BABEŞ-BOLYAI UNIVERISTY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION COMPUTER SCIENCE

Diploma Thesis

STOCK MARKET PREDICTION

Supervisor:
Lect. Dr. Tudor Mihoc

Author:
Flaviu-Andrei Jurj

2020

Abstract

Stock market has been a studied domain for a long time especially due to the attention received from investors. More specifically, stock market prediction is known for its complexity and volatility. Initially, the task was done solely by humans, but lately with the technological evolution more and more operations are automated including the actual prediction. In the beginning of artificial intelligence, basic methods were used to complement statistical methods. Moving forward through the timeline more methods were tested and as with most domains in which artificial intelligence was used the current standard involves more or less machine learning.

This thesis will approach the stock market prediction problem as a time series forecasting one. It will be seen from the two classic perspectives: as a classification problem predicting only a general trend such as increase, decline or stagnation and as a regression problem predicting a set of prices during different time horizons. The problem will be solved from a deep learning perspective experimenting with different architectures which will mainly consist of specific variations of recurrent neural networks which have proven their effectiveness in other problems from other domains: long short-term memory networks and gated recurrent unit networks.

In terms of results, ... TODO

This work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.

Contents

List of Figures

Chapter 1

Introduction

The stock market can be defined as a loose network of economic transactions of stocks (also called shares) which represent ownership claims on businesses. The act of investing has changed a lot during recent years becoming more and more automated through various electronic platforms which even try to predict the actual movement of the market.

There are many strategies which may be adopted by an investor, but ultimately they reduce to the most important task: the capability of predicting to some degree the movement of the market. In this thesis, we will evaluate the reliability of machine learning for tackling this problem. Technical analysis seeks to determine the future price of a stock based only on the previous trends of the price without taking into account details and fundamentals of the company.

This thesis will present a generic perspective regarding the problem of stock market prediction. A section concerning the recent techniques and related work will drive some of the information for the future problem solving. A comparison will be attached regarding the possible variations and various architectures will be compared. Finally, an in-depth presentation of the results will be followed by the lifecycle of developing an application which will incorporate the previous results, going through each stage and phase necessary.

My solution and results ... TODO

In the following chapters, a complete perspective will be given to stock market prediction. The thesis starts with a serious of subsections regarding the stock market fundamentals, difficulties and controversy. This section will build the necessary information while responding to some of the known arguments regarding stock market prediction. Further on, the scientific problem will be presented with its variations (classification and regression). The methodology regarding the used and preprocessed training data will be complemented by the proposed approach and the artificial intelligence concepts behind it. Finally, an in-depth display and comparison will be done regarding the obtained results and issues. Moving on, the next chapter will present some key concepts with respect to the technologies used in order to develop the application. The following set of sections will present the applied stages in the lifecycle development process starting with stakeholder requirements and advancing through various design stages reaching the implementation and end goal - the user manual. Ultimately, a small chapter of conclusions will be presented summarizing the contributions and results obtained, ending with suggested further improvements.

Chapter 2

Stock Market

The following chapter will present the main points regarding the problem solved by this thesis. Initially, key concepts and fundamentals will be introduced in order to gain the minimum financial information. A short summary will introduce after the generic techniques used in stock market prediction alongside some difficulties. In the second main section, a scientific approach will be taken in to offer the problem definition and its variations and some methodologies regarding the training data. Furthermore, the proposed approach will be detailed alongside some necessary theoretical aspects and finally finishing it with the obtained results.

2.1 Stock Market

2.1.1 Stock Market Fundamentals

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. It was created a long time ago in order to facilitate the raise of capital of companies, promoting a transparent way of trading regarding company assets. Nowadays, its main purpose is to regulate the exchange of stocks or other financial assets, ensuring a fair environment for both investors and corporations (whose stocks are traded in the market). It can be seen as the staple of the global financial system. The stock market created a dynamic system encouraging permanent innovation and improvement in every domain.

From an investor perspective, each action is the result of an investment strategy which contains a set of rules, behaviours or procedures. Although, there are many investment strategies, most of them can be classified in two separate groups: fundamental analysis and technical analysis. Fundamental analysis represents the analysis of a company's past performance as well as the credibility of its accounts through various factors and indicators based on the financial statements, business trends and general economic conditions. On the other hand, technical analysis is not concerned with any of the company's financial prospects and seeks to determine the future price of a stock based only the trends of the past price (a form of time series analysis).

For the following reasons, fundamental analysis is usually seen as a long-term strategy, while technical analysis is seen as a short-term one.

This thesis will tackle the topic of stock market prediction more from a technical analysis perspective being more suitable for automated non-subjective decisions. We will further evaluate the evolution of different methods and techniques reaching the latest trends of machine learning strategies, more specifically deep learning which at the moment is not well researched for financial time series forecasting research. [12]

2.1.2 Techniques in stock market prediction

Even though the stock market has a long history, only in the last couple of decades we have seen real development and research in terms of techniques to automate or aid the investor in the process of stock market prediction. At our current state, we can group all techniques in the following categories: statistical, pattern recognition, machine learning, sentiment analysis and hybrid.[13] At their core, they can be classified as mainly technical analysis, but they can also borrow some aspects from fundamental analysis.

Prior to the emergence of machine learning, statistical techniques were used which they often assumed linearity. However, they were mostly replaced step by step with other techniques which are more and more researched especially with the exponential growth in computational power. All the other categories may be considered to some degree as machine learning, but they are split accordingly due to the fact that they usually have different goals in mind. While pattern recognition works mostly on predicting certain figures or shapes which repeat themselves with unknown periodicity, machine learning category covers many techniques which have some degree of similarity and not being different enough to represent an entire category. Sentiment analysis represents also a trending methodology which have gained a lot of momentum lately even in financial time series analysis, but it is still mostly researched for recommendation systems.

One subcategory of machine learning which haven't been intensively researched due to its only recent success in other domains is represented by deep learning. Deep learning has been a major breakthrough in many domains such as object detection, speech recognition, natural language processing and so on. This thesis will approach the problem of stock market prediction as a time-series problem using the company historic data to predict on a short-term horizon the price or the trend. We will compare the results from both classical perspectives as a regression and as a classification.

2.1.3 Difficulties and controversy

Stock market has been a studied domain for a long time generating debates and controversy regarding whether it is possible or not to consistently predict its movement. The prediction problem is still an open problem due to its complexity taking into account the volatility of the stock market.In the following paragraphs, we will discuss about two theories which have generated controversy among people.

Chaos theory is a branch of mathematics focusing on the study of chaos - states of dynamical systems whose apparently random states of disorder and irregularities are often governed by deterministic laws that are highly sensitive to initial conditions. One underlying principle of chaos, also called the 'butterfly effect', describes how a small change in one state of a deterministic nonlinear system can result in large differences in a later state. Chaotic behaviour exists and is mostly characterized in natural systems such as weather, climate or even heartbeat irregularities. For this reason, many consider that the weather forecast for example can be considered accurate only in the following 2-3 days.

Coming back to our domain of interest, chaos theory is seen only as a spontaneous occurrence in some systems with artificial components such as the stock market and road traffic. While there exists some research studying the effects of chaos theory in economic and financial systems, the empirical literature that tests for chaos in economics and finance presents very mixed results.[3] There is little consensus that the chaos theory may only illustrate sudden shocks and crashes of the market which happen very rarely and can be considered insignificant. During the stock market history, there existed unpredictable events called 'black swan' which were characterized by their extreme rarity and severity impact, but no real linkage with the chaos theory has been done.

The second hypothesis of which we are going to discuss is more closely related to the stock market domain and is called the 'efficient-market hypothesis'. The 'efficient-market hypothesis' is a hypothesis in financial economics that states that asset prices reflect all available information at a given time. This would basically imply that all publicly known information about a company, which obviously includes its price history, would already be reflected in the current price of the stock. Accordingly, changes in the stock price reflect release of new information, changes in the market or random movements around the value that reflects the existing information set. Burton Malkiel, in his influential 1973 work 'A Random Walk Down Wall Street', claimed that stock prices could therefore not be accurately predicted by looking at price history. As a result, Malkiel argued, stock prices are best described by a statistical process called a "random walk" meaning each day's deviations from the central value are random and unpredictable.

However, investors and researchers have disputed the hypothesis both empirically and theoretically. For example, Warren Buffet who is considered by many one of the most successful investors in the world rebutted this hypothesis in its speech in 1984.[4] Moreover, there are accepted events which are considered 'stock market anomalies' by the hypothesis since they are violations in which consistently abnormal returns could have been earned by some investment strategies that are constructed based on potential market inefficiencies. Not lastly, there are imperfections in the financial markets which are attributed by behavioural economists to a combination of cognitive biases such as overreaction, overconfidence, information bias and many others.

2.2 Scientific Problem

2.2.1 Problem definition

Stock market prediction is the act of trying to determine the future value of a company stock or other financial instrument traded on an exchange. In terms of a general scientific category, the problem will be evaluated from the perspective of a time series. Time series analysis and time series forecasting are the two linked activities which are done over a time series. However, while time series analysis is mostly linked to statistics, time series forecasting has evolved as being mostly referred with machine learning in mind. In comparison with other problems, time series require more attention regarding the processing of data since it has a natural time ordering which must not be broken in any way.

The problem of stock market prediction may also be split depending on what information we want to obtain: regression if we want a continuous output variable such as the closing price of some stock or classification if we only want a hint regarding the future trend. This thesis will try to use both categories for different time horizons such as 1-day, 3-day, 5-day etc. For regression there are two main versions from which we will choose only the first:

- single point in future of the closing price for a company stock
- set of points corresponding to each day in the time horizon - allowing us to see the general flow chart of the company stock price

For the classification category, in terms of methodology this thesis will focus only the binary classification version ("buy/sell"). Other introduced classes such as stagnation may be defined, but they are rather hard to define and split based on other parameters. Based on all this variants, we are going to compare and decide how should we interpret all results into an automated algorithm which decides whether to buy or sell.

2.2.2 Methodology in training data

The following subsection will present the details and methodologies behind choosing and preparing the data for future training. As mentioned already in a previous section, stock market prediction may make use of various information in order to attempt forecasting future prices depending on many factors such as the time horizon.

Because we are not going to predict intraday evolution of prices (which is usually done in fast markets as Forex), we are going to use the classic data which is provided for evaluating a company stock value. It is available daily and contains five measurements abbreviated usually as OCHLV:

1. Open - the value of a single company stock at the beginning of the daily trading session
2. Close - the value of a single company stock at the end of the daily trading session
3. High - the maximum value of a single company stock obtained during the entire daily trading session
4. Low - the minimum value of a single company stock obtained during the entire daily trading session
5. Volume - the number of shares that were traded during the entire daily trading session

Based on this data, we could theoretically have a complete technical analysis taking into account only the company's past data. However, we will also add at least one stock market index representative for the market from which the company comes from. Stock market indexes can be seen as powerful indicators for global and country-specific economies. Because we are going to use stock data from popular American companies, we'll choose from S&P500, DowJonesIndustrial and NasdaqComposite. Using stock market indexes should help the network getting a grip of the general economic trend in addition to only the company's past data which may or may not reflect a lot the past economic trend of the general economy. In terms of actual data that is obtained from these stock market indexes, we are talking about the same five measurements mentioned earlier, but at the level of that stock market index (which is usually seen as a conglomerate of the most important companies of that market). It should be noted that is totally possible to trade only stock market indexes as opposed to trading individual company stocks.

