prices from all the stock from `DAY_ONE` to the previous market session that are not in the database.
- **scrap_prices_recent**: retrieves the prices of the current market session for all the stocks in the database.
- **scrap_news_archive**: retrieves all the news from all the newspapers from `DAY_ONE` to the previous day that are not in the database.
- **scrap_news_recent**: retrieves the news of the current day for all the newspapers in the database.
- **scrap_social**: updates the Facebook reactions, comments and shares of all the news in the database.

Although the programming details are not covered in this document, a few key aspects of the implementation should be mentioned:

- As seen in the appendix A, for each date and stock an opening, closing, minimum and maximum prices are stored in the database along with the volume and the adjusted closing price. However, in this thesis only the closing price is used to compute daily variations. The same happens with the news, as only the title of news are used in the data modeling.
- The AlphaVentage API and SharedCount API are used as data providers for the stock prices and social signals, respectively. However, the news are scraped directly using the `urrlib` library of Python.
- All the implementations use the `ThreadPoolExecutor` class of the `concurrent.futures` library to increase the processing speed, as it would take months to process that big amount of requests otherwise.

Unfortunately, the implementation is not publicly available as it is intended to be used someday in the future.

3.2 Data Modeling

Model the acquired data and make good predictions is the heart of this thesis. In this section we will explain and justify all the process and the experiments that have been performed in this thesis.

3.2.1 Data Mining

One of the principles of deep learning is that the bigger the amount of data the better. This affirmation is true, however it comes with a required condition: the data must be relevant. Imagine that we want to predict if there is a cat in an image and we have a dataset with a million different images, but only a ten percent of it has the correct labels while the remaining are just incorrectly labeled images. The result will be a neural network with very low accuracy.

The same idea is applied when we try to relate news with stock markets. We have a big amount of news, but only a small portion of them have effect in the price of a certain stock. If we use this big amount of uncorrelated

data and the true reason of a big variation of stock is not represented in the input, the neural network will not be able to learn anything.

For this reason we must make it as easier as possible to the network to learn. The procedure is simple: we must remove as much uncorrelated data as possible without removing the good one. In this thesis, we create the datasets using two different types of news:

1. First of all, we select only those news with the predicted stocks in the title.
2. After that, we fill the gaps with the most shared news of each day until we get M [1] news per date. To select only relevant news, we choose news whose categories are "Market News", "Business News" or "Financials".

In both cases, we must clean these data to generalize the inputs without losing relevant information. The sanitation process is divided in 9 steps:

1. Remove introductory text of a title.
2. Remove title tails.
3. Transform the text to lowercase.
4. Remove points of abbreviations.
5. Remove numerical words. This is done because the majority of titles with numerical values are related with financial results, which should have its own predictor.
6. Remove commas.
7. Replace special characters (for example, a hyphen) with spaces.
8. Remove double spaces and spaces at the beginning or at the end of the title. This situation may appear as a consequence of the previous steps.
9. Lemmatization of all the tokens of the title.

In the following table we can see a few examples of the data mining applied to the titles:

Original Title	**Sanitized Text**
BRIEF-Whitebox Advisors reports 7 pct passive stake in Basic Energy Services	whitebox advisor report pct passive stake in basic energy service
BRIEF-Samsung, Google introduce two Chromebooks designed for Google Play	samsung google introduce two chromebook design for google play
BRIEF-Italy car sales rise 13 pct in December - ministry	italy car sale rise pct in december
BRIEF-U.S. Department of Labor sues Google for compensation data	we department of labor sue google for compensation datum

Table 3.1: Examples of sanitized new titles

As we can see, the sanitized titles maintain the information of the original ones but with a simpler form. However, the use of a lemmatizer can produce errors, like in the word "us", which is understood as the pronoun instead of as the abbreviation of the United States.

[1] In this thesis we will experiment with $M = \{5, 25, 50\}$.

3.2.2 Experiments

As mentioned on the previous section, in this thesis we will experiment with datasets containing different number of news per day. Besides, we will focus on creating models for specific stocks, specifically we will try with two of the best known companies of the world:

- Alphabet Inc. Class A (**GOOGL**)
- Apple Inc. (**AAPL**)

We will evaluate the correlation between news and stock price changes using three different architectures. The goal of this thesis is not only to obtain an advantage against the market, but also to study the performance of three types of neural network architectures: recurrent networks, convolutional networks and attention-based networks. As will be explained in the results, we have decided to create shallow neural networks as we do not have enough data to train deep networks.

For all architectures the first layer will be the same: an embedding layer with the weights of the Word2Vec [4] implementation of Google that follows the paper of Mikolov et al. [8]. After that step, we compute the mean of the word embeddings of each new title to obtain a sentence embedding, as done in [2, 12].

As we design our architectures to classify between two classes, the last layers will be a linear layer that converts the hidden size to a two dimensional output followed by a softmax so the sum of both values is one.

Finally, we add a *Prediction Threshold* hyperparameter which establishes the minimum required probability of a prediction so it is taken into account.

Convolutional Architecture (Conv1D + Max-Pooling)

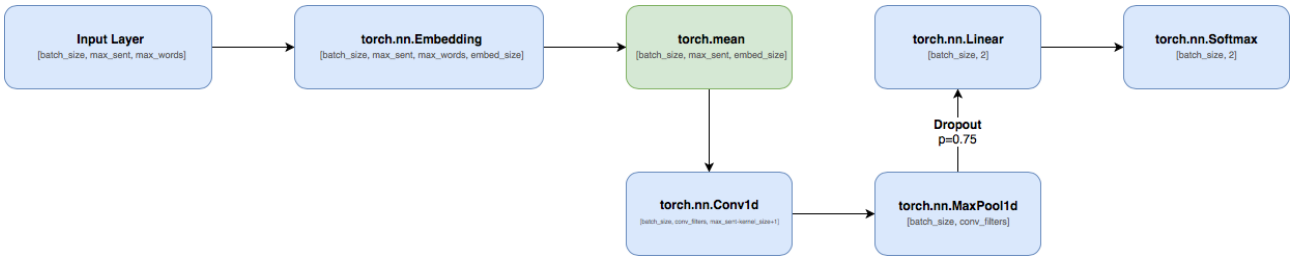


Figure 3.1: Convolutional Architecture (Conv1D + Max-Pooling) blocks diagram

Intuition: The combination of a convolution and a max pooling should act as an alarm for the linear layer, detecting important features in the news that should be related with the variations of stock prices. Besides, this is the fastest implementation of the three architectures as convolutions are light weight operations.

Hyperparameters: Batch Size = $\{32\}$, Prediction Threshold = $\{0.5, 0.65\}$, Convolution Filters = $\{5, 7, 9\}$, Kernel Size = $\{1\}$.

