



M Ű E G Y E T E M 1 7 8 2

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Tóth Kristóf

**TŐZSDEI TERMÉKEK
TECHNIKAI ANALÍZISE
NEURÁLIS HÁLÓZATOK
HASZNÁLATÁVAL**

KONZULENS

Dr. Ekler Péter

BUDAPEST, 2023

Tartalomjegyzék

Összefoglaló	4
Abstract	5
1 Introduction	6
2 Background	8
2.1 History of the stock market.....	8
2.2 Strategies for investing	8
2.3 Recognizing patterns.....	9
2.4 Challenges.....	10
2.5 Choice of topic.....	11
3 Related research	12
3.1 Understanding the problem.....	12
3.2 Types of neural networks.....	12
3.3 Long short-term memory (LSTM).....	14
3.4 Convolutional neural networks	15
3.5 Dataset	16
3.5.1 Processing the dataset	17
3.6 Neural network architecture.....	18
3.7 Implementations.....	20
3.8 Failures.....	23
4 Results	24
4.1 Classification	24
4.2 Price prediction	25
5 Web-based application for visualization	28
5.1 Node.js development	28
5.2 Software architecture	28
5.2.1 Angular	29
5.2.2 Nest.js	38
5.3 Model integration.....	39
6 Conclusion and future work	42
7 Acknowledgements	44
8 Bibliography	45

Összefoglaló

Mindenkinek ajánlott a félretett pénzét valamilyen módon befektetnie, erre egyik lehetőség lehet a tőzsde. Manapság viszont erősen algoritmizáltan és trendekből levont következtetések alapján működik a kereskedés, kisbefektetőként könnyű elveszni az információáradatban.

A rengeteg adatnak birtokában érdemes megfontolni annak a lehetőségét, hogy neurális hálózatokkal meg tudjuk-e jósolni a tőzsdei termékek (részvény vagy ETF) jövőbeli árát, vagy a trendeken belüli elmozdulásaikat előrejelezni, esetleg az előrejelzések alapján a megfelelő döntéseket hozni, ugyanis a neurális hálózatok képesek óriási adathalmazokban mintákat felfedezni és elkülöníteni egymástól, ezért is veszik át ezek a modellek egyre inkább a vezető szerepet az elemzésben, amire kimerítően mi emberek nem vagyunk képesek.

Egy lehetséges megoldás a tőzsdei termékek technikai analízise, amely alatt a historikus napi/heti árfolyamok elemzését értjük. Sajnos ezek a termékek is ki vannak téve a mindennapi gazdasági, politikai és egyéb történéseknek, amelyeket előre nem lehet látni, viszont jelentősen módosíthatják az árakat, kilátásokat. A dolgozat célja egy technikai analízisre alkalmas neurális hálózat létrehozása és finomhangolása, amely a múltbeli adatok alapján képes meglátni bizonyos mintákat, és az alapján tanácsot adni, hogy mi lenne a helyes döntés a vételükkel/eladásukkal kapcsolatban.

Napjainkban a webes megoldások különösen népszerűek, hiszen könnyedén elérhetők asztali és mobil környezetből, és az elkészített modelleket bele lehet „csomagolni” ezen alkalmazásokba is. A cél egy modern, jól skálázódó és látványos webes felület elkészítése az adatok megfelelő vizualizálására.

Dolgozatomban mintegy 8000 részvény és ETF több évtizedes adatát vizsgálva készítem el a modelljeimet, és kísérlek meg ezek alapján előre jelezni. Munkám során amellet hogy olyan tapasztalatokra teszek szert, illetve olyan algoritmusok kerülnek kidolgozásra, melyek más területen is felhasználhatóak, kipróbálok a neurális hálók többféle konfigurálását és paraméterezését, és megállapítani, melyik a legalkalmasabb az aktuális feladatra.

Abstract

The stock market offers a chance for everyone to invest at least some of their money throughout their lifetime. Trading has become so automated in modern times that small investors may become overwhelmed by the volume of information.

With all the historical data available, one must consider the possibility of employing artificial neural networks (ANN) to forecast any type of trend or movement of a stock or ETF (Exchange-traded Fund), or at least, use these networks to assist us in any manner throughout the decision-making process. Artificial neural networks can identify patterns in enormous datasets that people are unable to do, which is why various deep learning algorithms are becoming more and more common in the trading industry.

The technical study of stock prices, which limits consideration of these articles' historical data, maybe a way to make predictions. Unpredictable political and economic developments have a significant impact on these prices, which is unfortunate. In order to predict prices or at least aid in trading decisions, this research aims to build neural networks and fine-tune them for technical analysis.

Web applications are getting more and more popular nowadays because they are easily accessible from mobile and desktop environments, also the created models can be packaged into the apps, and used out of the box. The objective is to provide a cutting-edge, scalable, and aesthetically pleasant user interface for accurately visualizing data.

To attempt forecasting, I will analyze more than 8,000 stocks' and ETFs' more than decades-long data to create my models. During my work I will create such algorithms and get experience in a field that is applicable in multiple industries, I will configure neural networks in a variety of ways before finding out, which is the most adequate to the task at hand.

1 Introduction

We've all heard the advice that we should invest at least a small amount of our money into an asset to generate wealth with compounding interest, or at least combat inflation. Hopefully, most of us take this advice, but choosing the right paper or asset, at the right time for the right price at the proper broker is quite challenging, also as we, people, are not always making decisions purely on logical thinking, making these financial decisions can be overwhelming.

Type of paper	Description
Stocks	The owner of the stock receives shares of ownership in a public company.
Bonds	Bonds are fixed-income securities that corporations and governments issue to raise money to fund projects.
Cash	Cash investments—commonly called cash equivalents—are short-term investments (treasury bills, certificates of deposit, etc.).
Mutual funds	A mutual fund invests in groups of stocks, bonds, and other securities, its portfolio is managed by financial professionals.
Index funds	Index funds are a special form of mutual fund that is managed passively.
Exchange-traded funds	ETFs buy a basket of securities, like stocks and bonds. They can focus on a particular sector.
Annuities	Contract between an individual and an insurance company.
Derivatives	Derivatives are financial instruments whose values are based on an underlying asset (futures contracts, options contracts, swaps, etc.).

Table 1 – Types of investment assets [1]

The abundance of papers, stocks, bonds, and ETFs can also cause some headaches for us. Nowadays the most popular choice of a small investor is the stock market (this decision also depends on region, accessibility, and financial education). It is easy to register at an online broker, you can even log in with your Google account so the procedure takes only a couple of minutes. Uploading money is also made easy by most of the payment processors, and buying a piece of a business also takes only one click, so

everything is set up to make this process seamless and easy, the only hard decision(s) we have to make is which one to buy. What is the right price? What is our goal or strategy?

It is a fact that we can't predict a stock's price in the long run, as it is highly dependent on economic and political events, none of them can be foreseen, but one of these so-called black swan¹ events can highly affect stock prices in any direction, and in the short and long term, too.