In addition to those, we will attempt to incorporate some technical indicators in order to search whether the introduction of these features in the input data. They represent mathematical calculations based on some historic data (price, volume) and are aimed to aid the forecasting of future trends. They are widely used in categories such as pattern recognition with intraday trading.

In terms of preprocessing the data, we must standardize or normalize in order to aid the model in learning the relevant futures. Due to the fact that we evaluate a time-series problem, a particular attention should be given to the split between training and validation data. Because the validation data is considered to represent the unknown future, the strategy on which we standardize the data must be used only on the training data and applied with the same parameters obtained on the validation data. We are going to experiment with two different methodologies. First, with classic standardization which involves moving each feature values to a mean close to 0 and a standard deviation close to 1. Secondly, we are going to experiment with a min-max scaling which brings a set of values into the interval $[-1, 1]$. It is clear that in both cases, the validation data would not necessary respect the same strict results in the end because the validation data may contain values outside the training data set, but it still is highly important to bring the features much closer since we use data with various interval differences between them. For that reason, in terms of normalization we might choose an increased min-max interval on the training data to cover possible variations in the future data to some degree.

2.2.3 Related Work

Before diving into individual papers and results, we will present a recent state of the domain based on a literature review. Citing [13], we can group all the major techniques and recent advancements currently used in stock market prediction in five categories: statistical, pattern recognition, machine learning, sentiment analysis and hybrid. They mostly fall under the broader category of technical analysis, but a small portion of them might include elements of fundamental analysis also. In the following pages, we will focus on recent papers from all categories apart from the statistical one due to the following reasons: most techniques were used for a long time fine-tuning them only lately and there is emerging a new consensus that with enough care artificial intelligence techniques such as support-vector machine, neural networks may cover everything a statistical regression may compute. Before diving into them, we would like to mention that the results should not be compared directly due to the large degree of variety in terms of data and types of measurements for results.

Pattern Recognition

Pattern recognition has started as an activity to recognize patterns automatically which were usually considered important by experienced investors rather similar with the advancement in visual processing. The patterns are used mostly to identify future trends and are usually used during short periods of time. At its core, most of the work regarding pattern recognition can be seen as a subclass of template matching, but lately other concepts were introduced to improve the results due to the volatility of the market such as perceptually important points (PIP).

The first paper we are going to cite is "A dynamic trading rule based on filtered flag pattern recognition for stock market price forecasting" [1]. The flag pattern is considered to be a relatively common pattern found in the stock market. It consists of an initial clear trend followed by a consolidation period which is also range bound and it ends with the continuation of the initial trend. The flag can be observed as the consolidation period which takes a zigzag movement and offers a flag-like shape due to the bounded range movement. The paper distinguished itself through a variety of filters applied alongside a win-loss strategy while offering a relatively robust mechanism. The paper evaluated the trading on a large period on the DJIA index which stands for Dow Jones Industrial Average - a famous and popular index in the United States incorporating stocks from 30 different companies and representing a valuable resource to study the general trend of the market. They have used both short and medium terms: 15min and 1 day, while obtaining a lower risk overall with regard to previous similar research. An important aspect in their research besides the flag pattern was given by the usage of a very well-known technical indicator: exponential moving average (EMA). Moving averages are widely used in the stock market to reduce the noise inflicted in the market daily, while maintaining a general trend. Exponential moving average is an improved weighted version of the classical one, offering more relevance to recent prices and less relevance to older data. In terms of results, the accounted for the percentage return (profit) which has reached in the best case scenario a 280% return.

The other paper that we are mentioning for this category is "Pattern Matching Trading System Based on the Dynamic Time Warping Algorithm" [9]. One key feature of 'dynamic time warping' techniques is that they do not necessarily require a strict resemblance structure between the template and the actual pattern found. The speed in the stock market can be referred to different trend curves amplitudes which at their core are almost alike when eliminating some factor. The experiment determined the entry and exit points of trading by matching the daily index futures time series data with fixed patterns using dynamic time warping. There was experimentation with two different sets of fixed patterns - length 13 or 27 - using different training periods (sliding windows) and incorporating some exit-loss strategy. An interesting aspect to be mentioned is the time horizons used along the research while employing a relatively new concept: each trading day period was split in two groups. The morning was reserved for the trading data and acquiring information for the pattern matching, while the afternoon was used for the actual trading. In terms of results, they obtained an annualized return of 19.17, while employing a known measurement in finance: the 'sharpe ratio' - characterizes how well the expected return of an asset compensates the investor for the risk - obtaining a value of 0.94.

Machine learning

Machine learning has greatly evolved in the recent decade due to the evolution of computational cost and techniques, but it was present in the financial sector for a lot longer starting from support-vector machines, decision trees and progressively evolving to random forests, neural networks and so on. Both supervised and unsupervised learning is largely studied across. In the following papers, we will talk more in detail about recent techniques which incorporate deep learning methodologies. Deep learning has been the staple improvement in recent years regarding object detection or speech recognition working on the assumption of automated incremental feature detection. We are going to cite a recent paper summarizing focused deep learning in finance, offering a spectrum regarding the current trending movement [12]. As a standard, we may observe that the time series nature of our problem influenced the decision in using large margin recurrent neural networks models - mostly LSTM and GRU - which are known to maintain a context and create some long-term dependencies.

In order to present the power of basic RNN structures, we are going to choose two small papers regarding prediction of the stock market: [5] and [11]. The first study targeted the comparison between various (simple) neural networks structures for predicting Google assets on a five year time frame. The study requires minimum preprocessing of the OCLHV data with a min-max scaler. The past data window used consists of 30 days and found out that in general LSTM outperformed classic RNN or GRU, reaching a 72% accuracy binary prediction for a 5 day time horizon. The second study targeted a price regression prediction in comparison to the first one using the history data of the 'Nifty 50' stock market index - the representative index for the Indian stock market. Six years of OCLHV data was used with standard normalization as preprocessing and Root Mean Square Error (RMSE) was used as the result measurement with a testing (validation) value of 0.00859.

Another subcategory present for stock market prediction was the usage of reinforcement learning. Instead of price prediction, reinforcement learning uses experience gained through interacting with the environment and evaluative feedback to improve the ability of making decisions. As an addition, deep learning was introduced to occur along allowing for improved features creating deep reinforcement learning (DRL). We are going to mention here a recent paper that implemented a multi-objective DRL approach for intraday trading [14]. In order to address the high noise and non-stable financial data received when exploring the unknown environment, they chose a multi-objective deep reinforcement learning (MODRL) to simultaneously explore the environment, while recurrently making decisions. The neural network is split in two major parts: a set of four fully connected layers responsible for learning the features from financial data while reducing the uncertainty of the input data and a two-layer structured part for implementing the self-taught reinforcement trading composed of an LSTM and a fully connected layer. The multi-objective refers to the two objectives which are measured to obtain the performance of the model: the profit and associated risk. In addition, dropout techniques were used in the learning future part of the model to reduce overfitting. The model was tested on three different stock market indexes from China and when compared to other similar reinforcement learning techniques, managed to achieve improved performance overall with a 'sharpe ratio' of approximately 0.11.

Sentiment analysis

Sentiment analysis is a very recent subcategory in comparison with others due to many factors such as Internet consumption skyrocketing, social media and capability to process big data. In the stock market, sentiment analysis works on the assumption that short-term movements and fluctuations are largely influenced by the 'investors' feelings' and which can be seen empirically in many moments, most prevalent in 'panic selling'. Due to algo-trading which dominates more and more the stock market there have been numerous occasions lately when a set of automated decisions suddenly selling a specific asset would cause a chain reaction in which more and more investors will continue replicating the same behaviour causing a crash for that particular asset. In terms of data, two major sources are usually chosen: financial news from respected news outlets or large amounts of posts from Twitter. It is clear that while the news are on average more professional and correct, they are also delayed to a point in which the effect has been already observed in the market or on social media.

Further, we are going to present and cite the following research paper: "Big Data: Deep Learning for financial sentiment analysis" [15]. Instead of data mining for the most relevant features, they have decided to use a deep learning model that will learn of its own the relevancy of each information. As data, they have received permissions from a specialized company StockTwits Inc. to access their dataset which contain historical prices, buying or selling recommendations, but also access to a specialized network of people with investment knowledge. They have found difficulties in processing the data and filtering messages from top investors versus average ones because many messages were not labeled entirely. They have compared results from several architectures containing recurrent neural networks (LSTM) and convolutional neural networks (CNN). In addition, they have also used word2vec - a deep learning technique which produces high-dimensional vector representation of each word or document. Surprisingly enough, CNN was by far the most performant model reaching a 90% accuracy, while the other mentioned techniques reaching accuracies close to 70% and very similar to logistic regression. A possible explanation might be that CNN are already recognized for finding features in Big Data and variations may be considered with LSTM being only a secondary level part in the network.

Hybrid methodologies have always been a predictable continuation of the classic techniques in order to obtain better results. As already seen in previous mentioned papers, while the general methodology can be seen in one category, there is already a decent chance that other techniques were incorporated more and less. Sentiment analysis will definitely become more and more incorporated along more 'unbiased' techniques like pattern matching in order to gain a balanced perspective as long as there will be public development of open and good quality data sources.

2.2.4 Proposed approach

Financial time forecasting has been the top problem of computational intelligence for both academic financial researchers and industry investors due to its broad implementation areas and impact. As mentioned previously, many techniques and methodologies have evolved throughout the recent decades alongside the exponential growth in computational power. Machine learning has become one of the headlines in computer science and more recently a sub-domain of machine learning, deep learning, has started to receive the most attention of the industry. Even though deep learning has been recognized as the next breakthrough in many problems such as speech recognition, image recognition, natural language processing and many more, in terms of time series forecasting and more specifically stock market prediction the research is still continuously developing. The current state of art is a lot harder to define since there are many different approaches to what is going to be forecast and how those results should be incorporated in a strategy that would maximize the profit and minimize the loss. Consequently, this thesis will bring a clearer comparison between similar approaches in terms of output results and strategies, but keeping a consistent approach in terms of the actual deep learning model used.

Deep learning at its core is more of an abstract concept representing a class of machine learning that uses multiple layers to progressively extract higher level features from the raw input. In terms of architectures, there are many variants which have seen success in one or more domains: deep neural networks, deep belief networks, recurrent neural networks and convolutional neural networks. For the specific domain of stock market prediction, there has not been decided a definitive architecture which should be seen as the starting stone. However, based on the fact that the problem of stock market prediction can be seen as a time series forecasting problem where the past data should influence the future trend, there is a growing trend in using variations of recurrent neural networks since they were created exhibiting temporal behaviour in mind and some sort of 'memory'.

Based on the systematic literature review [12], we can draw many conclusions from which will orientate our search. We are going to discuss about several topics: subtopics of financial time series prediction considered and categorization of techniques and network structures that were used.