Recurrent Architecture (LSTM)

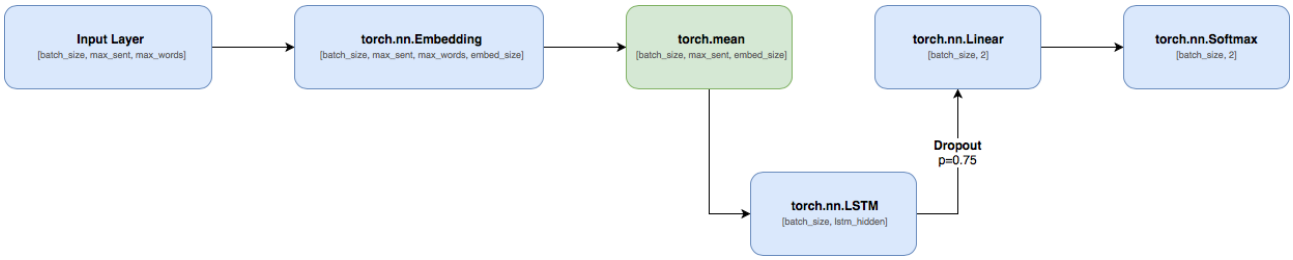


Figure 3.2: Recurrent Architecture (LSTM) blocks diagram

Intuition: The LSTM is the recurrent unit with the best performance nowadays. This unit should find relations in the news that relates with the stock variations directly.

Hyperparameters: Batch Size = $\{32\}$, Prediction Threshold = $\{0.5, 0.65\}$, LSTM Hidden Size = $\{3, 5, 7, 9\}$.

Attention Architecture (Simplified HAN)

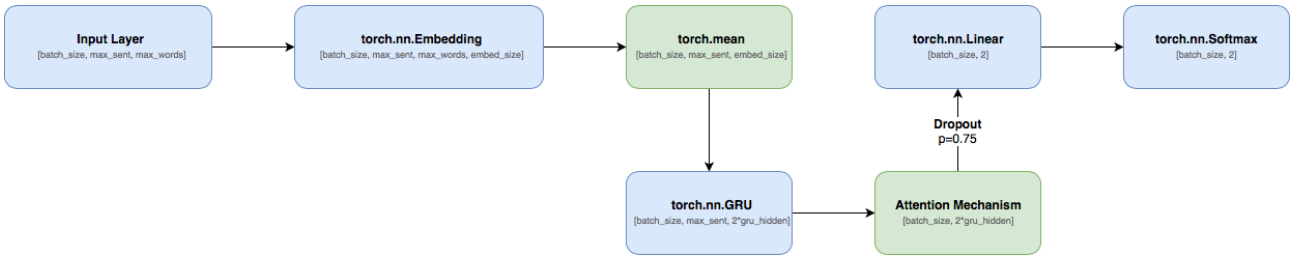


Figure 3.3: Attention Architecture (Simplified HAN) blocks diagram

Intuition: The GRU layer should find relations between the news while the attention mechanism should weigh the importance of them.

Hyperparameters: Batch Size = $\{32\}$, Prediction Threshold = $\{0.5, 0.65\}$, GRU Hidden Size = $\{7, 11\}$.

3.2.3 Loss Function and Optimizer

Two of the most important pieces of a deep learning model are the loss function and the optimizer. While the first measures the error of the predictions of the model, the second establishes how the back propagation step must be done.

When dealing with classification problems, one of the most used loss functions is the cross entropy loss, which in `Pytorch` is just the combination of applying a logarithm to the softmax module and a negative log likelihood loss. In combination, the equation that computes the loss of a sample for a two classes classification problem is:

$$\text{Loss}(\hat{\mathbf{y}} = [p_0, p_1], y \in \{0, 1\}) = y \log(p_1) + (1 - y) \log(p_0)$$

On the other hand, we use the Adam optimizer as it adapts the learning rates of each parameter independently, turning it into a very fast convergence algorithm. The formula applied to each parameter update is:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Where m_t is the first order moment and v_t is the second order moment of the gradients; η is the learning rate and $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

3.2.4 Categorization and Error Measurements

In this project we treat stock variations as a classification problem, that is to say, we do not try to estimate the exact rise or fall of the price of a certain stock but instead we try to classify it in one of two classes. We create one class representing a positive change and another class representing a negative change, however with this approach we would have an unbalanced training set. To solve that, for each dataset, we define a `class_weights` parameter which is passed to the loss function so both classes contribute equally to the computed loss.

Given a stock closing price of a certain day t P_t^c and a closing price of the previous day P_{t-1}^c, we compute the price variation as:

$$\Delta P_t = \frac{P_t^c - P_{t-1}^c}{P_{t-1}^c}$$

Then, we assign a class to each ΔP_t as:

$$y_t = \begin{cases} 0, & \text{for } \Delta P_t \leq 0 \\ 1, & \text{for } \Delta P_t > 0 \end{cases}$$

Then, given a set of outputs $\hat{y}_t$, we measure the accuracy as:

$$Acc(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\#\text{Correct Samples}}{\#\text{Total Samples}} \times 100$$

3.2.5 Profitability Test

To simulate a real-life scenario we will define a trading strategy using our models. As our model's output is a daily prediction, the strategy will focus on buying at the opening and selling at the closing.

For our experiment we will suppose that we have $Q = 10.000\$$ to buy or going short (depending on the prediction of the evaluated model) every day. As we predict only one stock per model, we will suppose that we use all the money to invest in that stock buying (or shorting) at the opening price and selling (or buying) at the closing price every day. [2]

Given a model output $\hat{\mathbf{y}}_\mathbf{t} = [p_0, p_1]$, where:

$$\begin{aligned} p_0 &= P(\Delta P_t \leq 0 \mid \text{news}_t) \\ p_1 &= P(\Delta P_t > 0 \mid \text{news}_t) \\ p_0 + p_1 &= 1 \end{aligned}$$

Then, we will buy Q_{buying}^t or we will short $Q_{shorting}^t$ for a certain day t following the equations:

$$\begin{aligned} Q_{shorting}^t = Q \quad \text{if} \quad p_0 \geq \text{prediction\_threshold} \quad \text{else} \quad Q_{shorting}^t = 0 \\ Q_{buying}^t = Q \quad \text{if} \quad p_1 \geq \text{prediction\_threshold} \quad \text{else} \quad Q_{buying}^t = 0 \end{aligned}$$

Notice that, for each day, we use all the money only if the probability is above a given threshold. This mechanism should avoid betting in a stock which could go either way with similar probabilities.

Then, we can compute the benefit of a single day t as:

$$B_t = \begin{cases} -\Delta P_t \cdot Q_{shorting}^t & \text{if} \quad p_0 \geq \text{prediction\_threshold} \\ \Delta P_t \cdot Q_{buying}^t & \text{if} \quad p_1 \geq \text{prediction\_threshold} \\ 0 & \text{otherwise} \end{cases}$$

Finally, we compute the average daily benefit, which will be used as a measure of profitability in the results chapter as [3]:

$$\bar{B}_{daily} = \text{mean}(B_t) \quad \forall B_t \neq 0$$

[2] In our experiment we suppose that we can buy portions of a share as it does not affect to the final profitability.

[3] The equation is in fact an abuse of notation. The real meaning is that we take the mean of the benefits of the days that we have invested to obtain an averaged measure.

Chapter 4

Results

As explained in the previous chapter, we will evaluate the performance of three different architectures using three different datasets:

- **5n_DS**: specific news of the predicted stocks plus categorical news until reaching 5 news per date.
- **25n_DS**: specific news of the predicted stocks plus categorical news until reaching 25 news per date.
- **50n_DS**: specific news of the predicted stocks plus categorical news until reaching 50 news per date.