For the short term on the other hand, trading has become so automated and analytic, that we have to consider the possibility, that there are certain patterns, that are recurrent on the market, and exploring and exploiting these patterns can end up in some capital gain. We have to consider that our capital is always at risk whatever we invest, and we shouldn't have irrational expectations. It is also hard for us to handle seeing our portfolio reduce to its fractional value, so it would be beneficial to have software, that can decide instead of us, or at least help in deciding if this is the right time to buy the stock and see what projections do we have for the future.

¹ <https://economictimes.indiatimes.com/definition/black-swan-theory>

2 Background

This chapter gives an overview of investing, strategies, and challenges. I will also explain why I chose this field to research, and why the topic is relevant for everyone.

2.1 History of the stock market

The history of the stock market dates back to medieval Europe, where traders gathered to exchange goods, anticipating a future rise in prices in hopes of a profit. This mostly meant purchasing and selling commodities, until in 1611, The Dutch East India Company became the first publicly traded company, traders were able to buy equity in the company. The goal was to fund ships across the sea to gather goods. Those who funded received a dividend in return. Evolving through centuries, the modern stock market embodies the exchange for immaterialized papers of thousands of companies, whose owners get ownership of the company, expecting a higher selling price or dividends. In the Western world, stocks are an established part of one's investment or retirement portfolio, unfortunately in Hungary, only a couple percent of the population hold any kind of stock.

2.2 Strategies for investing

There are multiple approaches to buying a stock. One can consider holding for the long term, that is when one should focus on the outlook of future performance of the company. This investing strategy is called value investing [2]. Investors with this strategy are aiming for undervalued stocks which could perform well in the future. One should differentiate the company's value from its price in the stock market. Measuring the company's value mostly consists of checking its balance sheet, and only if it is appealing one checks if he/she is paying the right amount of money at the time.

Investing into exchange-traded funds (ETFs²) can also be a viable option because those papers provide a basket of securities/papers, so one's portfolio is more diversified, and there is less risk of volatility.

² <https://www.investopedia.com/terms/e/etf.asp>

Moreover, there is another approach, which is more focused on the trends of the stock price, when based on the past short-term movements one is making their moves speculating for an upward and downward movement. When expecting an upward movement, one has to „long” a stock, which means that they purchase that paper, for a downward movement one can do „short selling”, which means borrowing a paper and selling it on the open market, and planning to buy back later for less money in the future, keeping the difference.³ These actions can happen at a high frequency, doing these multiple times a day is called day-trading. It is argued worldwide, if this form of investing is even investing or just speculation, and after taxes does one even have any money left, but there was a study made in 2014, that states, that it is a valid option for generating wealth, at least for the most skilled ones. They investigated the Taiwan Stock Exchange and saw that most of the traders lose money even before taxes, and those who succeed benefit from periods during high information symmetry (e.g, around earnings announcement), which suggests that these traders may possess internal information (this can not be ruled out), or at least they are making moves much more quickly [3]. It is yet unclear how they can forecast those movements, but profit can be made indeed with day trading. In this regard, we can consider strategy as investing.

2.3 Recognizing patterns

There are multiple patterns for day traders to recognize, I'll show one of the most popular trading patterns, which is the Fibonacci trading strategy.

It is using retracement levels between a previous high and low, and expecting a correction in an upward or downward trend. The local low is considered 0% and the local high of the period is 100%. The Fibonacci retracement levels are 23.6%, 38.2%, 61.8%, and 78.6%. While not officially a Fibonacci ratio, 50% is also used. After these ratios are near to Fibonacci ratios that is why the strategy is called after Fibonacci.⁴ These levels indicate that if the price is finding a support or resistance, we can expect a movement in the opposite direction shortly, so one can create positions on the market based on our expectations. To better visualize this phenomenon, you can see the retracement levels of

³ <https://www.investopedia.com/terms/s/shortselling.asp>

⁴ <https://www.investopedia.com/terms/f/fibonacciretracement.asp>

the USD/CAD currency pair, where the price retracted around 38.2% of a move down before continuing.



Figure 1 - Retracement levels [4]

This pattern does not repeat always for sure, but knowing it can help us in our portfolio. We should ask ourselves, are there any more patterns that we could follow? Do we have to recognize them, or we can automate the recognition process? What tool can analyze huge datasets and recognize patterns within? Your answer is correct, it is neural networks.

2.4 Challenges

Investing in the financial markets and analyzing prices is far more complicated than feeding a stock chart to a neural network model and seeing the results. Markets are deeply embedded in the global economy, political, military, and economic events, human psychology is even involved, the perception about the future can also be influential, and we can't base our future predictions on the past. Tobias Scädler made a study, where he investigated, whether markets are moving logically, irrationally, or somewhere in between, and if the latter, what the ratio is.

He was curious whether there were signs of movements of stock prices in any kind of periodic manner or rather behaving irrationally. To tackle this problem he used an over 200-year-old method, which is mostly used for analyzing frequencies and signal-processing, so-called Fourier transformation. The goal of the Fourier transform is to convert a function into another function, which represents the frequencies the original

function consists of. If one was able to apply this transform to stock prices and have any kind of success, one would discover the representation of the stock charts we see in another way, low frequencies would mean long-term trends, high frequencies would suggest short-term movements. The benefit of these would be that seeing these trends would give us some insights to where the stock price is going, therefore we could base our trading on that, or at least it would have an effect on us.

Tobias however, had a major issue, that stock prices are not continuous, also we have to accept the fact that stock exchanges are closed for more than 10 hours between trading days, and there is no trading during weekends and holidays, which almost take up 30% of the calendar days, however the world events do not stop. He came up with a solution that one can consider stock prices (open, close, high or low) as a discrete set of signals, and connecting them with exponential functions, because “dealing with financial markets, an interpretation of (positive or negative) compound interest seems to be appropriate”, Tobias states.

He looked for trends within the timespan of 10 days and 20 years, due to the nature of the dataset and computational limitations. He came up with the conclusion that there is more irrationality in every area of the market, more in the tech sector, and less in the industrial sector, but irrationality dominates over logical periodicity. [5] Therefore predicting stock prices solely based on their past performance is more than challenging.

2.5 Choice of topic

I chose this topic to look into because this field is the mixture of my professional and hobby activities, I devote much of my freetime to get informed about personal finance, and I am curious about the potential of neural networks and innovation in IT, I consider myself financially educated, yet deeply into learning new areas of the financial markets. While developing the web application I was inspired by popular trading platforms like Robinhood.

3 Related research

This chapter is about the research I've made to gain the background knowledge that is needed to come up with adequate solutions from an analytic and point of view, to understand the core mathematical concepts and paradigms, and as an engineer, to be able to use that knowledge to create a product.

3.1 Understanding the problem

Neural networks have multiple use-cases in several fields, from image recognition to weather forecasting. The difference between the two mentioned areas are that images are independent from each other and can be given in any order to the neural network, weather and stock prices are related to a time series, and the order of the datapoints do matter.

The hypothesis is that there is a connection between data points near each other. One can assume that models which can store any kind of state inside them will perform better in predicting future prices.