Firstly, let us begin with the subtopics of financial time series which were considered in the literature review. The paper research included not only stock market dependent activities, but also other markets which have a certain connection with it such as commodity (oil, gold etc.), cryptocurrency or forex markets. Even though, there might be certain differences between them, at their core in all applications the same underlying dynamics occur. As mentioned also in this thesis before, all categories can be clustered in two main groups based on their expected outputs price prediction and trend (movement) prediction. Although, price prediction is definitely the harder problem of the two increasing the difficulty exponentially from classification to regression, in terms of financial and economics interests the actual improvements are not seen as a major importance taking into consideration also the risk management. For that main reason, many researchers consider trend prediction a more crucial study area than the actual price prediction considering the actual implications. However, from the statistics of the literature review [12] it can be observed that trend forecasting represent less than 40% of the actual research considered. Regarding the classification categories which were used two methodologies accounted for almost any study:

- two-class approach as in buy or sell which basically represent an upward or downward trend
- three-class approach which represent one of the following three patterns: decline, increase or stagnation

Other types of classifications are relatively hard to promote or justify for some simple reasons.

First, it will be hard to come up with new and justified categories which would not blur the lines between them. Secondly, new categories will imply in theory some percentage margins based on which you would decide for example whether the increase is small or large. Those percentage might also be adjusted depending on how long the time horizon is for the actual predictions. Lastly, more categories would increase the actual difficulty of the problem and can be justified only if you find along a method which would be able to reward the improvements which were attempted. In terms of regression, the main distinctions between solutions without mentioning the time horizon is whether the problem is approached as a single-point or multi-point regression. The difference stems from the actual output of the problem which can be only a single point in the future based on the decided horizon or a set of points for each day in the future until the decided horizon. While the second category is on a whole another level in terms of difficulty, it is also more justified in terms of trying to improve and solve the original classification problem. The first category would eventually be used alongside a investment strategy and a risk management literally in the same way as the classification one, while the actual set of points may offer more hindsight regarding the market future information and changing an actual investment strategy as a whole. In this thesis, we will try compare the results from multiple approaches and attempt to link some of them into an investment strategy algorithm.

Secondly, we are going to summarize the trending techniques of architectures used in financial time series problems. According to [12] recurrent neural networks dominated in terms of used architectures. The main reason for this occurrence is the actual nature of our problem which implies data across a timeline with time-dependent components. Even though, recurrent neural networks accounted for more than half of the models, we should mention that many variations of recurrent neural networks have been included in this category and classic recurrent neural networks weren't nearly as present as more popular choices (as in other domains) such as long short-term memory (LSTM) or more recently gated recurrent unit (GRU). LSTM networks have been successfully used in domains such as speech recognition, music composition or even time series prediction giving them a stronger sense of security. Besides recurrent neural networks, other techniques were also used such as deep multi layer perceptron (DMLP) due to its acceptance in the past of its 'older brother', the multi layer perceptron (MLP), convolutional neural networks (CNN) or deep reinforcement learning. Taking everything into account, this thesis will use variations composed of LSTM and GRU networks for all the previously mentioned variants of problems due to their recognized worth in another domain and their large flexibility in terms of both classification and regression problems.

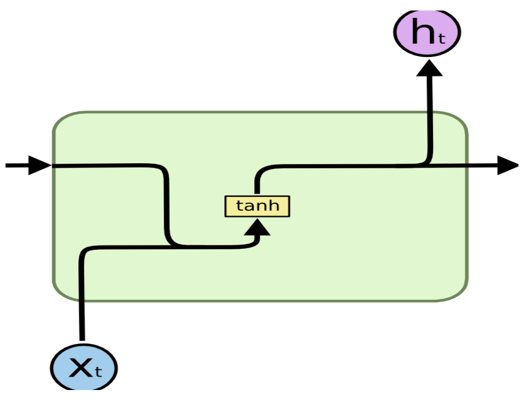
2.2.5 LSTM and GRU techniques

In the following subsection, we are going to present the details regarding the structure and characteristics of long short-term memory and gated recurrent unit layers and networks. The discussion will progressively evolve from classic recurrent neural networks to LSTM which were first proposed over two decades ago and finally to GRU which is more of a newer concept being more 'light-weight' than LSTM while maintaining most of their counter-part abilities.

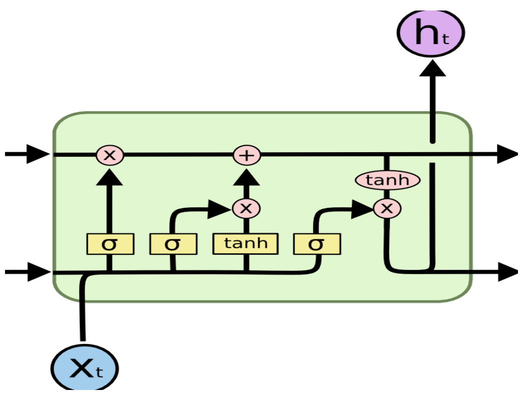
Recurrent neural networks (RNN) have been created with series in mind such that neighbouring inputs might have an influence on each other in some way. A RNN remembers a portion of its past (previous inputs) and influences the results of future inputs. In order to achieve this behaviour, RNN in addition to classic neural networks contain a hidden state which is updated and used for each input received in the series. This hidden state can be thought of as a context based on prior inputs. Moreover, this structure allows to process variable length sequences of inputs, making them applicable for unsegmented tasks such as handwriting recognition. In addition, when speaking of recurrent neural networks we can introduce the topic of 'parameter sharing' which has been successfully used in image classifying convolutional neural networks. However, a major difference should be noted in the actual definition of the neighbour between RNN (elements in a series) and CNN (neighbouring pixels in an image).

Even though, RNN were a significant improvement in the theory of machine learning, their classic versions didn't see much success initially. Practical difficulties have been reported in training recurrent neural networks to perform tasks in which the temporal contingencies present in the input/output sequences span long intervals [2]. Based on the previous citation, it was demonstrated that it exists a trade-off between efficient learning by grading descent and maintaining relevant context information for long periods of time. Practically, RNN usually suffer of the well known problem of 'vanishing gradient'. The main issue arises when training a multi-layer network while using a backpropagation and gradient-based learning algorithms for updating the network values. While the initial error is transmitted along the entire network in order to adjust the weights accordingly, there is an increased chance of getting an update which progressively becomes more and more insignificant for the actual weight of that cell causing a valid update which actually doesn't influence the network behaviour a lot. Establishing on these aspects, further research was done to search for stricter structures and architectures which would improve the previous state of the art. Long short-term memory has been a major breakthrough in finding structures that would outperform previous networks, being proposed in 1997. As a testimony, even today many applications in different domains representing state of the art are using LSTM architectures under the hood.

Long Short Term Memory networks -abbreviated as LSTM- were introduced by Hochreiter and Schmidhuber in 1997 [7] and were further refined while replacing classic RNN more and more. The main difference between them is the new structure of a singular cell. While the repeating module between different layers is mostly the same, an LSTM cell has a much more complicated structure in order to address the long-term dependecies issue. In order to compare them and explain the structure we are going to examine and compare two images with the content of each cell.



(a) Classic Recurrent Neural Network Cell



(b) Long Short Term Memory Cell

Figure 2.1: Comparison between Classic Recurrent Neural Network Cell and Long Short Term Memory Cell

Original site: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

As we may observe, the LSTM cell maintains the key points of a RNN cell -input vector, tanh to adjust the weights in the interval [-1, 1] and the hidden state- while adding a set of layers inside the cell which create certain structures which are also called gates. Next we are going to create a legend for the elements inside the figure in order to allow us to explain the structure.

- X_t - input vector
- h_t - hidden state after computation inside the cell
- x within a circle - pointwise multiplication
- + within a circle - pointwise addition
- σ - sigmoid function
- $tanh$ - hyperbolic tangent function
- yellow rectangle - neural network layer

Regarding the arrows which enter or exit the figure we have:

- first arrow entering (upper half of the figure) - previous cell state denoted C_{t-1}
- second arrow entering (lower half of the figure) - previous hidden state denoted h_{t-1}
- first arrow exiting (upper half of the figure) - new computed cell state denoted C_t
- second arrow exiting (lower half of the figure) - new computed hidden state denoted h_t and also being used as output if necessary

The main idea behind LSTMs is the cell state C_t which can be seen more of a consistent value in comparison to the hidden state. From the figure 2.1b, the cell state runs along the horizontal line in the upper part. In comparison to the hidden state which suffers many changes and operations, the cell state can be seen as rather regular. This implementation facilitates the flow of past information close to unchanged allowing to maintain long-term dependencies more further in the series. As the cell evolves through the series, information gets added or removed via gates. The gates are small neural networks themselves that decide and learn what information is relevant to keep or forget during training. As mentioned in the previous list, besides pointwise basic operators there are two classic functions applied: tanh and sigmoid. While tanh is used to regulate the network maintaining the values in a small interval [-1, 1], the sigmoid squishes the values between 0 and 1. This approach is useful in easily deciding what to do with the current information. A value close to 0 in a multiplication is equivalent to the idea of forgetting that data, while a value close to 1 represents important data which should be kept as intact as possible.

The interaction between the hidden state, the cell state and the input is dictated in a LSTM cell by some small neural networks called gates. There are three types of gates which we'll further explain: forget gate, input gate and output gate. In order to have a visual perspective, we'll use an edited version of the previous figure for a LSTM cell.

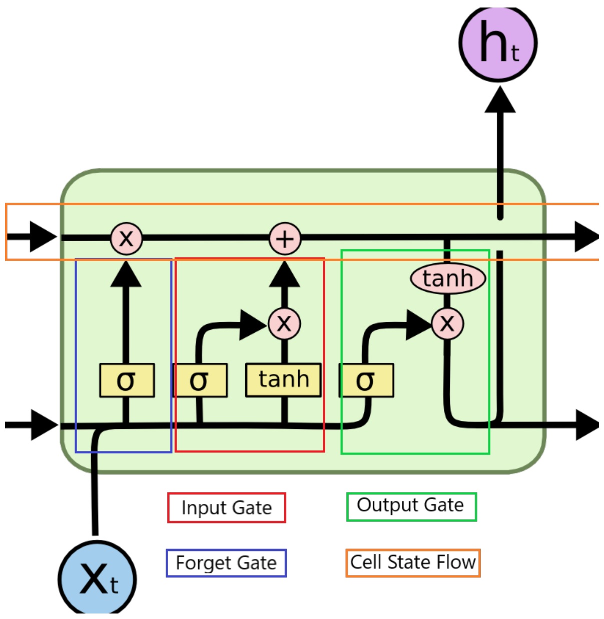


Figure 2.2: Edited Long Short Term Memory Cell With Regions

Original picture (edited after): https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Forget Gate

First gate in the flow of a LSTM cell is the forget gate. This layer decides what information is kept and what information is thrown away. As input information for this layer, we use the input vector x_t and the previous hidden state h_{t-1}. That information is passed through the sigmoid function which as discussed manages with numbers between 0 and 1 what to forger or remember. The output of this layer is a vector which is used for the previous cell state C_{t-1}, consequently having a length equal to that state. If we denote by f_t the output of the forget gate, the following equation may be seen as the behaviour of this layer:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Input Gate

Second gate in the flow of a LSTM cell is the input gate. The purpose of this layer is to update the content of the cell state based on the new input received. As opposed to the previous layer, this structural gate has a more complicated flow because it must maintain a consistent regulation of the values in the network. The same input as from before, x_t and h_{t-1} is passed on each input branch function: σ or tanh. The sigmoid branch is dealing as before of filtering the useful information to be stored later denoted i_t, while the tanh branch helps in regulating the network by creating a vector of new candidate values denoted $\tilde{C}_t$. Combining those two together will obtain the output vector of this gate which will dictate modifications for the new cell state. Let us observe the equations for each branch, while the output vector will be denoted further just by the product of those two:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Cell State Updates

Before moving on to the last gate of a LSTM cell, there must be a discussion first about the update of the cell state which has already received two vectors from the previous gates. The cell state flow begins with the previous values denoted C_{t-1} which are pointwise multiplied with f_t, the output vector of the forget gate. The result of this first set of operations is further advanced in the flow in order to incorporate the new result of the next set of layers, the input gate output vector denoted $i_t * \tilde{C}_t$. In terms of mathematical operations, the output vector is simple pointwise added to the previous result obtained in the flow. After this step, the value obtained is the final cell state for this iteration and will be lastly used in the computations of the new hidden state. The mathematical equation of computing the cell state with mentioned notations is:

$$C_t = C_{t-1} * f_t + i_t * \tilde{C}_t$$

Output Gate

The last set of operations in a LSTM cell are represented by the output gate. Its purpose is to decide the content of the next hidden state based on all previous information and operations. From figure 2.2, we can observe that we are going to unify the previous hidden state, the input vector and the newly computed cell state. The sigmoid layer as before decides which parts are going to be relevant, in this case which parts are going to be relevant to output. The sigmoid result will be denoted as o_t to represent the output vector. The newly computed cell state is passed through a tanh function which regulates the values for the new hidden state. The results are multiplied pointwise obtaining the new hidden state which is the output for our gate and is also represented as the output in prediction problems. The equivalent mathematical operations are:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

In order to offer a different perspective for the whole theory behind a LSTM cell, a pseudo-Python code will be attached to represent all discussed operations.