Where, for each market session, we select the news affecting that session using a temporal window that goes from the previous session closing time until the predicted market session opening time. First of all, let's give a few insights about the data contained in these datasets:

Dataset		**5n_DS**	**25n_DS**	**50n_DS**
Train Samples	**Dates Range**	01/01/2010 - 31/12/2016		
	Number of Samples	1632		
	Avg. Daily Variations [%] [3]	[0.057, 0.030]		
	Specific News [%] [4]	[18.9, 30.89]	[5.07, 8.45]	[2.53, 4.25]
Val. Samples	**Dates Range**	01/01/2017 - 31/12/2017		
	Number of Samples	241		
	Avg. Daily Variations [%]	[0.123, 0.149]		
	Specific News [%]	[13.11, 25.64]	[3.20, 6.45]	[1.60, 3.22]
Test Samples	**Dates Range**	01/01/2018 - 31/05/2018		
	Number of Samples	103		
	Avg. Daily Variations [%]	[0.049, 0.111]		
	Specific News [%]	[14.17, 27.18]	[3.68, 7.30]	[1.84, 3.65]

Table 4.1: Datasets used in the thesis

In this chapter we will analyze the results obtained for the three different architectures separately. For that purpose, we will use different plots of the accuracies, the losses and the benefits. To avoid repeated results,

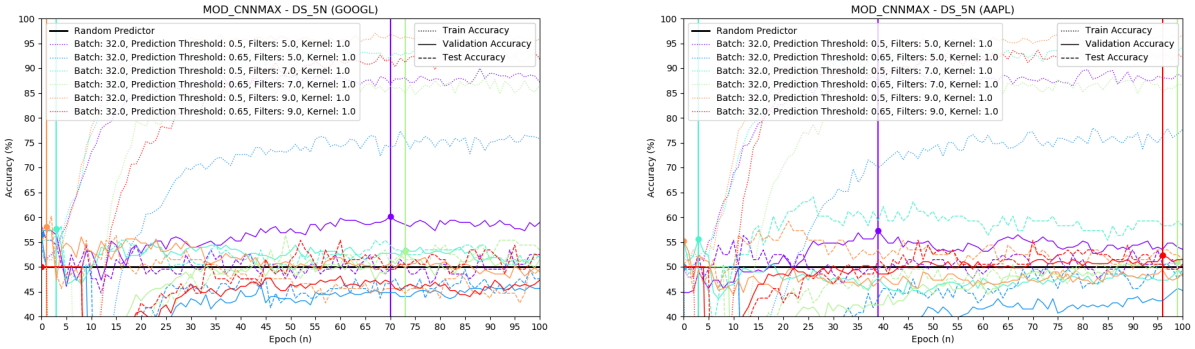
[3] Average daily variations for [GOOGL, APPL].

[4] Percentage of news containing the words 'google' and 'apple' in the title.

although a lot of different experiments have been performed, only a small selection of plots will be exposed in this thesis.

4.1 Convolutional Architecture

The first architecture and the lighter one uses a convolutional layer to find relationships between news and stock prices. Convolutions have been used in NLP since the beginning, competing against recurrent units in this type of tasks. First of all, let's compare the accuracies obtained for the two predicted stocks using the dataset of 5 news per date:



(a) Accuracies of the `GOOGL` stock

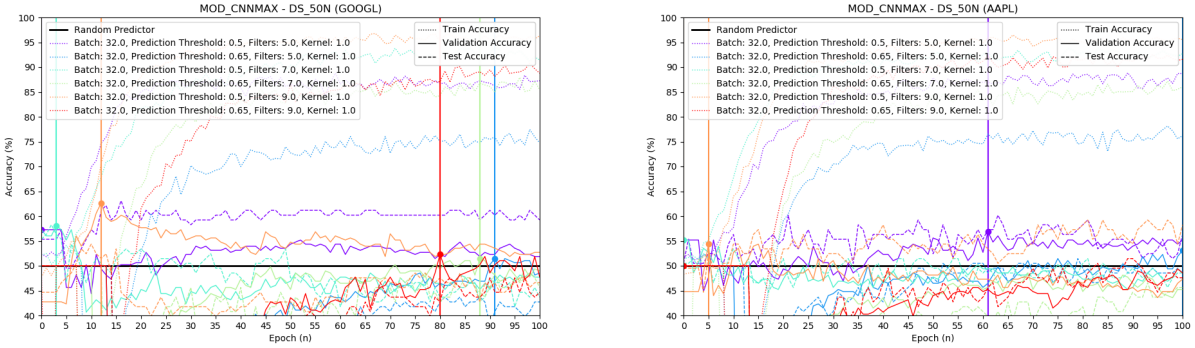
(b) Accuracies of the `AAPL` stock

Figure 4.1: Accuracies of the Convolutional Architecture using the 5n_DS dataset

As we can see, we experiment with six different combinations of hyperparameters. In detail, we set the kernel size of the convolution to 1 and experiment with 5, 7 and 9 filters for the prediction thresholds of 0.5 and 0.65. The reason to set the kernel size to 1 in this experiment is because higher values have proved to work much worse in previous experiments, while altering the number of filters seems to work slightly better.

When seeing this two plots we can observe that the purple line (which represents the model with 5 filters and the prediction threshold set to 0.5) is the best predictor in terms of validation accuracy, reaching almost 60% for the `GOOGL` stock in the epoch 70. In fact, we can also see that this configuration of the neural network is the only one which does not archive the best results on the first epochs, as the other architectures with more parameters seem to overfit the training dataset pretty fast even with the dropout probability set to $p_{drop} = 0.75$.

Now, let's see what happens when the used dataset contains 50 news per date instead of only 5:



(a) Accuracies of the `GOOGL` stock

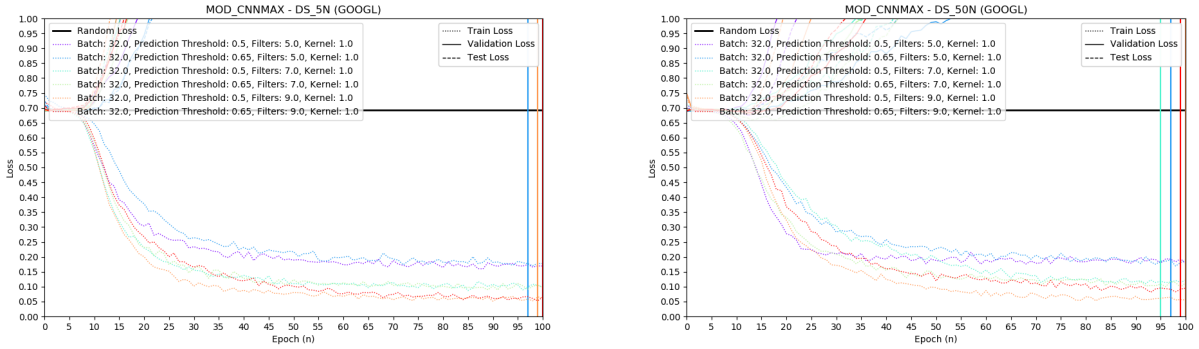
(b) Accuracies of the `AAPL` stock

Figure 4.2: Accuracies of the Convolutional Architecture using the 50n_DS dataset

Something interesting seems to happen. While with 5 news we obtained a better result for the validation splits, when we increase the number of news to 50 the result of these decrease but the results obtained for the test

splits seem to perform much better than in the previous case. In this case, the configuration that seems to work better is the same that the previous one, with an orange peak on the GOOGL plot that seems to be a false positive as it does not generalize well.