3.2 Types of neural networks

All neural network are built up from layers of perceptrons or neurons, which is the smallest building block of the network, and one can stack layers onto each other. All perceptrons have inputs, weights and biases, activation function and an output. The output of each layer is measured as the inputs are multiplied by the weight of the connection as a transformation function, all summed up and forwarded to the next layer according to the activation function. These elements will be discussed deeper in other sections of this chapter, but to clarify, this process can be described with the following equation:

$$y = f_n(f_{n-1}(\dots(f_1(x))))$$

Where y is the output, x is the input and f_i is the i th transformation function. The most popular activation functions used in neural science are the following:


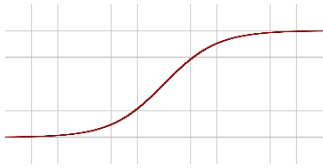
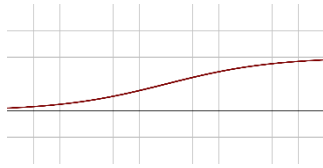
relu (rectified linear unit)	tanh	sigmoid
		
$f(x) = \max(0, x)$	$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$\sigma(x) = \frac{1}{1 + e^{-x}}$

Table 2 - Popular activation functions [6]

Building up from perceptrons or neurons, there are two broad and architecturally different neural networks, one is the so called feedforward neural networks, where information flows only in one direction between the layers, applying the feedback of a result happens with backpropagation method. One important attribute of these types of networks is their statelessness, depending on the problem one is trying to tackle it can be a benefit or a drawback.

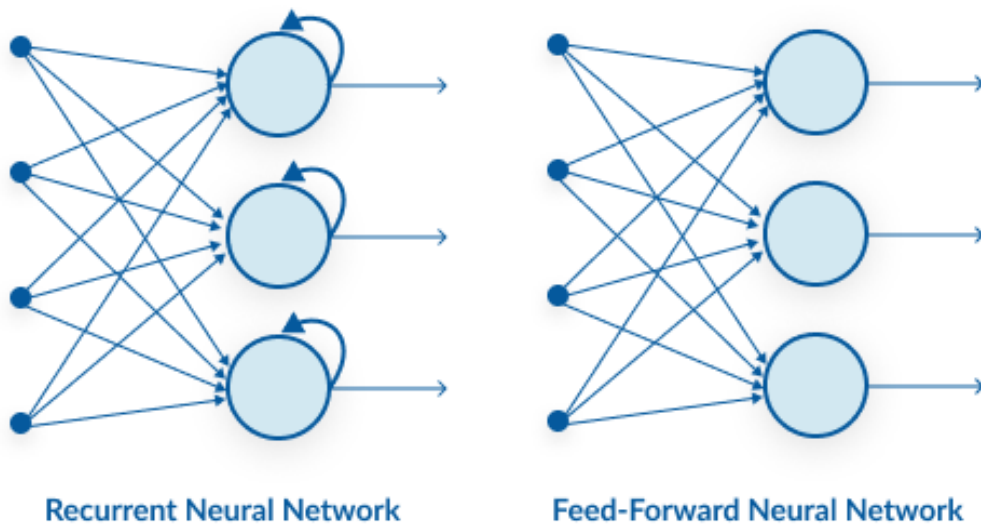


Figure 2 - Difference between recurrent and feed-forward neural networks [7]

On the other hand, there are recurrent neural networks, where flow of information can be bi-directional, which means that output of a certain layer or node can be an input of a node in the same layer, or have an effect on it. This means that these nodes can store an internal temporary state, or at least information can persist because of the dynamic

feedback they receive, which can be exploited for recognizing patterns between nearby datapoints. [8].

3.3 Long short-term memory (LSTM)

LSTM networks are a special type of recurrent neural networks, which is commonly used for series analysis. The concept of LSTM were created to aim to tackle the problem of RNN-s, which is the explosion or vanishing of gradients. This means that on the figure above during feed-back to the same node happens values are getting raised to the power of 2, after multiple repetitions values can converge to infinity or null, depending on the exponent.

What sets LSTM models apart from traditional RNNs is their ability to capture long-range dependencies in the input data. This is achieved through the use of memory cells, which allow the model to selectively remember or forget information over time. LSTM models consist of multiple memory cells, each with three main components: an input gate, a forget gate, and an output gate. The input gate determines the effect of new information in the cell, forget gate controls the information discarded. Obviously, output gate determines the output of the cell which is being sent to the next node. This architecture successfully removed the gradient problem mentioned above. [9]

Inside LSTM cells \tanh and $\text{sigmoid}(\sigma)$ activation functions are used.

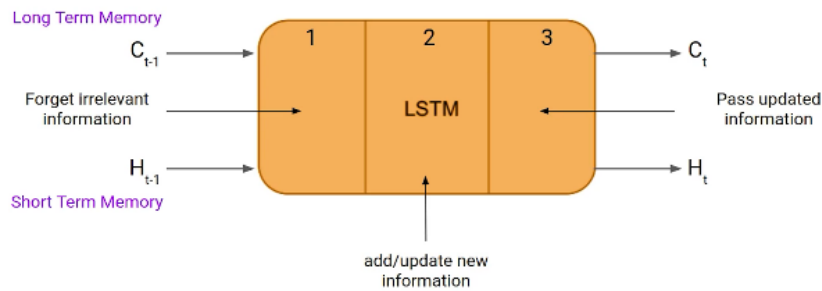


Figure 3 - High-level architecture of an LSTM cell [10]

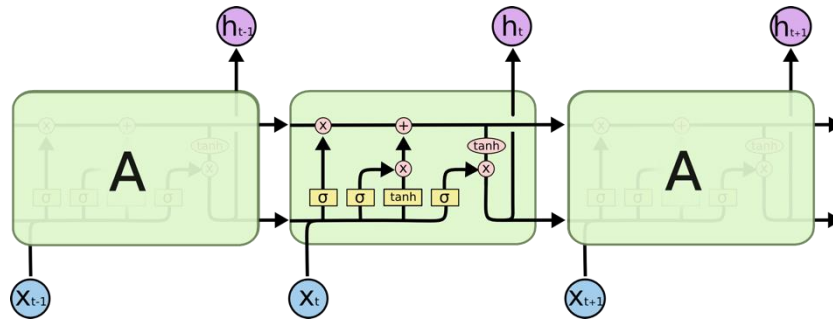


Figure 4 - LSTM cells connected to each other [11]

There is a systematic literature overview about the approaches in stock market prediction, the most popular solutions included some kind of generic neural networks, the second most popular was LSTM, appearing in 2015 and since then it is getting a bigger and bigger share of the publications, so this indicates that LSTMs are gaining popularity, which is may because of it's efficiency and versatility. [12]

3.4 Convolutional neural networks

Convolutional neural networks are commonly used for image recognition tasks, however they are also performing well in time series classification. [13]

Vanishing gradients and exploding gradients, seen during backpropagation in recurrent neural networks, are prevented by using regularized weights over fewer connections. [14]

They are a special type of feed-forward neural network, which's layers convolve the input and pass the result to the next layer. This happens with kernels/filters, which are sliding windows (in a certain dimension according to the input data's dimension) on the data. Pooling layers can be used in these models to reduce the dimension of data, and also reduce calculation time and complexity. The flatten layer's only task is to make multi-dimensional data to a one-dimensional array.