Algorithm 1 Pseudo-Python code for the flow of one LSTM cell

```
def lstm_cell(prev_ct, prev_ht, input_vector):
    combined = prev_ht + input_vector
    ft = forget_layer(combine)
    candidate = candidate_layer(combine)
    it = input_layer(combine)

    ct = prev_ct * ft + candidate * it
    ot = output_layer(combine)
    ht = ot * tanh(ct)
    return ht, ct


for input in inputs:
    ct, ht = lstm_cell(ct, ht, input)
```

Finally, we are going to talk about Gated Recurrent Unit (GRU) neural network layer. It was introduced recently, in 2014. GRU has many similarities with LSTM as the existence of a forget gate, but has fewer parameters allowing the overall network structure to become more ’lightweight’ and reducing training times. In terms of performance, there have been different studies reigning in favor of one or another, but in most types of problems they perform similarly in terms of accuracy and the choice is at the latitude of the developer. As with the LSTM, we will attach a picture regarding the structure of one cell and comparing it.

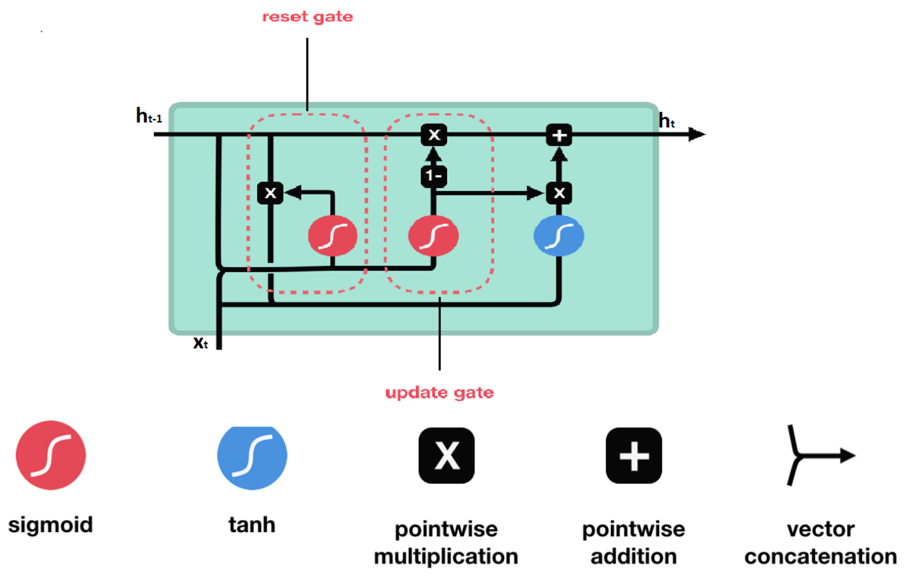


Figure 2.3: Gated Recurrent Unit Cell

Original picture (edited after): https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

As before we can observe the same main operators: sigmoid function, hyperbolic tangent function, pointwise addition or multiplication. However, we can clearly see a reduction in the number of gates and consequently operations.

Reset Gate

The reset gate decides the amount of past information to forget by using both the previous hidden state of the cell h_{t-1} and the current input vector x_t. While at the first glance it may seem similar to the forget gate from an LSTM, in reality they serve different purposes. We are going to denote with r_t the result of this gate operation which we'll be used further.

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

Update Gate

The update gate represents a clear distinction from LSTM by creating from two different gates, forget and input, a single gate in GRU. It decides what information is added and what information is thrown away respectively. We'll denote by z_t the first result of the sigmoid in the update gate which is passed in two separate sets of operations which will finally construct the new hidden state h_t.

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

GRU doesn't have anymore an output gate since the cell state and the hidden state are merged at the end. On the last set of operations will denote with $\tilde{h}_t$ the result representing the current memory content which in the end is linked with the results from the update gate to compute the final hidden state h_t.

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Finally, before moving to the next subsection we are going to address two studies regarding the general performance and structure of the LSTM and its counterparts. While the discovery of the LSTM structure may seem at first to some small degree random, since then many studies were done in order to find an improve architecture which is able to constantly outperform the classic one.

Firstly, we are going to cite "An Empirical Exploration of Recurrent Network Architectures" [8] which is a study conducted in 2015 for searching various architectures starting from LSTM. The methodology was empirical consisting in algorithm very similar to Genetic Programming. A set of best architectures is at each step maintained and mutations are conducted in order to search for new solutions. Each architecture was filtered initially with a classic memorization problem and after a set of more 'real-life' simulation problems were conducted to address an accuracy. They have gone through over ten thousands architectures finding that no architecture consistently beat LSTM and GRU on the whole set of problems considered. An important remark found is that the LSTM network must contain a bias of 1 to the forget gate in order to be able to maintain a high performance (mentioned in the initial LSTM paper, but forgotten often).

The other citation used is "LSTM: A Search Space Odyssey" [6] which tried to tackle the same problem. However, they only considered nine architectures and evaluated the influence of hyperparameters across a set of problems. In terms of hyperparameters, the learning rate and hidden layer size influence the performance the most, while other hyperparameters such as input noise were found to diminish the results for most architectures. In the end, they concluded that some elements of the LSTM can be eliminated or merge in order to gain training time improvements, but the overall performance of the architectures is very similar across the tested problems with fine-tuned parameters.

2.2.6 Results

We are going to present initially some comparisons which will then drive our further possible improvements. We will evaluate the problem from a single point prediction in the future and binary classification. Some of the variations used will be:

- past data interval - 30 or 180 days
- Used data:
  - only company OCHLV data
  - company and three major stock indexes OCHLV data

  - company and three major stock indexes OCHLV data and a set of technical indicators for the company

- Data preprocessing:
  - Data normalization - MinMaxScaler on each column
  - Data standardization - $(x - mean(x))/std(x)$

The sample data period is 1990-2005 on JPMorgan Chase company assets and training-validation is split in 80-20 manner. The future horizon interval for prediction will be set to 5 days. The following network architectures will be used as a starting point and we'll work with the better version of the two:

Algorithm 2 Network Architecture I

```
multi_step_model = tf.keras.models.Sequential()
multi_step_model.add(tf.keras.layers.LSTM(128,
                                input_shape=x_train.shape[-2:],
                                dropout=0.2,
                                return_sequences=True))
multi_step_model.add(tf.keras.layers.LSTM(64,
                                dropout=0.3))
multi_step_model.add(tf.keras.layers.Dense(32))
```

Algorithm 3 Network Architecture II

```
multi_step_model = tf.keras.models.Sequential()
multi_step_model.add(tf.keras.layers.LSTM(128,
                            input_shape=x_train.shape[-2:]))
multi_step_model.add(tf.keras.layers.Dense(256,
                            activation='relu'))
multi_step_model.add(tf.keras.layers.Dropout(0.3))
multi_step_model.add(tf.keras.layers.Dense(128,
                            activation='relu'))
multi_step_model.add(tf.keras.layers.Dropout(0.3))
multi_step_model.add(tf.keras.layers.Dense(64,
                            activation='relu'))
```

Based on the experimental results, we have found out that:

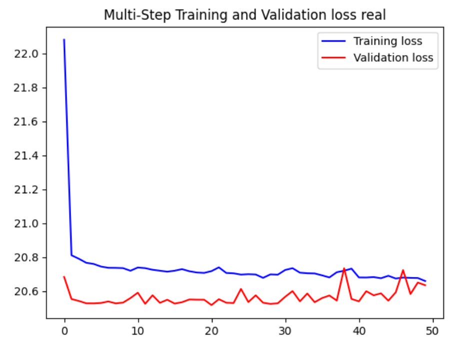
- MinMax normalization reduces the loss more significantly than standardization when evaluating the single point prediction version (2.4)
- The addition of stock market indexes decreases the overall performance from just using the company data
- Architecture I seems to slightly outperform Architecture II while benefiting from technical indicators data
- Training on the classification matter doesn't improve by a respectable margin the results on single point prediction results converted to the same accuracy.

In the following table we will present the results of the single point prediction + converted accuracy for architecture I on 5 day-future prediction with MinMax normalization:

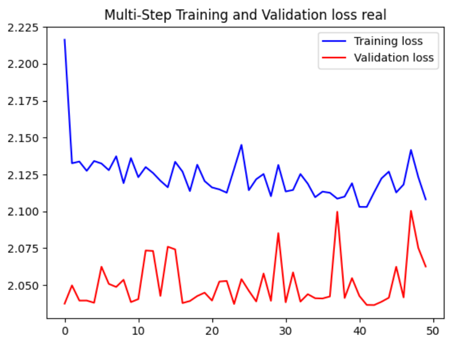
Architecture I 180 past data days

Metric	Company_Data	Company_WithIndex	Company_WithIndex_AndTechnical
TrainingLoss	0.0016	0.0016	0.0009
ValLoss	0.0003	0.0005	0.0004
TrainingAccuracy	54.15%	55.69%	56.21%
ValidationAccuracy	52.09%	46.62%	53.05%

The following figure is the comparison between the different preprocessing methods mentioned with the converted loss:



(a) Company Data Standardization



(b) Company Data Normalization

Figure 2.4: Comparison between Standardization and Normalization Data Preprocessing effects on the losses

Lastly, here is the evolution of the predicted price versus the real one overtime in the current stage:

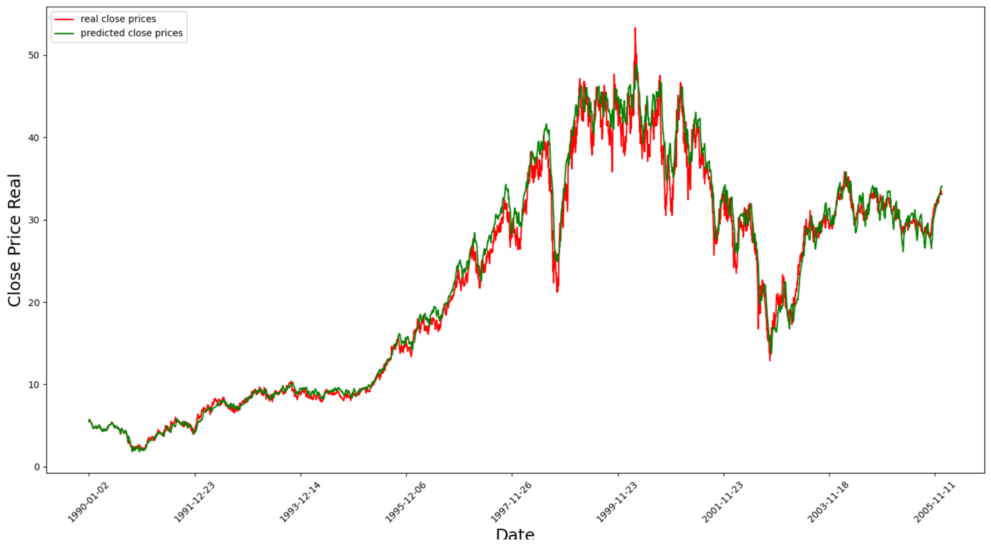


Figure 2.5: Company OverTime Prediction First Experimental Results

Based on the results obtained until now, we are going to experiment further with architectures similar with 2, MinMax normalization, specific use of technical indicators and various future time horizons and past data intervals.