Another key aspect to understand what is happening when training a deep learning model is the loss. Let's compare what happens with it using the two previous datasets:



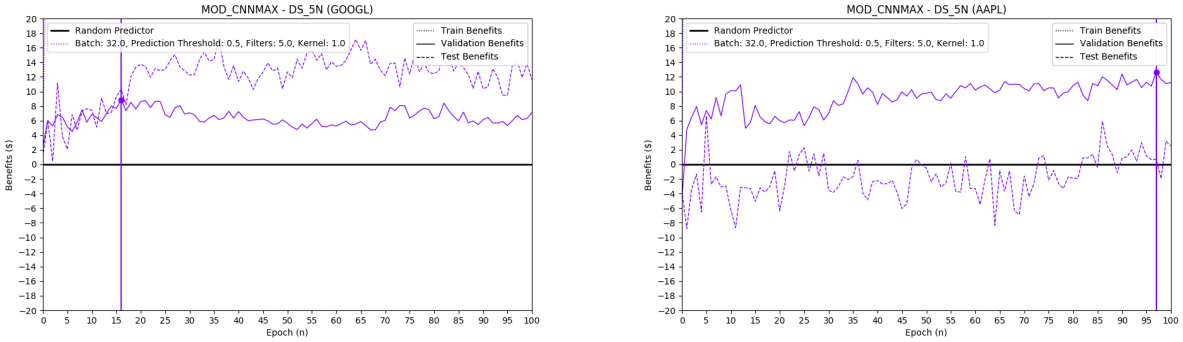
(a) Losses of the 5n_DS dataset

(b) Losses of the 50n_DS dataset

Figure 4.3: Losses of the Convolutional Architecture for the GOOGL stock

As we can see, the trend of both plots is noticeable: the training loss decreases while the validation and test losses increase. The first impression of this result could be contrary to the obtained accuracy results. How is it possible that we obtain an increase in the accuracy while doing worse and worse in terms of loss? The answer is simple: the neural network performs well on situations observed in the training split while it dramatically miss non-seen scenarios. This result could indicate the averaging word embeddings is not a good procedure to obtain sentence embeddings, so it would be interesting to try with different approaches in the future. We can also see that the loss of the dataset with 50 news increases with a smaller slope than the dataset with 5 news, which is an intuitive result as it is harder to overfit larger training datasets. [1]

However, the most important number to take into account is not the accuracy, but the benefit that we could obtain with our predictors. The next figures represent the daily benefit that we would obtain using the purple architecture following the procedure explained in the methodology chapter:



(a) Benefits of the GOOGL stock

(b) Benefits of the AAPL stock

Figure 4.4: Benefits of the Convolutional Architecture using the 5n_DS dataset

As we can see, while the predictions of the GOOGL stock are always positive for both the validation and test splits, the predictions of the AAPL stock are only positive for the validation samples. This last situation happens in all the other performed experiments. One explanation to this effect could be that we are not using important information that is affecting the predictions of the test samples. However, in this thesis we focus on using only textual news as data, so no broader explanation can be given.

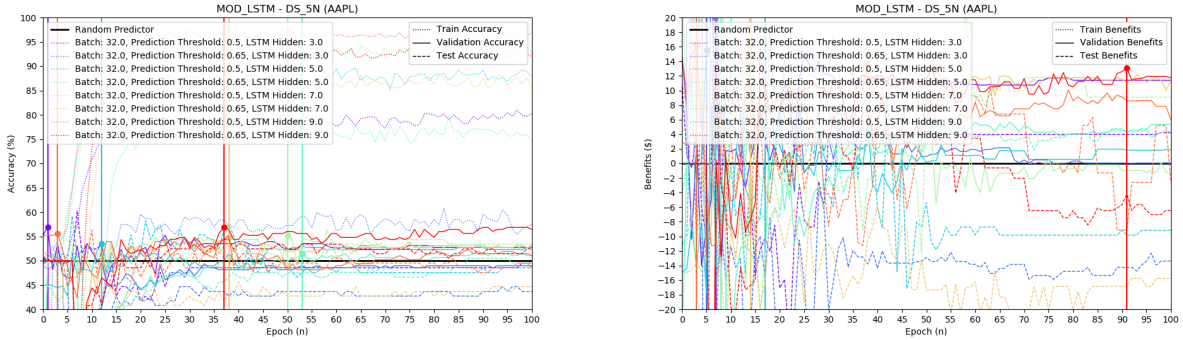
[1] This situation is the same for the three architectures, so no more loss plots are compared in the next pages.

There is another curious situation, which is that for the GOOGL stock the test benefit seems to be pretty good while the accuracy is not that good. This may have a very simple explanation, and that is that our model performs pretty well on predicting big variations on the stock while it behaves almost randomly for small changes.

4.2 Recurrent Architecture

The second architecture uses a network of LSTM cells to model the information contained in the news and relate it with stock variations. In theory, a LSTM cell is capable of remembering long-term relationships between its inputs. As our inputs are the sentence embeddings obtained by averaging the pre trained word embeddings, it should be able to find the most remarkable information of the set.

First of all, we will analyze the results obtained for the 5n_DS and the AAPL stock:



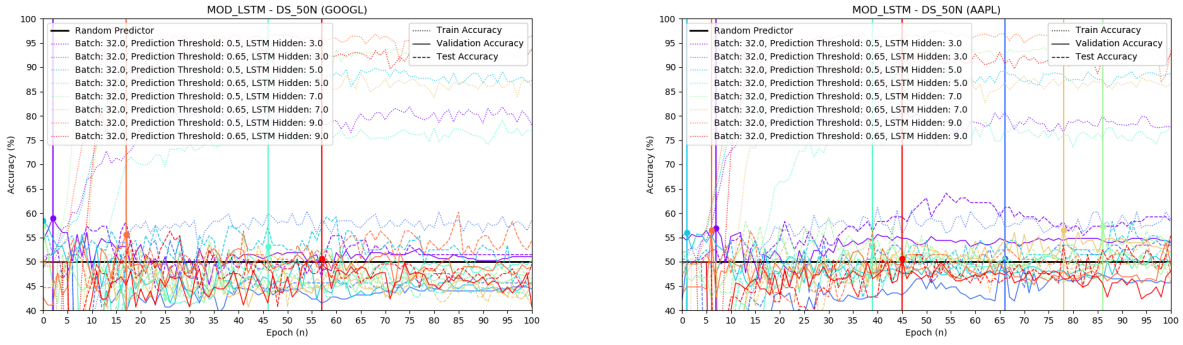
(a) Accuracy of the AAPL stock

(b) Benefits of the AAPL stock

Figure 4.5: Results of the Recurrent Architecture using the 5n_DS dataset

As we can see, the first epochs of both plots are so chaotic for all the combinations of hyperparameters. However, the lines start to be distinguishable from the epoch 40 onwards. In terms of accuracy, the best configuration seems to be the red line, with a prediction threshold of 0.65 and a hidden layer of size 9. The same happens with the benefits plot, archiving a benefit of about 14$ per day. However, we have the same situation that happened with the convolution architecture, and that is that the benefit of the test split is negative. In fact, we can see that while almost all the validation lines are above the random predictor, the test lines are under it.

One of the most interesting experiments of this type of recurrent architecture is to see what happens when we increase the amount of cells, which is the direct consequence of increasing the number of news per date. In the following plots we have the accuracies obtained using 50 news per date:



(a) Accuracy of the GOOGL stock

(b) Accuracy of the AAPL stock

Figure 4.6: Accuracies of the Recurrent Architecture using the 50n_DS dataset

The result is disastrous. While with 5 news per date we obtained a positive tendency, with 50 news per date we obtain predictors with worse performances than the random predictor. This situation leads to a big conclusion, which is that the long-term memory of LSTM does not help on finding relationship between news.