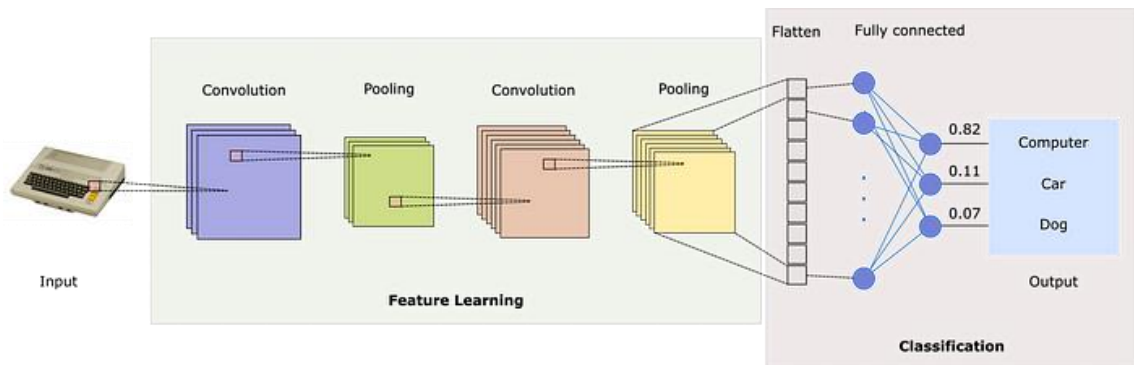


Figure 5 - Understanding convolutional neural networks [15]

According to the studies I came across LSTM models seemed to be the best performing, but from a scientific standpoint, I wanted to experiment with multiple architectures and models.

3.5 Dataset

Whatever model one creates, an appropriate dataset is mandatory. There are multiple popular websites to get data from, I acquired the data of more than 8000 stocks and ETF-s from Kaggle⁵. The only limitation of this dataset is that the latest timestamps are on the 1st April, 2020. Historic data are separated to different files for different stocks. Here you can see the structure of the datapoints of each file:

Type	Explanation
Date	Date of trading day
Open	Opening price of stock that day
Close	Closing price of that day
High	Highest price that day
Low	Lowest price that day
Adjusted close	Closing price after adjustments ⁶
Volume	Number of shares traded that day

Table 3 - Explanation of data points

Kaggle's aim is to create a machine learning and AI related community, so users can upload models and scripts beside datasets, users can view other's research and build

⁵ <https://www.kaggle.com/datasets/jacksoncrow/stock-market-dataset>

⁶ Data is adjusted using appropriate split and dividend multipliers ([source](#))

on top of that. To the dataset I acquired a user attached a script, which provided inspiration to create my models (the script is available [here](#)).

3.5.1 Processing the dataset

In this section I will discuss the high-level architecture of the model independently from the actual implementation. I will tell about that more in [implementations](#) section.

Preprocessing is a crucial step in pipelining our data. Creating an effective pipeline is one of the most important aspects of the efficiency of the neural network. The following figure explains at a high level how the data is processed in my application:

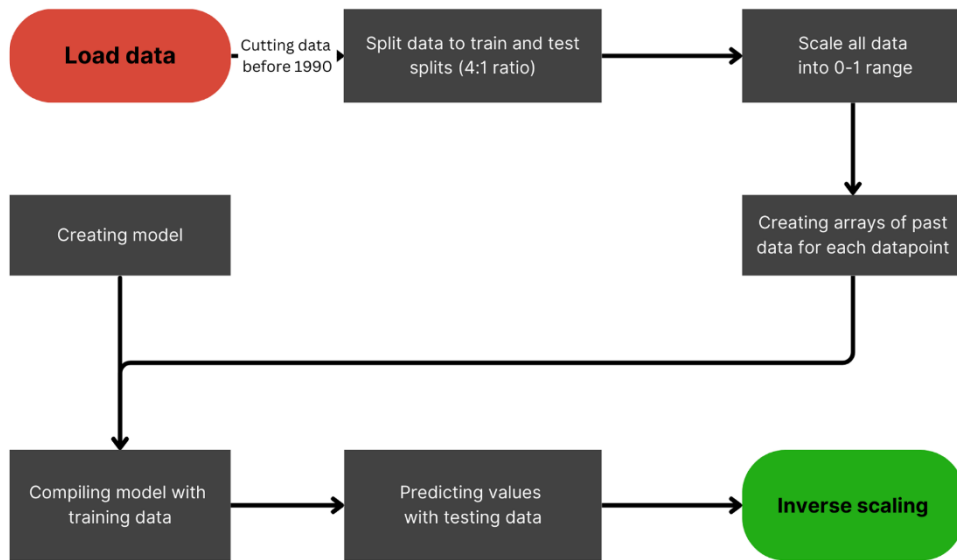


Figure 6 - High level procedure of dataprocessing

It is always a hard question how one should separate their data, in what ratio, should it be shuffled or not, it's importance is discussed in the paper of Dr. Kamila Pawluszek-Filipiak, who investigated landslide detection in her paper. Although, the problem she was discussing is much more different than mine, she raised my awareness about choosing train and test sets appropriately. [16] In the field of predicting from past data 80/20 ratio is widely accepted, so I stuck with that.

As different orders of magnitude could cause biases in the neural network, one always has to scale their data into a small range of numbers, in my case I chose the 0-1 range.

The aim of an LSTM model is to learn and discover connections between past data, so one has to attach the past n number of data points to a certain data point. After all of these preprocessing procedures the data is ready to be consumed.

After the model is created, trained and fed with the testing data, predictions are ready, but one has to inverse scale them with the original scaling method, just backwards.

3.6 Neural network architecture

The final structure of the neural networks can be seen in the tables down below, you can also find the Jupyter Notebook scripts in the [repository](#).

Layer type	Parameters
LSTM	units=64, return_sequences=True input_shape= last 2 dimensions of training dataset
Dropout	rate=0.2
LSTM	activation=relu, units=64, return_sequences=True
Dropout	rate=0.2
LSTM	activation=relu, units=64, return_sequences=False
Dropout	rate=0.2
Dense	units=1

Table 4 - LSTM network layers

If following layers are also LSTM layers, then the *return_sequences* flag should be *True* for the given LSTM layer. Dropout layers are added to avoid overfitting. *Dense* layers are used to connect to all of the previous layer's nodes, so they collect data from all possible directions. The parameter *units* is 1, because the result needs to be a list of one-dimensional data.

Layer type	Parameters
Dense	activation=relu, units=64
Dropout	rate=0.2
Dense	activation=relu, units=64
Dropout	rate=0.2
Dense	activation=relu, units=64

Dropout	rate=0.2
Flaten	
Dense	units=1

Table 5 - Feed-forward network architecture

Layer type	Parameters
Conv2D	filters=2048, kernel_size=(3,3), activation=relu, input_shape= last 3 dimensions of training dataset
MaxPooling2D	pool_size=(2,2)
Dropout	rate=0.2
Dense	units=64
Dropout	rate=0.2
Dense	units=64
Dropout	rate=0.2
Flatten	
Dense	units=1

Table 6 - Convolutional network architecture

To compare networks with different parameters developers can use the visualization toolkit Tensorboard⁷. It provides a visual representation of the learning process, architecture, biases of the networks.