Chapter 3

Technologies

In the next chapter, this thesis will present the main technologies used. The list of technologies follows the natural order of layers beginning with what have been utilized to construct and train the machine learning models and ending with the technology used for creating the client's application.

3.1 Tensorflow

Tensorflow is a free and open-source software library created originally developed by researchers and engineers working on the Google Brain team within Google's Machine Intelligence Research organization to conduct machine learning and deep neural networks research. It steadily evolved and become one if not the most popular framework to develop machine learning.

Tensorflow is a complex framework containing many layers of abstraction and components to link between different stages of the training and production. It works with other libraries in collaboration in order to facilitate an easier workflow. For our specific problem regarding stock data, Pandas and NumPy will be used along to prepare the data stored in CSV files for the actual model. Lastly, Matplotlib will be used along for generating the initial graphs and plots necessary for comparing different models and methodologies.

NumPy is a library for the Python programming language designed for adding suport to large, multi-dimensional arrays along a large collection of high-level mathematical functions to operate on these arrays. While limiting some of the classic functionalities with array, the library is a staple when it comes to efficient numerical computing in Python.

Pandas is another Python library specialized in data manipulation and data analysis. In particular, it helps tremendously in manipulating numeric tables and time series problems. It is highly customizable and offers seamless integration with CSV files. Its main concept is called DataFrame and represent a 2-dimensional labeled data structure with columns of potentially different types. It provides a large variety of functions built-in such as mean, maximum, minimum or standard deviation, slicing requiring few lines of code to create or normalize data from a source.

Tensorflow Structure

Under the hood, Tensorflow works based on the concept of the data flow graph as its computation model. A data flow graph is a model of a program with no conditionals. In a high-level programming language, a code segment with no conditionals—more precisely, with only one entry and exit point—is known as a basic block[10]. Its main usage is the ease offered in terms of distributing computation across CPUs and GPUs.

The nodes of the graph are represented by operations, while the edges are tensors. The main Tensorflow entities are: graph, operation, tensor and session. In recent updates, the session is not observed in the foreground, but the other components remain present. As previously mentioned, the graph is build as a set of Operation objects. Each Operation represents a graph node which stems for a unit of computation (addition, multiplication etc.). It takes a tensor as an input and produces a tensor as an output. The tensor is a generalization of vectors or matrices of higher dimensions. They do not represent or hold any particular value produced by an operation in the initial graph. They are used to define the type of value and dimensions acting like a placeholder until the actual execution.

Keras

Keras is an open-source neural-network library written in Python. It was designed as a high-level interface allowing for fast experimentation of deep neural networks. Tensorflow is supporting its own implementation of Keras fully integrated with its other components.

At its core, Keras offers numerous implementations for commonly used neural-networks building blocks such as layers, objectives, activating functions, loss functions. In addition to standard neural networks, it has support for convolutional and recurrent neural networks. Lastly, utility layers such as dropout, batch normalization or pooling can be easily added.

3.2 Spring Framework

Spring Framework represents an application framework and inversion of control container for the Java platform. While the core features of the framework can be used in any type of Java application, with the extensions there has been a large demand for web-type applications. Since version 5.0, Spring offers official support for the Kotlin language which aims to eliminate the issues of its much known predecessor Java while adding improved features at a much faster pace.

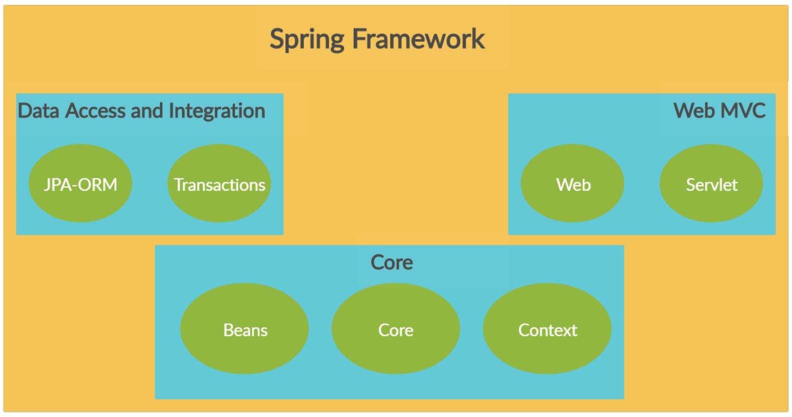


Figure 3.1: Spring Framework Components

Created with: https://creately.com/

Spring Inversion of Control

The spring framework several modules from which the core is definitely one of the most important ones. It offers the implementation of Inversion of Control (IoC) principle and is also known as Dependency Injection (DI). It is a process through which the developer only defines the necessary dependencies between objects through constructor arguments, arguments to a factory method or properties. From that moment, the core container will deal with the actual underlying issues of creating and injecting the instances necessary. As opposed to the general methodology in which the class instance controls the instantiation or location of its dependencies, the process is totally inverse resulting in the name of Inversion of Control.

The basis of the container is represented by two concepts: the beans and the context. A bean is an object that is instantiated, assembled and managed by a Spring IoC container. Besides that, the bean behaves just like the implied behaviour from the developer's implementation. The BeanFactory provides the configuration framework and basic functionality, while the ApplicationContext improves upon the aforementioned class and adds a new variety of operations. Consequently, the ApplicationContext represents the main Spring IoC container and is responsible for the aforementioned tasks. In order to grasp the necessary instructions, configuration metadata is searched and create based on different elements annotated by the developer in its code. The older standard was represented by XML which was steadily replaced by Java annotations and Java code.

In order to facilitate the linking between necessary dependencies, autowiring was introduced as an automated method through which the framework resolve what type of bean to be used and where. To obtain such a behaviour, specific should be given to naming conventions and bean classes such that there would be no conflicts in which two or more beans might be used for the same construction argument etc. The introduction of bean scopes allows to differentiate between different number of instantiations needed for a specific class. The singleton design pattern can be very easily achieved through dependency injection requiring no further boilerplate code prone to errors.

Data Access and Integration

Spring framework offers an extensive data access module allowing improved usage when communicating with your persistence choice. At its core, there is a transaction management system which provides an abstraction layer such that there will be a consistent programming model despite using different technologies such as Java Transaction Api, Java Database Connectivity or Java Persistence Api. Java Database Connectivity (JDBC) is the barebone application programming interface created by Oracle to define a universal communication standard between the application and various drivers for different database providers. While continuously improved, JDBC has more lately been considered a choice for maintaining only certain types of applications and more modern APIs should be used instead in new applications.

Java Persistence API (JPA) represents a higher level abstraction application programming interface which allows a future key aspect: object-relational mapping. Object-relational mapping allows data conversion between incompatible systems using object-oriented programming languages. In other words, it allows conversion between your domain classes and associated database tables. Spring offers JPA integration through Hibernate which is by far the most object-relational mapping tool for the Java programming language.

JPA offers idiomatic persistence, high performance and reliability. Idiomatic persistence enables the developer to write persistence classes using object orientated classes and linking them to the database structure through various annotations. Its performance stems from the available fetching techniques that can be used in order to load only the relevant data. At its core, JPA creates a small duplicate cached in memory database on which operations are made when executing some commands. Only when a transaction is committed, the changes will be ported over to the real database and will come into effect. In terms of fetching, there are two main fetching strategies: eager and lazy. Eager refers to fetching the entire data in one transaction, while lazy fetching will load some of the data only when needed in some operation. While lazy fetching is preferred, more attention should be made such that you won't access data outside a transaction environment.

In order to promote an intuitive platform for communicating with the layer, Java Persistence Query Language (JPQL) was introduced to break the barrier between SQL and the object-oriented model. JPQL offers a syntax very similar to SQL, but changes the focus on retrieving entities instead of rows or fields. JPA offers extended ORM support with advanced mappings and entity relationships. Advanced mappings offer inheritance support for objects in three different manners: single table, joined table or table per concrete class. In terms of relations, the platform offers the same relations as in the relational database design: one-to-one, one-to-many and many-to-many.

Spring Web Model-View-Controller

Spring Web MVC framework is the original web framework built on the Servlet API and has been provided since the beginning. It is designed around the concept of a DispatcherServlet which represents the front central controller and provides a shared algorithm for generic request processing while delegating future work to other components. Further, we are going to present only the main components that will be used along for our needs.

Starting with Spring 3.0, the framework allows the usage of the representational state transfer architectural style (REST) in order to create Web services. The framework allows painless declaration of various endpoints based on the HTTP protocol. As a standard transfer format in request, JavaScript Object Notation (JSON) is used and automated binding between JSON and POJO classes is done behind the scenes by some delegate of the DispatcherServlet.

A RESTful system favours a client-server architecture allowing improved performance, scalability or portability. A key concept of REST architectural style is the use of stateless requests by default, but a more detailed explanation will be presented in the next chapter regarding how to deal with statelessness, but maintaining information about what user is requesting the information. In order to filter and log the useful information for each request, the framework a base class GenericFilterBean which allows the custom behaviour to be set for all requests with a specific URI pattern.

3.3 Android Software Development Kit

Android Basics

Android is the most popular mobile operating system and one of the two major players in the market -alongside iOS. It is based on a modified Linux Kernel and other open source platforms, designed mainly for mobile devices such as smartphones, tablets and Google's version is the one used by almost everyone.

In order to facilitate the software development of third-party apps, Google created and published the Android Software Development Kit (SDK) which allows developers to integrate code mainly at the most upper-level of the system. Under the application layer, there exists a specific hierarchy on top of the Linux kernel, but they aren't concerning the application developers.

Different from many operating systems, an application is Android is almost totally isolated from the rest. Basically, each process in Android has its own Virtual Machine (VM) and each app has its own unique process creating a security sandbox environment. The Android system implements the principle of 'least privilege' which means that each app has access only to the minimal amount of components to work and nothing more. Consequently, each application contains a manifest file which specifies the features, permissions and hardware components which will be used by the application.