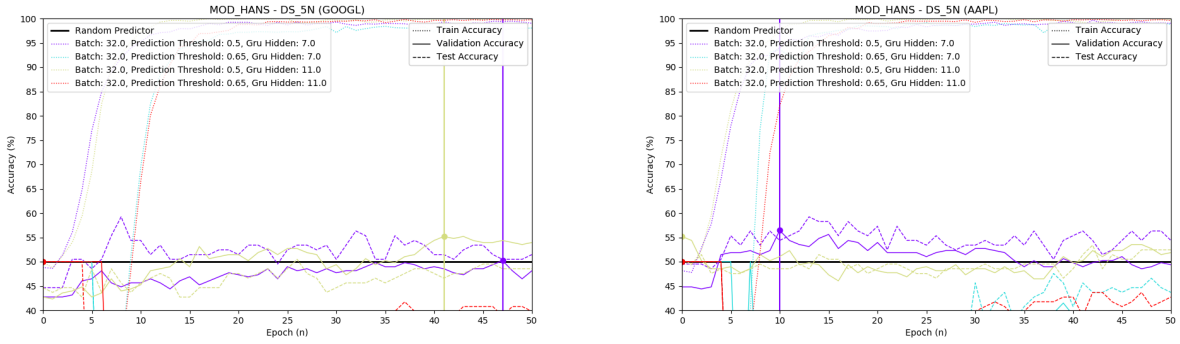
The last result is also obtained when dealing with the 25n_ds, so we can conclude that this type of recurrent architecture performs much better when the quantity of news is small.

Finally, it is remarkable that the best result of this architecture is archived using the prediction threshold of 0.65 instead of the 0.5 of the convolutional architecture. It seems that with a small quantity of news, the LSTM is capable of increasing the probability of the correct class, which in a real situation would lead to a more secure trading operation. It has not been included in the plots, but in this and all other experiments the number of samples that overcome the threshold increase as the epoch increases. This could be another sign of overfitting, as it is seeing the same cases over and over again and learns that certain situations have a very specific result.

4.3 Attention Architecture

The attention architecture uses GRU cells as core, which are followed by a weigh mechanism that tries to detect the most relevant outputs of the set of cells. One big difference of this architecture with respect to the previous one is that this one uses the hidden states of all the cells, while the previous one only uses the last hidden state as a predictor.

As we have seen so far, less news per date use to return better results. So let's analyze the accuracy of our two predicted stocks:



(a) Accuracy of the `GOOGL` stock

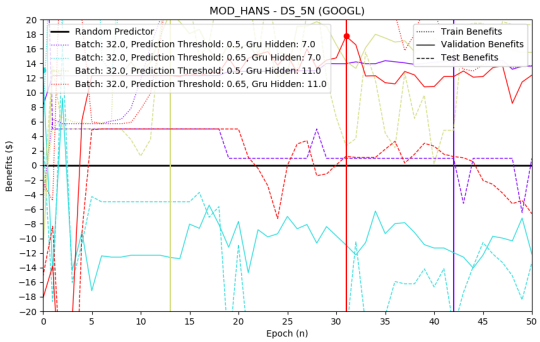
(b) Accuracy of the `AAPL` stock

Figure 4.7: Accuracies of the Attention Architecture using the 5n_DS dataset

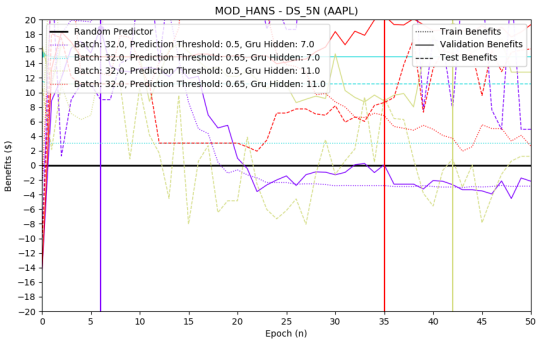
Notice that we have tested with less combinations of hyperparameters as this architecture is much more computationally expensive to train than the previous two. However, this computational cost does not seem to be translated into better results, but much worse accuracies. The only model that seems to have a slightly positive tendency is the one represented with a purple line, which corresponds to a prediction threshold of 0.5 and a hidden size of 7.

One remarkable fact that seems to be present in the three tested architectures is that the smallest hyperparameters use to be the ones with better results. This discovery has a very simple explanation, which is that a small training dataset requires a small neural network if we do not want to overfit the samples, which is the effect we have had in all our simulations.

However, as we have seen in previous results, the accuracy is not a good indicator of the performance of a stock market predictor. It is better to analyze the results of the benefits simulations:



(a) Accuracy of the `GOOGL` stock



(b) Accuracy of the `AAPL` stock

Figure 4.8: Benefits of the Attention Architecture using the 5n_DS dataset

Once again, the results seem too arbitrary to take conclusions. While certain combinations of hyperparameters seem to have positive tendency for the validation split, they have negative tendency for the test split and vice versa. The results are too bad to be considered in a real life scenario.

The plots of the other two datasets are equally bad, so to avoid filling this thesis with repeated content we do not include them.

Chapter 5

Budget

This project is software-based, so the costs related with it are basically the salary of the engineers and the used services.

There are two engineers implicated in the project, one considered as junior (the author) and another considered as senior (the supervisor). The dedication of the junior engineer is about 4 hours per day, while the senior's is about 2 hours per week.

In terms of software, only one paid software has been used in this project: the PyCharm IDE. This software has a monthly subscription instead of the one-time purchase.

The last items are the server where the database is hosted and the servers used to train the models. For the first case we use a VPS of OVH.com, however, to train the models, in this project we have used a server of the Center for Language and Speech Technologies and Applications (TALP), which has had zero cost. To simulate a real-life scenario, we will use Google Cloud Machine Learning Engine as a price reference.

The duration of the project is set to 17 weeks, which is about 4 months of work.

	Quantity	**Wage/hour**	**Hours/week**	**Cost**
Junior Engineer	1	12.25 €	20	4165 €
Senior Engineer	1	22.00 €	2	748 €

	Quantity	**Price / unit**	**N. Units**	**Cost**
PyCharm	2	19.90 € / month	∼ 4 months	159.2 €
VPS	1	7.25 € / month	∼ 4 months	29 €
GCloud MLE	1	1.15 € / hour	50 hours	57.5 €

Total Cost: 5158.7 €

Table 5.1: Budget of the Project

Note 1: All of this costs are from the business point of view, that is to say, the engineer numbers reference gross wages and the others prices include the Value-Added Tax (VAT).

Note 2: We estimate the wages per hour assuming a full-time gross annual salary of 25.000€ and 45.000€ for the junior and the senior engineers, respectively.

Note 3: The Google Cloud MLE cost is supposing the Basic_GPU tier of the USA. The currency has been exchanged from USD to EUR using the quotes of the day it has been writen.

Chapter 6

Conclusions

In this thesis we have tried to predict the stock market using news. For this purpose, we have divided the project in two big parts: one regarding with the data acquisition and the other with the data modelling. For the first block we have created a different types of scrappers that are able to obtain updated information and store it in an foolproof relational database. Then, in the second block, we have experimented with different types of shallow neural networks that model the relationship between news titles and variations of the `GOOGL` and `AAPL` stocks.