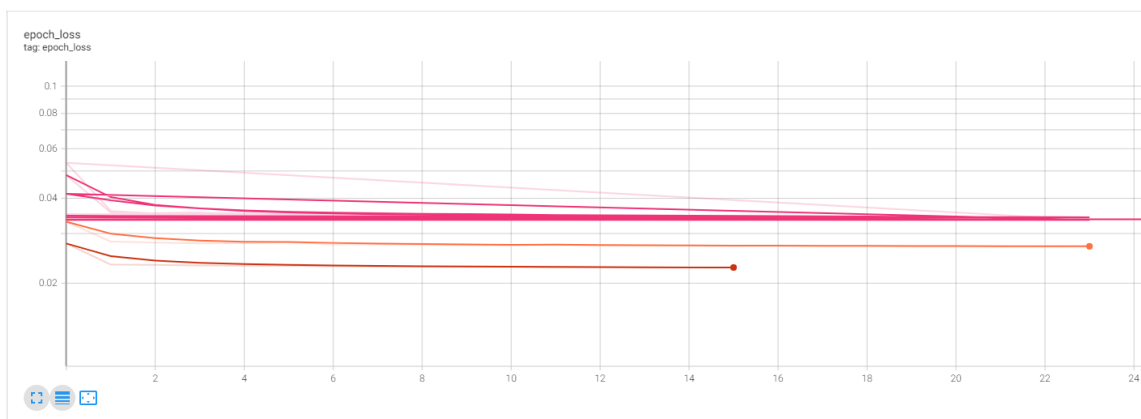


Figure 7 - Learning rate of models in Tensorboard

⁷ <https://www.tensorflow.org/tensorboard>

Tensorboard is also capable of generating a low-level architecture of the model and so much more:

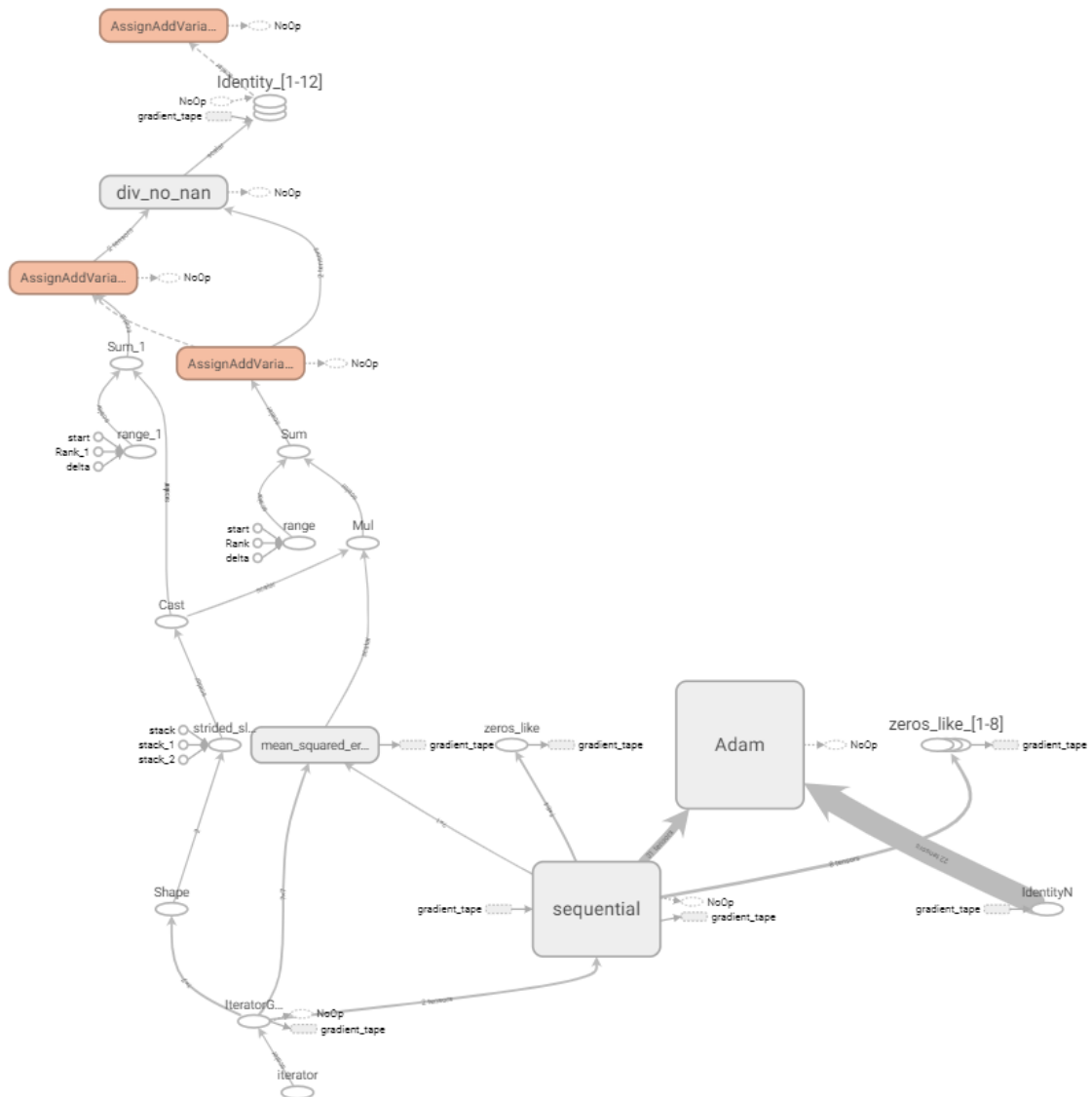


Figure 8 - Generated visual for low-level architecture by Tensorboard

3.7 Implementations

The software was implemented in Keras framework, which builds on top of the Tensorflow framework using the python language. The name „Keras (κέρας) means horn in Greek. It is a reference to a literary image from ancient Greek and Latin literature, first found in the Odyssey, where dream spirits (Oneiroi, singular Oneiros) are divided between those who deceive dreamers with false visions, who arrive to Earth through a gate of ivory, and those who announce a future that will come to pass, who arrive through

a gate of horn. It's a play on the words κέρασ (horn) / κραίνω (fulfill), and ἐλέφας (ivory) / ἐλεφαίρομαι (deceive).”⁸

Keras provides a high-level declarative approach to configure a neural network. I also used some machine learning related libraries to simplify my workflow. As I've discussed the pipelining architecture at a high level, now I'll go into detail with some implementation specialties.

Splitting data to train and test datasets can be implemented by a function call from Scikit-learn's⁹ python package.