App components represent the building block of any Android application. Each component is considered an entry point to your point, the one accessing being either the user or the system itself. There are four major existing types: activities, services, broadcast receivers and content providers. While each of them has its own lifecycle and serves a different purpose, activities are widely considered the staple in terms of building blocks. An interesting design aspect is the communication between components. While apps are isolated as mentioned earlier, each component of an application can access an external component of another application through a asynchronous message called 'Intent'. This message is maintained and passed along by the Android System itself in order to respect the previously mentioned isolation points. For this main reason, an Android application hasn't only one entry point as we were used from other systems changing the standard paradigm structure.

Activity Lifecycle

The activity component is the main process with which you facilitate an independent point of entry in your application. It also provides the layout which will be drawn on the screen by the operating system and usually represents the main interaction point with the user. Each created activity must be added to the manifest file which will contain other options available to extend your activity functionalities.

A core topic in Android is the lifecycle of a specific component, more importantly the lifecycle of an activity. The lifecycle of an activity can be seen as the flow of states through which an activity passes when it is created, paused, destroyed by the system etc. From the development perspective, the Activity class offers a series of callbacks which are called by the system itself when some event happens changing the state of the application. In order to easily present the different states and changes, we are going to use an official simplified schematics from the guidelines.

As we may observe in figure 3.2, the activity android lifecycle contains a relatively complicate graph with no more than six core callbacks to which some optional ones can be used in special cases. For an easier presentation, we are going to present them in groups by two similar to a constructor-destructor manner.

Firstly, $onCreate()$ and $onDestroy()$ represent the most 'outer' callbacks called when an activity is created from scratch or destroyed. The create is responsible for basic startup logic usually done once per the lifecycle of the activity such as inflating the initial view or settings listeners for user interactions. Lastly, the method received a parameter called 'bundle' which is maintained by the system to be able to reproduce an activity destroyed while in the background. The $onDestroy()$ method is much less used because many resources are usually released in previous callbacks such as *onPause*() and *onStop*(). It is called when an activity is finished or a configuration change occur requiring the system to redraw the entire UI. As a rule of thumb, the resources remaining open and becoming unnecessary should be freed in this callback.

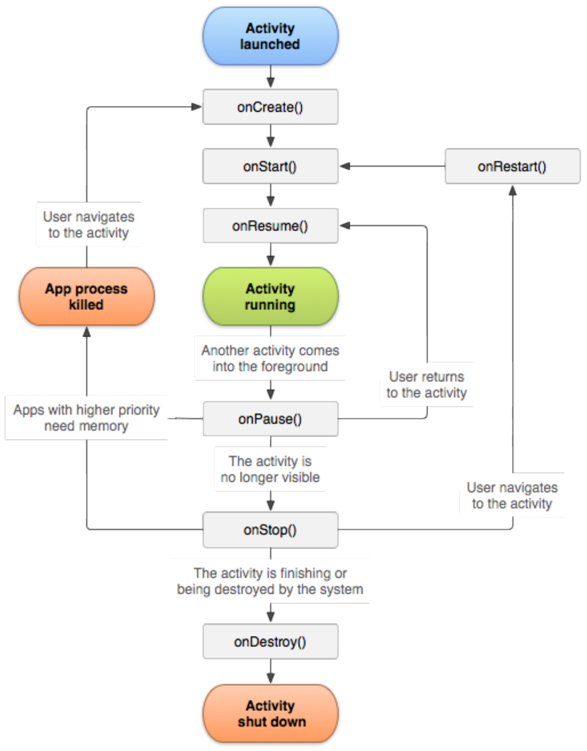

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Figure 3.2: Android Activity Lifecycle Flow

Original picture: https://developer.android.com/guide/components/activities/activity-lifecycle

Secondly, *onStart*() and *onStop*() represent the callbacks defined at the moment that the activity becomes visible or invisible to the user. The start method is responsible with assuring that any user interface element is totally linked between the activity and user interaction allowing the *onCreate*() to reduce in general size which has become in many large applications unacceptable to some extent. On the other hand, *onStop*() is considered more relevant in an importance ranking system because in general an activity should change its intensive behaviour when it is no longer the main focus for the user. It is called when a new activity is launched and will cover over the entire screen or preliminary to *onDestroy*() when an activity is to be destroyed. The method is responsible to stop or reduce actions which are no longer necessary such as animations, intensive sensor or location tracking and is recommended as the place to use CPU-intensive operations shutdown operations such as database calls.

Lastly, *onResume*() and *onPause*() define the interval time frame in which the user can successfully interact with the activity. In general, the *onResume*() method is only responsible to assure that the elements which may have been disabled in an *onPause*() callback are fully operating again since we can observe from 3.2 that there exists the possibility of a small cycle

$onResume() -> onPause() -> onResume()$ without going through other lifecycle callbacks. The $onPause()$ method is responsible for releasing resources which aren't necessary while the application is not in the foreground. However, there is an exception for this rule: multi-window mode where your activity might not be in focus, but still visible for the user and still processing (eg. music or video player).

Android Architecture Components

As we have seen, the Android platform has additional problems to be solved in order to assess the quality of a particular application. As we have presented the activity lifecycle while not mentioning fragments which represent sub-behaviours and sub-interfaces with their separate lifespan, there appeared a difficulty in deciding a suitable architecture for a generic application. Due to these factors, Google has decided to introduce a separate package from the core Android library called Android Jetpack in order to help developers write more high-quality and structured application in an easier manner. In the following paragraphs, we are going to present the 'Android Architecture Components' - a collection of libraries specialized for designing robust and maintainable application with a clean architecture.

The components methodology stem from the standardized structural design pattern decided for Android : Model-View-ViewModel (MVVM). In this manner, there is a clear separation between the data persistance layer (Model) and the user interface layer (View). The ViewModel acts as an intermediary layer between the two and favours event-driven programming exposing Observable objects to the View layer which is responsible to decide how to interpret the new consumed value. In order to achieve this minimum behaviour, three main components were added to facilitate it: Room Persistence Library, ViewModel and LiveData.

Room Persistence Library represents a huge improvement over the default SQLite database providing a supplementary abstraction layer. Even though initially you might get the impression that room allows for basic object-relational mapping, in reality it is strongly discourage due to the limitations and the speed necessary for displaying each frame of an application. It consists of three main set of objects: a database class with annotations for the entities objects, the entities classes which represent each a table in the database and a set of data-access-objects (DAOs) which contain interface with queries specific for the necessary use case.

LiveData is an observable data holder class. The improvement over a regular observable is the lifecycle awareness integrated in the classs, allowing for ease of use without implementing multiple callbacks to make sure that the lifecycle is respected whether we are talking about an activity, a fragment or a service. In addition to this, LiveData is integrate with Room and has support for Kotlin coroutines allowing for one-way or two-way data binding.

ViewModel class was designed to store and manage UI-related data while maintaining a life-cycle awareness. It was created in order to provide data for the UI even after a configuration change which causes the recreation of the entire user interface. A ViewModel can outlive the activity to which is linked and represents a much improved version of keeping data between changes as opposed to the initial standard bundle.

In addition to all these components, Navigation component and view binding will be incorporated to facilitate moving between different screens. View binding is responsible to create a safe environment for the specific layout used in the screen by compile time checking, while the navigation component allows for the creation and automatization of multiple navigation graphs which will present a clear flow in what actions can the user take to navigate through the application.

Chapter 4

Application Development

"Application Development" chapter will focus on presenting the taken steps during the software development stage. The flow will be progressive beginning with the necessary requirements and specifications from a stakeholder perspective. Further, after a clear use-case template will be created we will move to the next and most critical stage: the design. The design phase will expose the initial high-level architecture decided in order to facilitate different aspects such as reliability or performance. Following the initial high-level architecture, a more fine-grained and detailed presentation will occur regarding the elements and components from the application. A testing methodology will be added and decided to ensure a quality control. Finally, a linking between design and implementation will focus on the main functionalities of the application which expose a more complicated user-system interaction.

4.1 Application Planning and Requirements

4.1.1 Requirements and Specifications

In the next subsection, this thesis will present a brief discussion regarding the possible stakeholder requirements alongside a more detailed specification list from the development perspective.

The act of stock market prediction is usually incorporated in large financial applications which may offer a variety of functionalities. However, in those applications the core functionalities are considered of a higher importance and relevance than the actual reliability of the prediction. This application will try to incorporate some functionalities from generic financial applications, while maintaining its core to the actual prediction and useful information along in order to facilitate decisions for the users (investors).

The application has to be easily accessible and offer the users a friendly and familiar environment. The user should be able to select a list of favorite companies to follow. For each company, he should be able to check details about the company, check the historical price and check prediction on both historical and future data. Moreover, the user should be linked with some recent financial news which might affect the market or to run a simulation on a given interval of time on historical data with a fixed investment strategy. Lastly, in order to encourage user engagement and provide a good experience, a feedback platform is mandatory for improving further the development.

From the development perspective we have decided a list of specifications which should be fulfilled by the application. The application will have a sign-up and login system, using two types of users: client and admin with different permissions. Further, we'll split the specification list in two based on the role.

Client User

- Register and login as an user
- List the available companies to follow
- List some recent news which might affect the future trend of the market
- Ability to select and add favorite companies
- Remove a company from your favorite list
- Request various details about a favorite company
- Request the historical data for a favorite company as a plot
- Request prediction for historical events to check the stability
- Request future prediction for the present moment for different time-horizons
- Run an investment simulation based on a set of companies and a fixed strategy
- Feedback report system to the developers
- Notification system regarding feedback answer

Admin User

- Login as an admin
- Create a new entity company with its details
- Update company details
- Update company networks links to make it available for the clients
- Remove a certain company from available companies for clients

- Respond to user feedback
- Request the system to update financial data available with new request from external API

4.1.2 Abstract Design - Use Case Diagram

In the following subsection we are going to present the associated Use Case Diagram with the previously specified requirements specification. Two actors have been found along: the client and the admin. There are some use cases which belong to both, while most of them are separate.