The first big problem we faced was the difficulty to obtain quality data. There are a few news datasets over there, however they contain old data which makes it difficult to predict current tendencies. Besides, data providers make it difficult to obtain their content without paying abusive prices; for example, Bloomberg charges more than a thousand dollars per month for accessing their stock market data console. And here it comes the first conclusion, which is that predict the stock market has the entry barrier of obtaining the data by yourself.

However, the big amount of things that we have learned developing this project have to do with the data itself. For example, we have discovered the importance of normalizing the data before it feeds a neural network. When we deal with unstructured data like text, the possible sparsity of the data makes it really difficult for a machine to learn. To reduce that sparsity, the concept of embedding came to the scene, which is just a mechanism to decrease the dimensionality of a dictionary of words by somehow clustering words with similar meanings. However, with this approach and the small quantity of data we had the problem was not nearly close to be solved. We could only do two things: normalize the data before it was transformed to embeddings and use pre-trained embeddings. These two solutions, although not perfect, improved the performance of our solution enough to make it work.

The mother of all problems has been the quantity of training data we had. The stock market has two big difficulties in terms of data: one is that we can only obtain one sample per stock per day, and the other is that the tendency of the market has a huge impact (something that influenced the market positively a few years ago may have a negative impact in the present). This lack of data make this problem a small data problem, which limits the possibilities in terms of neural architectures.

We have proved in the results chapter that the best architecture is in fact the simplest one: the convolutional architecture. This leads to a very powerful conclusion, which is that the quantity of training data must be equated to the deepness of the architecture so it performs as excepted. Besides, tuning the hyperparameters has resulted to be the second most important thing of an architecture, just after the election of the architecture itself.

With this last architecture we have also seen that the only kernel size that seemed to worked was 1. The effect of a bigger kernel size would mix the different sentence embeddings, which in fact could be a good idea as the variations of a certain stock should be produced by the mix of all the news. This result could give us a big clue of what makes the stock market go up or go down, which is that it is mainly affected by individual news instead of combinations of them. A deeper study about this could prove one of our initial hypothesis, which is that the stock market is mainly affected by only one or two events every day.

We have also concluded that the accuracy is not a good indicator of the behavior of a stock market predictor, as not all days contribute equally to the final profit. In fact, it is not important if our predictor does not perform

well on days with a small variation in the price, but it could be catastrophic if it misses days with huge changes. For this purpose, the profit test is a much better indicator of how well a model is performing. As can be seen in the results section, a bad accuracy model can have a good benefit, which is in fact the result that we were looking for.

The weak point of this project has probably been the creation of sentence embeddings from word embeddings. Following the procedure of Peng and Jiang [9] and Vargas et al. [12], we decided to just average the 300 dimensions of the word embeddings to obtain a fixed-length representation of a new title. We have proved that this approach is not optimal, as it does not generalize well the events that they represent.

Another big surprise of this thesis has been poor performance of recurrent units with respect to convolutions. It has been said that, for NLP task, recurrent units are the best options as they are able to find long term relationships between inputs. However, for our problem, the relationship between titles was not that important, but the detection of certain events in individual titles was the key so we could create a good predictor. For this task, convolutional units seem to generalize better and obtain better results.

Future Work and Recommendations

This thesis focused on a very specific type of information, which are news. However, in a real-life scenario, a lot of more types of information affect the stock market with the same or even more importance than the news. In future works, the idea would be to include as much types of different data as possible so we could create a smarter predictor.

Besides, we have focused on using information with only one day of delay. Vargas et al. [12] proved that the impact of the information decays as it becomes older, however, Ding et al. [2] proved that we can obtain a better result if we use older data.

In terms of recommendations, I would recommend the people who want to work on stock market predictors to focus all they efforts on data. Obtaining the right data is the most important point of all, as it would be impossible for a neural network to learn from uncorrelated data.

Finally, I would like to invite all the people interested in the stock market and reading this thesis to email me if they want to collaborate in future works.

Bibliography

[1] Khaled A Althelaya and Salahadin Mohammed. Evaluation of Bidirectional LSTM for Short- and Long-Term Stock Market Prediction. pages 151–156, 2018.

[2] Xiao Ding, Yue Zhang, Ting Liu, and Junwen Duan. Deep Learning for Event-Driven Stock Prediction. In *IJCAI International Joint Conference on Artificial Intelligence*, 2015. ISBN 9781577357384. URL https://www.ijcai.org/Proceedings/15/Papers/329.pdf.

[3] Eugene F. Fama. The Behavior of Stock-Market Prices. *The Journal of Business*, 38(1):34, 1965. ISSN 0021-9398. doi: 10.1086/294743. URL http://www.jstor.org/stable/2350752.

[4] Google Inc. Word2Vec. URL https://code.google.com/archive/p/word2vec/.

[5] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982. ISSN 0027-8424. doi: 10.1073/pnas.79.8.2554. URL http://www.pnas.org/cgi/doi/10.1073/pnas.79.8.2554.

[6] Jaegwon Kim. *Supervenience and mind : selected philosophical essays*. 1993. ISBN 0521433940\r0521439965 (pbk.). doi: 10.1017/CBO9780511625220.

[7] Burton G Malkiel. A Random Walk Down Wall Street. *Foundations*, 13:464, 1973. ISSN 09500804. doi: 10.1111/1467-6419.00091. URL http://www.gradneyvistica.com/DownloadFiles/ARandomWalkAbstract.pdf.

[8] Tomas Mikolov, Greg Corrado, Kai Chen, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *Proceedings of the International Conference on Learning Representations (ICLR 2013)*, 2013. ISSN 15324435. doi: 10.1162/153244303322533223.

[9] Yangtuo Peng and Hui Jiang. Leverage Financial News to Predict Stock Price Movements Using Word Embeddings and Deep Neural Networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015. ISBN 9781937284961. doi: 10.3115/v1/D14-1148. URL http://arxiv.org/abs/1506.07220.

[10] Kira Radinsky, Sagie Davidovich, and Shaul Markovitch. Learning causality for news events prediction. In *Proceedings of the 21st international conference on World Wide Web - WWW '12*, page 909, 2012. ISBN 9781450312295. doi: 10.1145/2187836.2187958. URL http://dl.acm.org/citation.cfm?doid=2187836.2187958.

[11] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 2014. ISSN 15337928. doi: 10.1214/12-AOS1000.

[12] Manuel R. Vargas, Beatriz S. L. P. de Lima, and Alexandre G. Evsukoff. Deep learning for stock market prediction from financial news articles. In *2017 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA)*, pages 60–65, 2017. ISBN 978-1-5090-4253-1. doi: 10.1109/CIVEMSA.2017.7995302. URL http://ieeexplore.ieee.org/document/7995302/.

[13] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical Attention Networks for Document Classification. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, 2016. doi: 10.18653/v1/N16-1174. URL http://aclweb.org/anthology/N16-1174.