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(dataset, test_size=0.2, shuffle=True)
```

The function gives back the train and test tuple, which can be saved into variables, also split ratio can be defined, and one can also tell the function to shuffle the dataset or not.

Another important object I needed to import to the script is Scikit-learn's *MinMaxScaler*. It can scale data in all dimensions into a feature range, which can be defined, I used the range 0-1. The scaler's mechanic is the following:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Calling the fit function on the object calculates all the necessary values for scaling the data, but does not do any transformation. The *transform* function does the scaling, and gives back the scaled array. But this can be done in one step with calling *fit_transform*, but one has to keep in mind that the *transform* function does not calculate the scaling values. It is a common technique to call *fit_transform* for the training data and call *transform* for the testing data. Therefore values in the testing data can stay outside of the feature range. After prediction the scaler can be used to inverse transform the predicted values with the *inverse_transform* function.

⁸ <https://keras.io/about/>

⁹ <https://scikit-learn.org/stable/>

Neural network models can be instantiated by creating a *Sequential* object. with the calling the *add* function on the model one can add layers¹⁰ to the network, those will be chained sequentially. After adding the layers one can compile the model with it's *compile* function, where the loss function and the optimizer has to be defined. The loss function defines the methodology of calculating a model's preciseness in predicting. The model aims to minimize the function's value. Optimizers define the learning rate, weight decay according to a certain algorithm. For example, one of the most commonly used optimizers, the Adam optimizer implements the Adam algorithm, which according to [17], „computationally efficient, has little memory requirement, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters”.

After compiling, one has to train the model with giving the following parameters:

- Training data
- Expected results
- Epochs: the amount of times that the model iterates through the whole dataset
- Batch size: the amount of data points processed at once until a backpropagation happens
- Validation split (optional): during training the model can also test it's performance with splitting the training dataset to test itself
- Callbacks (optional): callback objects to do further logging or checkpoint saving

According to the *verbose* parameter, the epochs and loss function results will be printed to the console, and if the training is done, one can call the *predict* function, where the model will give its predictions.

You can find all the data processing scripts in the following Github repository: https://github.com/GuTory/profit_prophet_data_processing.

¹⁰ <https://keras.io/api/layers/>

3.8 Failures

Besides creating models for a certain stock, I also wanted to create an all round model, that consumes all historic data at once, but I came into obstacles that I was not able to tackle until now.