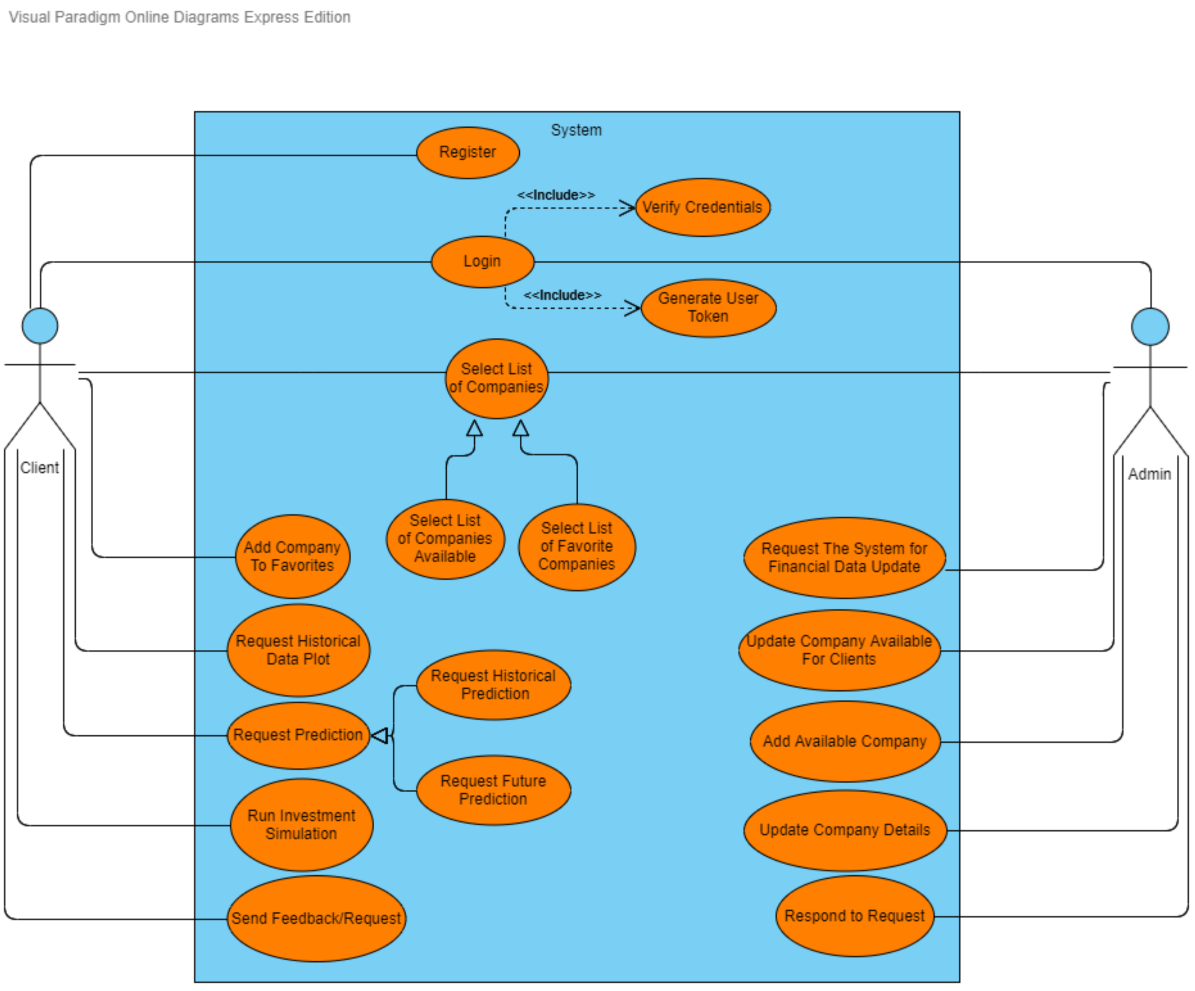

Visual Paradigm Online Diagrams Express Edition
System
Register
<<Include>>
Verify Credentials
Login
<<Include>>
Generate User Token
Select List of Companies
Client
Admin
Add Company To Favorites
Select List of Companies Available
Select List of Favorite Companies
Request The System for Financial Data Update
Request Historical Data Plot
Update Company Available For Clients
Request Historical Prediction
Request Prediction
Add Available Company
Request Future Prediction
Run Investment Simulation
Update Company Details
Send Feedback/Request
Respond to Request

Figure 4.1: Use Case Diagram

4.2 Application Design Phase

4.2.1 Application Architecture

In order to facilitate accessibility for the application, the decided architecture is of type client-server where the client application will be a mobile application. This architecture was chosen with multiple reasons in mind while being known for its flexibility and allowing easily addition of different types of clients if necessary.

The need for a dedicated server (of sets of servers) stems firstly from the amount of financial data used for predictions. Bundling large amounts of data with a portable mobile application is unacceptable. Moreover, this data needs frequent updates from known sources which have free API-s available, but with some low limits of requests per day. In addition, a large amount of neural networks will be maintained for different goals, time-horizons and companies. While nowadays mobile phones are becoming more and more powerful being capable of running some machine learning tasks, still most of the work is done in the cloud on dedicated servers. Lastly, moving the computational cost from the client to a server allows the application to be run on a larger variety of devices which otherwise might have been insufficient.

In order to facilitate possible integration in modern cloud systems, we are going to split the server computation cost in two major parts. The first part which will directly communicate with the client application will contain the entire business logic necessary to address most features which do not require any sort of prediction. The second part will behave similar to a Cloud API service for machine-learning operations regarding stock market prediction. It will be standalone and totally separated from the first part, maintaining only the network architectures already trained and expose a REST API for the business logic server to request any computations regarding prediction. The following picture attached contains a visual description of what has been discussed.

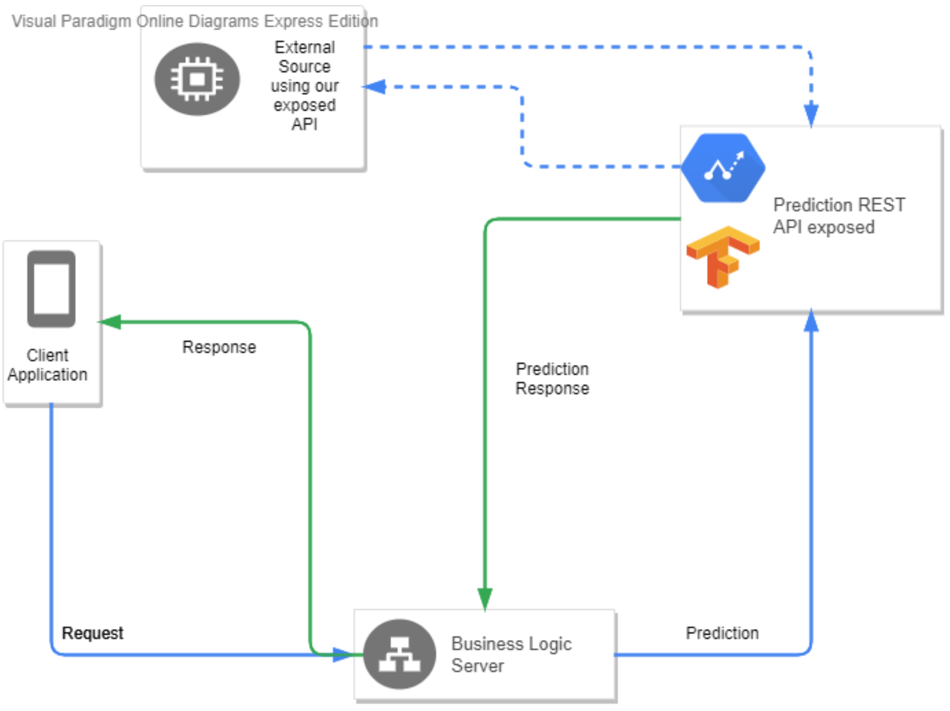

Visual Paradigm Online Diagrams Express Edition
External Source using our exposed API
Prediction REST API exposed
Client Application
Response
Prediction Response
Request
Business Logic Server
Prediction

Figure 4.2: High Level Architecture of the Application

4.2.2 Exposed Prediction REST API

4.2.3 Business Logic Server

As mentioned in the summary of application architecture, the main business logic will be maintained on a separate server from the prediction API, offering more flexibility and reliability. As mentioned in the previous chapter, we are going to use the Spring Framework which offers a large variety of services such as inversion of control, Java Persistence API or testing.

We are going to work with a structured hierarchic layered structure borrowing elements from specific design patterns and strategies. In the past, the classic used for paradigm for creating a web/desktop application was a fully fledged Model-View-Controller (MVC) structure favoring a separation between layouts and components. However, since we are creating a separately client mobile app with its specific architecture, it is clear that the server will become a RESTful Web Services provider replacing the View layer with unique stateless HTTP requests between applications.

In order to maintain a degree of modularity we are going to obey the design principle of Separation of Concerns (SoC). This principle encourages the separation of different sections in an application such that each one is addressing a separate concern. While not ideally possible in the context of many applications, it offers some valuable guidelines in structuring to some degree the generic architecture of a program.

We will begin by splitting the business logic server design in two major blocks: core and web. Core will deal with most of the business logic required containing the persistence layer, the domain or the validation schema. On the other hand, the web block will deal with the exposed REST API maintaining different path patterns for different functionalities dependant on the client's role and mappings from and to the data transfer objects (DTOs) used. In terms of content for request and responses, we are going to adopt JSON (JavaScript Object Notation) as the standard data-interchange format opposed to older version such as XML.

Further, we are going to split the main blocks into separate layers to further address each concern. Each separate layer will translate in the implementation phase into at least one package separating the code for an improved visibility. The order will begin from the most external point with respect to receiving an HTTP Request and going down the hierarchy until reaching the persistence layer.

- HTTP Filters Layer - before reaching the exposed REST API also known as the controller layer, each HTTP request should be filtered in order to assess its validity (URL, headers, content format). While the Spring Framework will deal with most of it automatically, we are still going to add some custom ones in order to address the security-login methodology or logging. More details will be found when talking specifically about the login functionality in detail
- Controller Layer - the main layer containing the mappings for request. This layer will com-

municate inside its web block with the others dependencies, while passing and retrieving data to the uppermost layer from the core block

- Data Transfer Objects Layer - it will cover the basic objects structures which are passed alongside the HTTP calls. It is used successfully in combination with the framework which is able to automatically convert back and forth JSON content and simple classes
- Mapping Layer - the mapping layer will include the converting logic between data transfer objects and the objects returned back by the core block separating this concern from the main controller layer above
- Service Layer - the uppermost and first layer of the core block. Its objective is to link the web block to the persistence and domain layers containing most of the primary business logic required using in addition a separate validation cover.
- Validation Layer - a relatively simple part of the core block responsible for validating the various inputs received from the client through the HTTP request absolving the service from this concern
- Domain Layer - contains the entities associated with the persistence layer representing the enterprise business model. In order to facilitate the Object Relational Mapping offered by the Spring Framework (via Hibernate), they include relevant relational mappings related to the database architecture
- Persistence Layer - offers a set of interface to ease the communication with the database by making use of the Java Persistence API (JPA) which is fully integrated in an extension of the Spring Framework

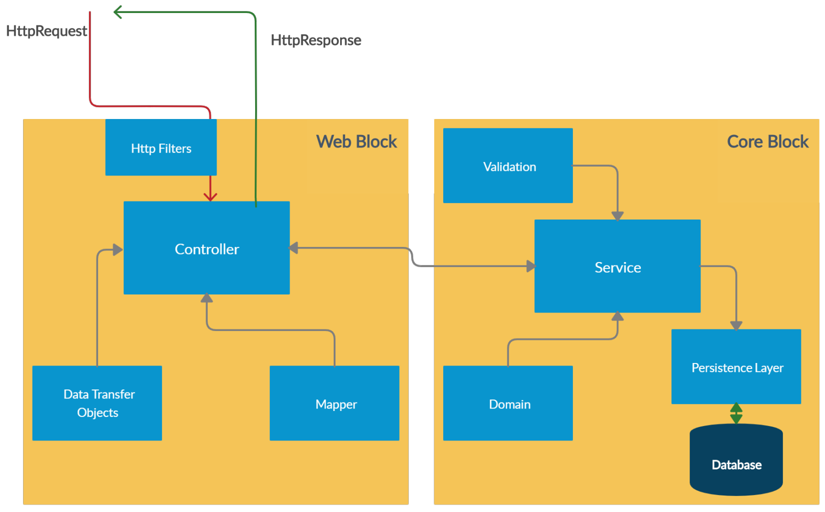


Figure 4.3: Business Server Logic Architecture

Created with: https://creately.com/

4.2.4 Client Mobile Application

As the last layer presented in the application architecture and the one directly interacting with the user, we will discuss about the necessary steps required for the mobile application. As mentioned in the previous chapter, we are going to make use of the latest facilities and paradigms offered by the Android Architecture Components library incorporated in Android Jetpack.