Appendices

Appendix A

Database Design

In this first appendix we will expose and explain the design of the database of the project. As mentioned in the introduction, the stock market requires to maintain the integrity of the data, as we are dealing with money. For this reason several auxiliary tables and fields have been included in the design. To begin, let's see the block diagram and the relationship between tables:

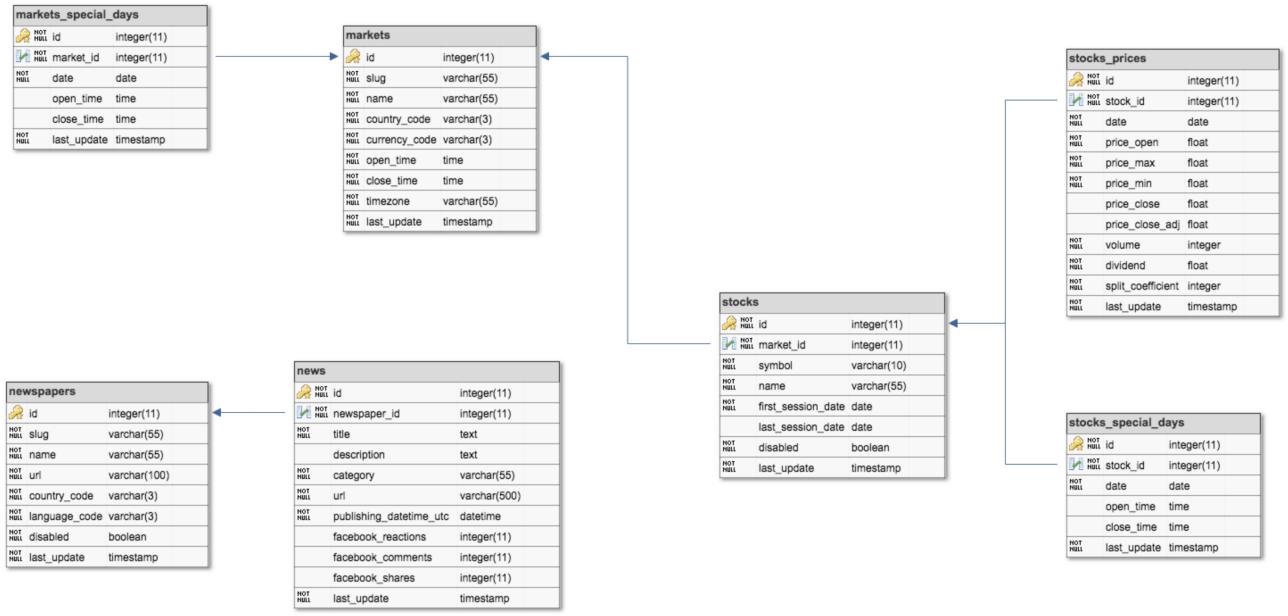


Figure A.1: Design of the Database

As can be seen, the database is composed by 7 different tables creating 2 different groups of data: one related with stock prices and the other with news.

The group of stock prices is composed by 5 tables:

- **market**: stocks can be traded in different markets. This table stores the information related with each one of them. The fields are a `slug`, which is a unique string identifying the market; a `name`, which is a readable identifier; the `country_code` and `currency_code`, which define the country and currency of the market; the `open_time` and `closing_time`, which define the trading hours; and the `timezone`, which establishes the relationship between the previous two fields and Coordinated Universal Time (UTC).
- **stocks**: in this table the different stocks of the system are saved. Every stock is identified with a `symbol` (for example, 'AAPL') and a `name` (for example, 'Apple Inc.'). Besides that, we must define in which `market_id` the stock is traded and the `first_session_date` of the Initial Price Offering (IPO). To deal with stocks that have stopped their activity or changed its name, we define the `last_session_date` field.

- **stocks_prices**: in this table daily prices of the stocks are saved. As in every relationship, the `stock_id` field defines the referenced stock. Then, for a given `date`, the fields `price_open`, `price_max`, `price_min`, `price_close`, `price_close_adj`, `volume`, `dividend` and `split_coefficient` are stored.
- **markets_special_days and stocks_special_days**: this two tables store days when a market or a stock has not had the usual trading period. For example, if a `market_id` closes on Christmas a new entry with the appropriate `date` and `open_time` and `close_time` set to null must be created. The same would happen if the market closes before the market `close_time`, in this case this last field must be filled accordingly.

As for the group of tables related with news, only 2 of them are required:

- **newspapers**: this table stores information of newspapers. As in a market, we define a `slug` (unique identifier) and a `name` (readable string) to identify the newspaper. Besides, a newspaper is associated with a website with a given `url` and is focused for the people of a `country_code` in a `language_code`.
- **new**: finally, this table stores all the news that have been scraped. Each new has been written in a `newspaper_id`. The information stored for each new is its `title`, `description`, `category`, `url` and `publishing_datetime_utc`. Besides, the fields `facebook_reactions`, `facebook_comments` and `facebook_shares` store the social information of the new.

In addition to unique identifiers, there are situations where a double unique identifier must be created. A double unique identifier imposes the restriction that the combination of two fields must be unique in a table. In this project, the following double unique identifiers have been used:

- The `stock_id` and `date` fields of the *stocks_prices* table.
- The `stock_id` and `date` fields of the *stocks_special_days* table.
- The `market_id` and `date` fields of the *markets_special_days* table.
- The `newspaper_id` and `url` fields of the *news* table.

As a last comment, the `disabled` field is used to ignore rows of a table while the `last_update` field is used to check the last time a row was updated.

Appendix B

Recurrent Units

In this chapter we will explain the internal behaviour of the best known recurrent gates. Standard neural networks treat input elements individually, that is to say, the network does not directly model the possible sequentiality of the data although there may be a temporal or spatial relation between them. There are cases when the nature of the data is essentially an ordered sequence, so another type of neurons must be used.

With this idea, the first recurrent network was the Hopfield network [5]. With the past of the time recurrent units have evolved, improving their performance every year.

The general design of a recurrent network is the following:

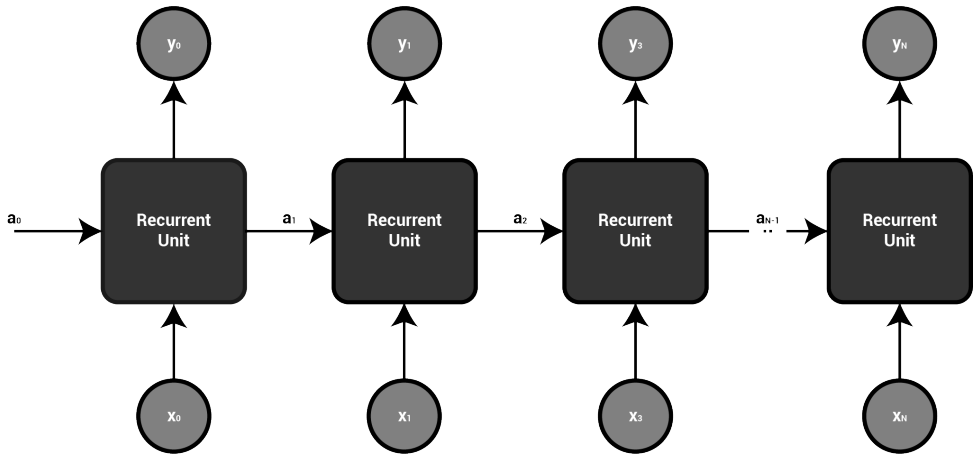


Figure B.1: Structure of a RNN

The idea is that as the information is a sequence, we will obtain a better prediction of $y^{<t>}$ if somehow we use information not only from $\boldsymbol{x}^{<t>}$ but also from $\boldsymbol{x}^{<t-1>}$ and previous samples. The most basic design of a recurrent unit follows the next equations:

$$\begin{aligned} \boldsymbol{a}^{<t>} &= f(\boldsymbol{W}_{aa}\boldsymbol{a}^{<t-1>} + \boldsymbol{W}_{ax}\boldsymbol{x}^{<t>} + \boldsymbol{b}_a) \\ \boldsymbol{y}^{<t>} &= g(\boldsymbol{W}_{ya}\boldsymbol{a}^{<t>} + \boldsymbol{b}_y) \end{aligned}$$

$$\begin{aligned} \boldsymbol{x}^{<t>} &\in \mathbb{R}^{M\times 1} \\ \boldsymbol{y}^{<t>} &\in \mathbb{R}^{M'\times 1} \\ \boldsymbol{a}^{<t>} &\in \mathbb{R}^{M'\times 1} \\ \boldsymbol{W}_{ax} &\in \mathbb{R}^{M'\times M} \\ \boldsymbol{W}_{aa} &\in \mathbb{R}^{M'\times M'} \\ \boldsymbol{W}_{ay} &\in \mathbb{R}^{M'\times M'} \\ \boldsymbol{b}_a &\in \mathbb{R}^{M'\times 1} \\ \boldsymbol{b}_y &\in \mathbb{R}^{M'\times 1} \end{aligned}$$

However, this basic unit has a big problem: vanishing gradient. For long sequences the network is not able to remember what happened before, so new mechanisms should be used.

B.1 GRU (Gated Recurrent Unit)

The GRU is one of the most used units of the current times. It defines the concept of gates, which intuitively define the change that the hidden state of a unit in every time step. Specifically, the GRU defines two different gates:

$$\boldsymbol{\Gamma}_u = \sigma\Big(\boldsymbol{W}_{ux}\boldsymbol{x}^{<t>} + \boldsymbol{W}_{uc}\boldsymbol{c}^{<t-1>} + \boldsymbol{b}_u\Big) = \sigma\Big(\boldsymbol{W}_u[\boldsymbol{x}^{<t>}, \boldsymbol{c}^{<t-1>}] + \boldsymbol{b}_u\Big)$$
$$\boldsymbol{\Gamma}_r = \sigma\Big(\boldsymbol{W}_{rx}\boldsymbol{x}^{<t>} + \boldsymbol{W}_{rc}\boldsymbol{c}^{<t-1>} + \boldsymbol{b}_r\Big) = \sigma\Big(\boldsymbol{W}_r[\boldsymbol{x}^{<t>}, \boldsymbol{c}^{<t-1>}] + \boldsymbol{b}_r\Big)$$

As we can see a gate is just a combination of a linear plus non linear function using the signal and the previous state as inputs and a weight matrix and bias vector as trainable parameters. Once this two gates are computed, the following two equations return the final state of the cell:

$$\tilde{\boldsymbol{c}}^{<t>} = tanh\Big(\boldsymbol{W}_{cx}\boldsymbol{x}^{<t>} + \boldsymbol{W}_{cc}(\boldsymbol{\Gamma}_r * \boldsymbol{c}^{<t-1>}) + \boldsymbol{b}_c\Big) = tanh\Big(\boldsymbol{W}_c[\boldsymbol{x}^{<t>}, \boldsymbol{\Gamma}_r * \boldsymbol{c}^{<t-1>}] + \boldsymbol{b}_c\Big)$$
$$\boldsymbol{c}^{<t>} = \boldsymbol{\Gamma}_u\tilde{\boldsymbol{c}}^{<t>} + (1 - \boldsymbol{\Gamma}_u)\boldsymbol{c}^{<t-1>}$$

Intuitively, we can see that $\boldsymbol{\Gamma}_r$ is used to modify the previous state, maybe because the cell has detected that something has changed and must be notified to further cells. On the other hand, $\boldsymbol{\Gamma}_u$ has a more generalist use as it contains how much the state changes with respect to the previous one one applied the non-linear functions.

B.2 LSTM (Long Short-Term Memory)

The LSTM is an improved version of the GRU, or better said, a most general case of it. In this case, the output are two different states computed using three different gates:

$$\boldsymbol{\Gamma}_u = \sigma\Big(\boldsymbol{W}_{ux}\boldsymbol{x}^{<t>} + \boldsymbol{W}_{ua}\boldsymbol{a}^{<t-1>} + \boldsymbol{b}_u\Big) = \sigma\Big(\boldsymbol{W}_u[\boldsymbol{x}^{<t>}, \boldsymbol{a}^{<t-1>}] + \boldsymbol{b}_u\Big)$$
$$\boldsymbol{\Gamma}_f = \sigma\Big(\boldsymbol{W}_{fx}\boldsymbol{x}^{<t>} + \boldsymbol{W}_{fa}\boldsymbol{a}^{<t-1>} + \boldsymbol{b}_u\Big) = \sigma\Big(\boldsymbol{W}_f[\boldsymbol{x}^{<t>}, \boldsymbol{a}^{<t-1>}] + \boldsymbol{b}_f\Big)$$
$$\boldsymbol{\Gamma}_o = \sigma\Big(\boldsymbol{W}_{ox}\boldsymbol{x}^{<t>} + \boldsymbol{W}_{oa}\boldsymbol{a}^{<t-1>} + \boldsymbol{b}_o\Big) = \sigma\Big(\boldsymbol{W}_o[\boldsymbol{x}^{<t>}, \boldsymbol{a}^{<t-1>}] + \boldsymbol{b}_o\Big)$$

Once the gates have been calculated, we can compute the candidate state $\tilde{\boldsymbol{c}}^{<t>}$ and the two returned states:

$$\tilde{\boldsymbol{c}}^{<t>} = tanh\Big(\boldsymbol{W}_{cx}\boldsymbol{x}^{<t>} + \boldsymbol{W}_{ca}\boldsymbol{a}^{<t-1>} + \boldsymbol{b}_c\Big) = tanh\Big(\boldsymbol{W}_c[\boldsymbol{x}^{<t>}, \boldsymbol{a}^{<t-1>}] + \boldsymbol{b}_c\Big)$$
$$\boldsymbol{c}^{<t>} = \boldsymbol{\Gamma}_u * \tilde{\boldsymbol{c}}^{<t>} + \boldsymbol{\Gamma}_f * \boldsymbol{c}^{<t-1>}$$
$$\boldsymbol{a}^{<t>} = \boldsymbol{\Gamma}_o * \boldsymbol{c}^{<t>}$$

As we can see that $\boldsymbol{\Gamma}_u$ is used to define how much we have to update the state, while $\boldsymbol{\Gamma}_f$ is more about how much we have to forget. Finally, the second state, $\boldsymbol{a}^{<t>}$, is just the element-wise product of the first state with the $\boldsymbol{\Gamma}_o$ gate.

Glossary

AI Artificial Intelligence. 11

API Application Programming Interface. 18

CNN Convolutional Neural Network. 13

GRU Gated Recurrent Unit. 14, 27, 37

HAN Hiererchical Attention Network. 14

IPO Initial Price Offering. 34

LSTM Long Short-Term Memory. 11, 14, 26, 27, 37

NLP Natural Language Processing. 11, 24, 31

ORM Object Relational Model. 17

RNN Recurrent Neural Network. 13

SVM Support Vector Machine. 11

TALP Center for Language and Speech Technologies and Applications. 29

UTC Coordinated Universal Time. 34

VAT Value-Added Tax. 29

VPS Virtual Private Server. 17, 29