Firstly, I labeled all data points with a label to know which stock it belongs to, and created theoretically the same pipeline of preprocessing to the aggregated dataset. Companies have been established and are being traded on the stock market since different points in time, so they have a different weight in training, which can cause biases, for example an ETF called SCA has only one data point, so it would get lost in a batch, so it would make no sense to predict the future price for SCA. As I had more than 28 million data points, It was even a struggle to load it into memory, not even calculating sliding windows for each data point (where memory requirements go up from 4BG to ~50-60GB), so I was unable to run my scripts, and no cloud provider offered a service for this kind of memory need free of charge. Limiting the dataset, but having a couple of million data points has even caused the loss function to be a Nan value, if that happens, there is no way to predict. Around under a million data points runs were successful, but that is only 4% of the dataset. So I stuck to individual stock training. Furthermore, stocks that do not have a long track record, fail on the compiling side, the loss function gives a Nan value, therefore, the model in incapable of predicting.

I came to realize too late that fitting data to the model multiple times train the model incrementally, I've found this discussion and it's verification in a Github issue¹¹, but I was not able to experiment this feature until now.

¹¹ <https://github.com/keras-team/keras/issues/4446>

4 Results

During my research I've created an LSTM, a convolutional and a simple feed-forward neural network for the task. The LSTM was used for actual stock prediction and also the direction of movement was measured, the other two served classification purposes, whether they are expecting a rise or fall in prices.

4.1 Classification

The convolutional and feed-forward networks were not so much different from each other, they also performed similarly. Classification was made according to which direction the model expects the price to move. When expecting an upward movement the investor's move could be a buy, for downward movement selling or shorting is appropriate.

Convolutional models reached a highest of 57% success rate for predicting the right movement, but for only a few selected stocks, training rates were mostly between 50-52%, testing rates in the range of 52-54%. Here you can see some selected stocks and their results:

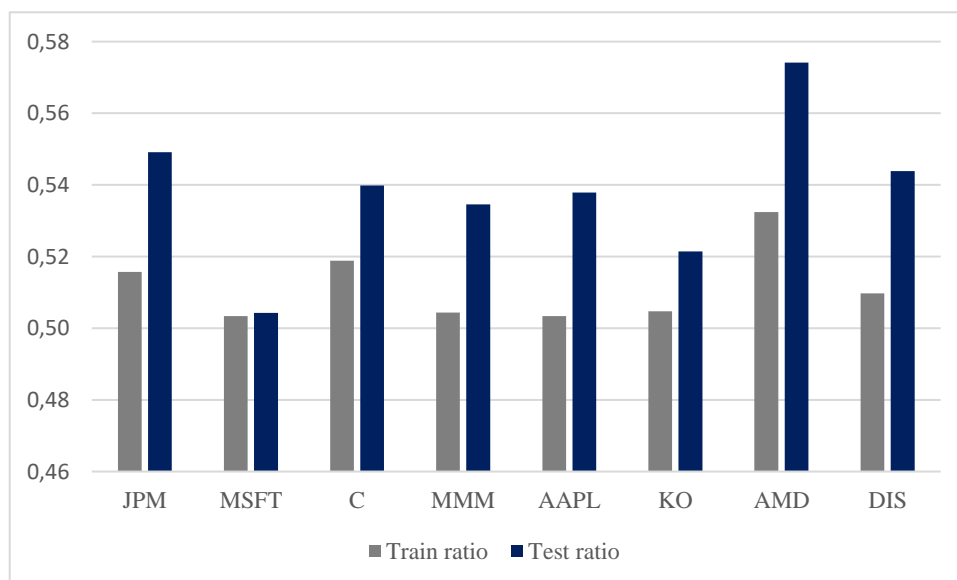


Figure 9 - Convolutional model success rate

Feed-forward networks predicted in a pretty similar way: percentagewise convolutional models predicted 0,07% better than regular feed-forward networks.

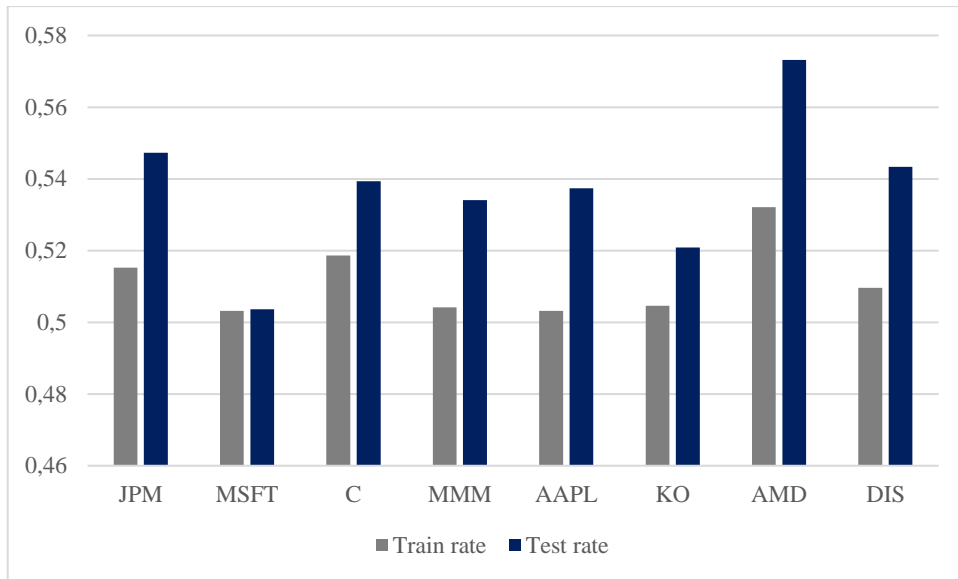


Figure 10 - Feed-forward neural network success rate

4.2 Price prediction

I also measured the prediction of right movement for the LSTM model, and it performed significantly worse than the previous models. Success rates were between 47-49%, some stocks hitting the 50% mark.

It is hard to find the right parameters for a neural network, as there are so many variations for each element. Some of the most influential parameters however were the number of past data points that were attached to a data point, as the following figures prove it:

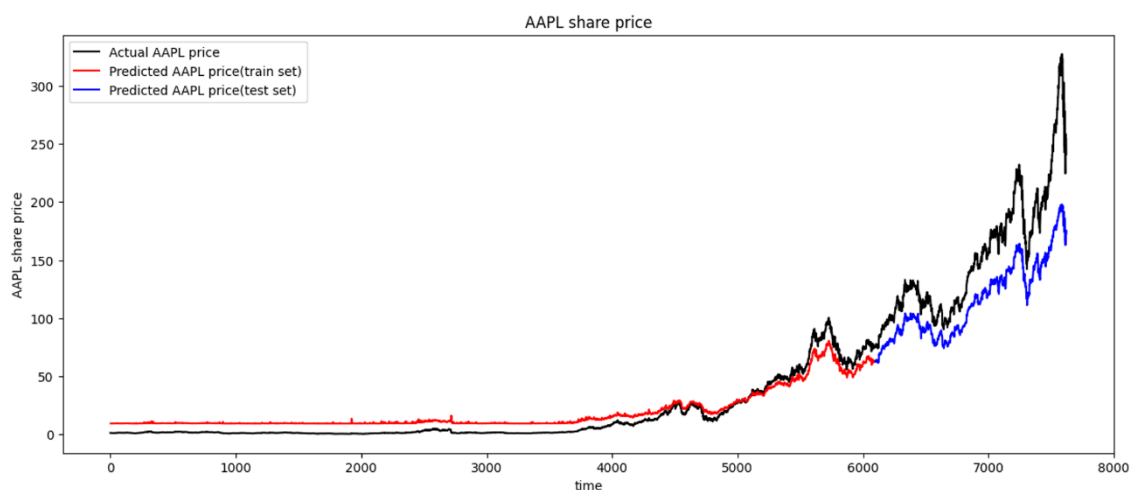


Figure 11 - Apple prediction with the data of the past 2 days