Based on the recommended methodologies offered along the library, we are going to choose the advocated generic design pattern in terms of structuring the app: Model-View-ViewModel (MVVM). MVVM is a software architectural pattern that facilitates the separation of the development of the graphical user interface (the view) from the business logic (the model). The ViewModel acts as an intermediary layer between being responsible for data transformations from the model and exposing possible data binding logic between the view and the viewmodel. At its core MVVM has many similarities with another software architectural pattern Model-View-Presenter (MVP), but there is one major difference. While in MVP the presenter sits at the same level with the views and there is a one-to-one mapping between the view and the presenter, in MVVM there is not such a dependency. Instead, the MVVM exposes an available data binding on which the view connects and observe creating a small code footprint in terms of updating the UI.

While being a totally supported architecture, MVVM couldn't have been easily supported in Android before Architecture Components library. The main reason stems from the lifecycle of an Android application and its UI components (activity, fragment) which are often destroyed due to configuration changes or other events. With that motivation in mind, the ViewModel class was created for storing and managing UI-related data in a lifecycle conscious way while surviving to many types of events. Moreover, the ViewModel offers a separation for executing asynchronous tasks or sharing data between fragments which will further help us in the process. In the following paragraphs, we are left to discuss the components of the View and Model.

Starting with the view, the structure of the UI 'screens' will be composed of few activities and more fragments per activity striving to the proposed 'single-activity' methodology recommended by Google. We will have a total of three activities across the entire app: Login, Client and Admin. The decision is made in order to split the responsibilities of each role discussed in the requirements and a separate logic for the login-register process. Each activity will be composed of multiple fragment, each representing a certain UI screen. In order to facilitate a clear and organised navigation back and forth while incorporating necessary arguments, we are going to make use of the Navigation Architecture component. It consists of three key components: the navigation graph, the navigation host and the navigation controller. The navigation host represents just an empty UI container which will display inside the fragment destination. The navigation graph is an XML file defining all navigation-related information in one centralized location. Includes all individual areas of content in your app also called destinations and the possible paths that a user can go through. Lastly, the navigation controller is the object responsible for managing the contents inside a navigation host and it is used to trigger routing to a different destination. Since the navigation component is designed to represent the entirety of actions inside a single activity, we will have in each activity such a component with its key parts. In terms of data updates, as mentioned previously we will work with an event driven methodology based on the observer pattern. Each view will have a reference to the necessary ViewModels and listen to updates on which the UI will react accordingly

Lastly, we are going to present the Model layer. In general an Android application communicates with two different components in the model layer: the local database and the network. In order to break the gap such that the ViewModel is not communicating directly with those providers, the repository is introduced as the middle layer. In order to reduce the number of necessary network requests every time some UI element is created, we will store in the persistence layer (the database) some of the information received from the web service. As a provider, we will use Room Persistence Library which offers an abstraction layer over SQLite and integration with LiveData - an observable data holder class. As a result we would obtain direct binding from the Persistence Layer all the way through the UI for certain exposed information with automatic updates once the database was modified. For communicating with the web service through HTTP requests, we will use Retrofit. Retrofit is an open-source library designed to ease the code required for HTTP requests reducing the necessary boilerplate code. In order to maintain the principle of single source of truth, the repository will act as an intermediary layer between the the view-model and the data model components. The standard source of data is the database which is updated with responses from the server, but for the data which will not be cached locally the repository will acquire it every time from the server and pass it to the viewmodel layer.

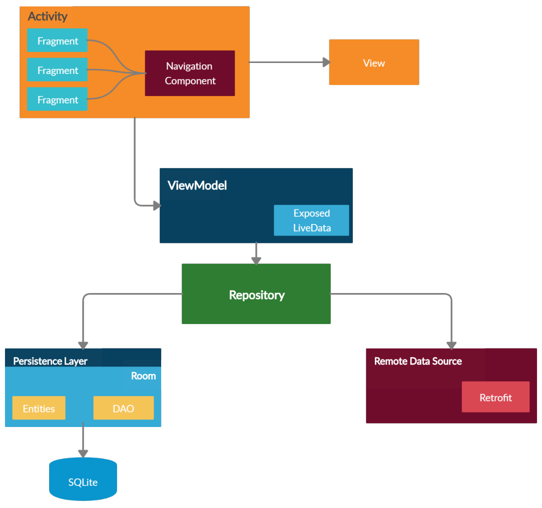


Figure 4.4: Mobile Client App Architecture

Created with: https://creately.com/

4.3 Application Core Functionalities

4.3.1 Login

The first important functionality which is going to be implemented and tested is the login. In order to offer a personalized experience for each user, an account will be associated to each individual. However, since the API exposed from the business server will be available both for client and admin tasks, further methodologies should arise to differentiate and assure the security stability of the system.

The characteristics of the REST (Representational State Transfer) architectural style will influence the most. Because a request is basically stateless in this context, the server cannot understand by itself if some consecutive requests are from the same user or not. For this reason, we will implement token-based authentication system which is being used successfully in various systems with specialized authentication servers. We will not utilize in this case the official solution offered by the framework Spring Security in order to add more overhead to our application, while maintaining an open view about what is happening behind the scenes with each request.

In a token-based authentication system, the system must hold live information about the list of connected users and their minimal useful states. For this reason, a new class will be created called Session which will maintain the useful contents for a single connection. Consequently, the session will contain the connection token, the user ID for reducing business operations, the email associated with its account, a Boolean denoting whether the account has admin rights or not and finally a creation time in order to limit the availability time for a connection. In order to maintain fast retrieval for a session based on the token, the server will maintain an in-memory map containing a token as the key and the associated session as a value. Every time a new connection -login action will be successful- a new entry will be added in the map, while a logout or an expired request will be equivalent with removing that entry. From that moment, each subsequent request will contain the token as an 'Authorization' header attached.

Implementation and Testing

On the mobile side of the application, the login implementation involves several components. First, there is an associated LoginActivity which has linked a layout file containing the specific GUI page. The layout offers two fields similar to an authentication form for the email and the password, each offering an initial hint and an error message after typing until the content respects the pattern checked by a separate validator class. Besides the input fields, there is only one button for starting the login action to the server. From the moment it is pressed, the last front-end validation is checked again for the patterns. If the validation passes, a new coroutine scope is created to handle the computational cost outside the UI thread. Inside the coroutine scope, a network service class is called in order to request the server for a new connection. If the request is was successful, the token will be stored at the application level and the main client activity will be opened. Otherwise, an error message will be displayed in the user interface.

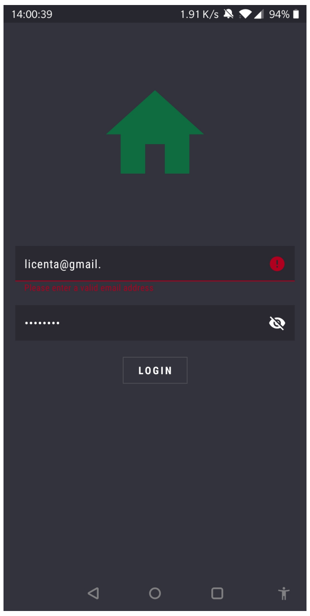


Figure 4.5: Login GUI Screenshot

On the server side, the specific login request passed initially through only one basic logging HTTP filter. It reaches the LoginController which will pass the request further along to the core block which will check whether the account combination exists or not. If the request is valid, a new session object is created with its associated token which will be passed back to the web block. Otherwise, an exception is thrown with the according error message which will be called, logged and passed back in the HTTP Response. From this moment, the client will be responsible to hold the token in a private space and pass it along as an 'Authorization' to every subsequent request. Every other request will have a specific URL pattern which will be filtered by the other necessary HTTP filters.

Each server-side HTTP filter is inheriting from GenericFilterBean which is a component offered by the Spring Framework to specify more easily the necessary behaviour. Behind the scenes the base class for filtering is the old Filter interface proposed by the Java Servlet library. Each class must override a single method called doFilter() which is receiving the request and the response as parameters and is implementing the business logic along the filter chain. In order to specify specific parameters for our filter such as the order of execution, the URL patterns and such, each class should be associated to an unique FilterRegistrationBean which will inform the framework how and when to instantiate and arrange them.

In terms of testing, we are going to test at two different layer levels while incorporating both unit and integration testing. Firstly, a set of tests will be created at the persistence layer with the help of data JPA tests from the framework. These tests will assess the quality of basic database operations such as finding the user with a specific email. Because we want to maintain our official MYSQL database in a stable state, the framework helps us by creating for each suite of tests an in-memory database (H2) which allows for fast computations and total isolation from the production database. Secondly, we will have a set of tests at the controller layer while incorporating mock objects for the core block. Consequently, we can unit test the controller layer while making sure that the HTTP request respect and follow the desired structured. Lastly, we will add integration testing by incorporating all the layers along our server, while maintaining the observation from the first set that the database used will not coincide with the production database.

Chapter 5

Final Conclusions

Bibliography

[1] Rubén Arévalo et al. “A dynamic trading rule based on filtered flag pattern recognition for stock market price forecasting”. In: *Expert Systems with Applications* 81 (2017), pp. 177–192.

[2] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[3] Chris Brooks. “Chaos in foreign exchange markets: a sceptical view”. In: *Computational economics* 11.3 (1998), pp. 265–281.

[4] Warren Buffet. *Here’s What Warren Buffet Thinks About The Efficient Market Hypothesis*. URL: `https://www.businessinsider.com/warren-buffett-on-efficient-market-hypothesis-2010-12`.

[5] Luca Di Persio and Oleksandr Honchar. “Recurrent neural networks approach to the financial forecast of Google assets”. In: *International journal of Mathematics and Computers in simulation* 11 (2017).

[6] Klaus Greff et al. “LSTM: A search space odyssey”. In: *IEEE transactions on neural networks and learning systems* 28.10 (2016), pp. 2222–2232.

[7] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[8] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. “An empirical exploration of recurrent network architectures”. In: *International conference on machine learning*. 2015, pp. 2342–2350.

[9] Sang Hyuk Kim et al. “Pattern matching trading system based on the dynamic time warping algorithm”. In: *Sustainability* 10.12 (2018), p. 4641.

[10] Anany Levitin. *Introduction to the design & analysis of algorithms*. Boston: Pearson, 2012.

[11] Murtaza Roondiwala, Harshal Patel, and Shraddha Varma. “Predicting stock prices using LSTM”. In: *International Journal of Science and Research (IJSR)* 6.4 (2017), pp. 1754–1756.

[12] Omer Berat Sezer, M Ugur Gudelek, and Ahmet Murat Ozbayoglu. “Financial time series forecasting with deep learning: A systematic literature review: 2005–2019”. In: *Applied Soft Computing* (2020), p. 106181.

[13] Dev Shah, Haruna Isah, and Farhana Zulkernine. “Stock Market Analysis: A Review and Taxonomy of Prediction Techniques”. In: *International Journal of Financial Studies* 7.2 (2019), p. 26.

[14] Weiyu Si et al. “A multi-objective deep reinforcement learning approach for stock index future’s intraday trading”. In: *2017 10th International symposium on computational intelligence and design (ISCID)*. Vol. 2. IEEE. 2017, pp. 431–436.

[15] Sahar Sohangir et al. “Big Data: Deep Learning for financial sentiment analysis”. In: *Journal of Big Data* 5.1 (2018), p. 3.