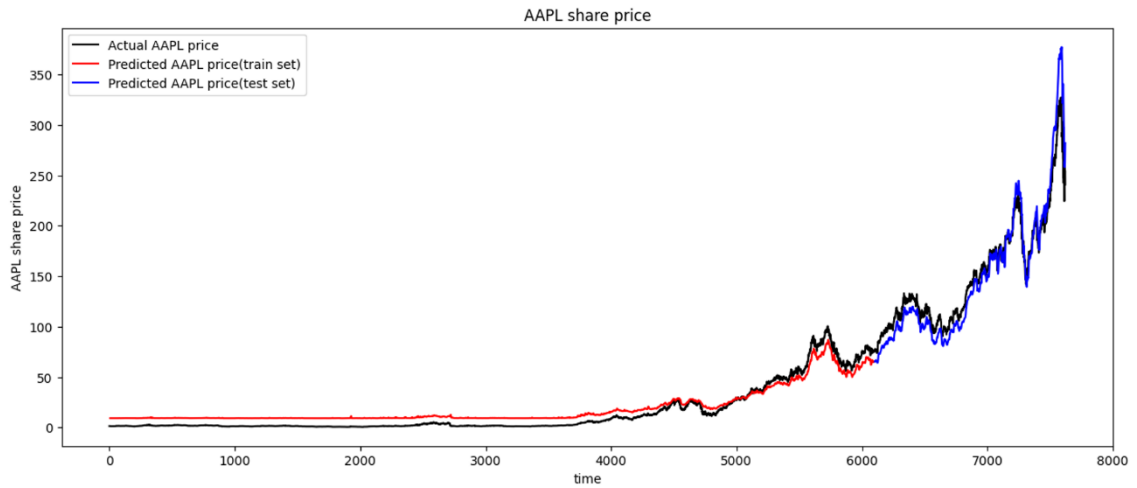


Figure 12 - Apple prediction with the data of the past 6 days

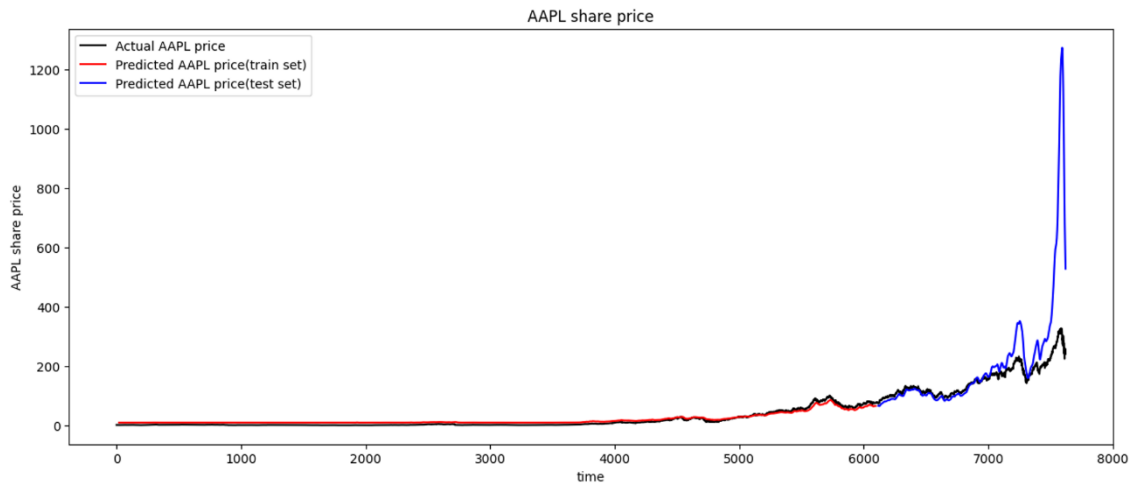


Figure 13 - Apple prediction with the data of the past 21 days

According to these findings, days between 6 and 14 seemed the most suitable.

Another crucial parameter was the [batch size](#), which not only made computations much more longer when it was a small number (between 8-16), but also made predictions worse:

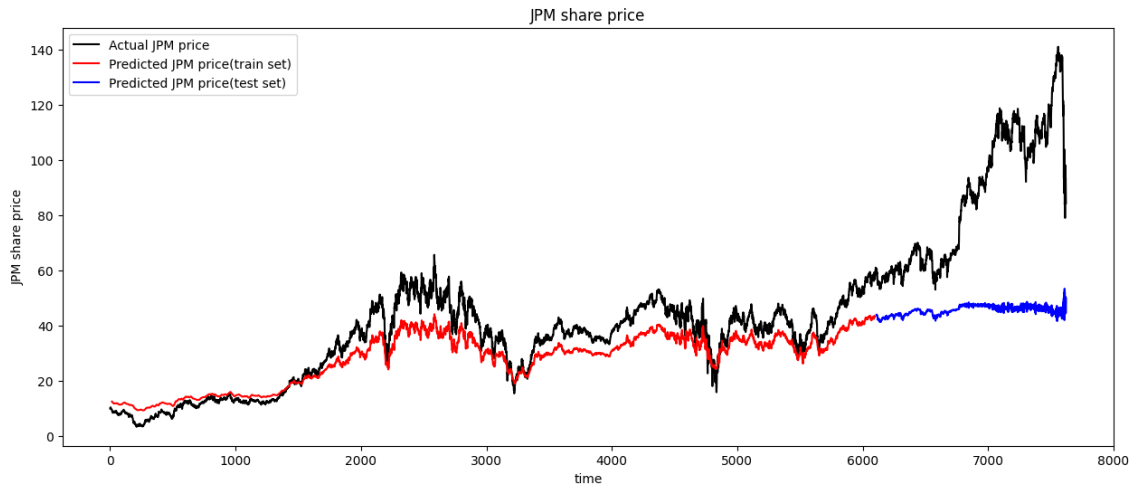


Figure 14 – J.P. Morgan prediction with a batch size of 8

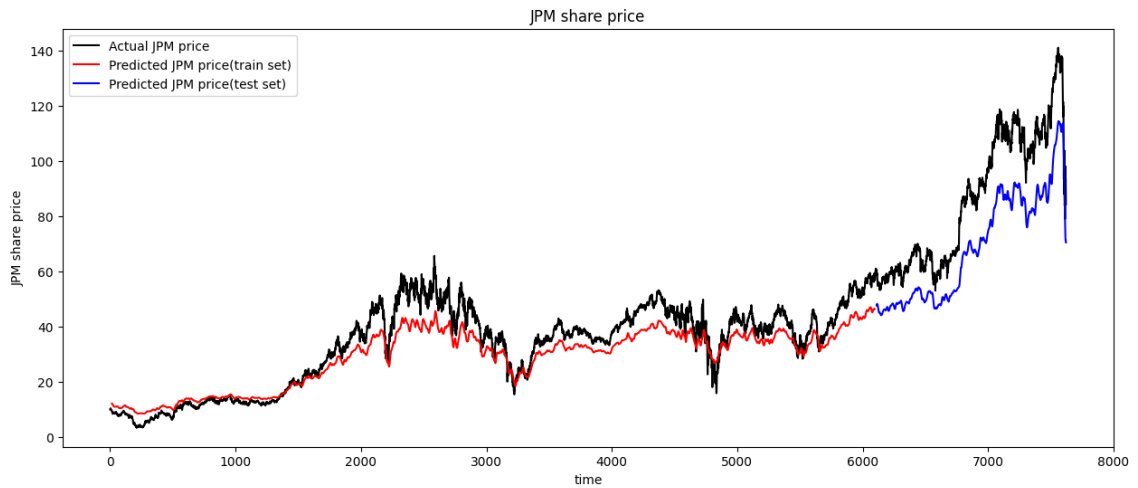


Figure 15 - J.P. Morgan prediction with a batch size of 64

Other parameters did not influence the model in such a way, but it is still possible, that there are more optimal parameters. I mention in the [future work](#) section that I am interested in finding better solutions for fine-tuning.

5 Web-based application for visualization

Aside from researching the possibilities of stock prediction and creating neural networks, I have created an all-round product, which integrates the model and presents its predictions to an aesthetically pleasing user interface. I will explain the development process and engineering perspectives for the software I've created in the following sections.

5.1 Node.js development

Node.js is an open-source javascript-based software framework for creating scalable web applications. It elevates the developer experience by a package manager (NPM, Node Package Manager), which enables importing numerous managed libraries, those can be imported and used instantly in one's application. As it is hard to create scalable and maintainable applications in JavaScript, developers can write their code in Typescript also, which helps in avoiding runtime errors, and enhances maintainability. Problems a developer comes across are easily researchable, and the community is large.

Because of the previously mentioned reasons, multiple UI and backend frameworks use Node as their backbone. In my project I used Angular for creating the user interface, and Nest.js to manage the backend logic of the application. For storing data I used Firestore, which is provided and maintained by Google.

5.2 Software architecture

The basic concept for creating the application was to use a standard three-layer architecture, where each layer has its separate roles, and they are loosely coupled.

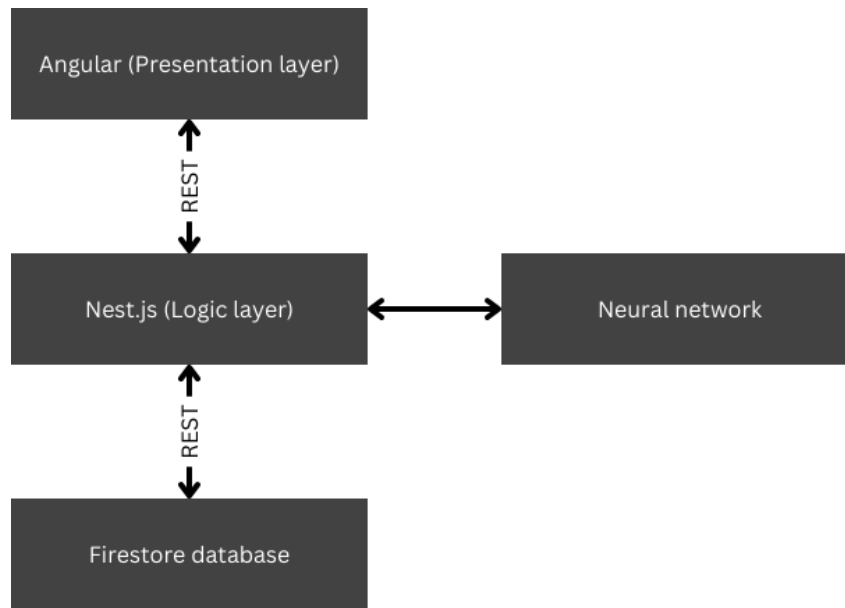


Figure 16 - Communication between application components

Inside both applications I developed I used Microsoft's Domain Driven Design¹², to further separate application components according to their domain, therefore, file structure was easy to manage and navigate in, which made development more seamless.

5.2.1 Angular

Angular is a component-based framework for creating web applications, it is developed and maintained by Google and uses Typescript as the primary language. It also has a huge collection of first-party libraries to manage routing, forms, client-server communication and so much more. From small projects to enterprise applications, Angular is able to scale, the documentation is easy to process and is up to date.¹³

The basic building blocks of Angular were components that were organised into modules. Since version Angular 14, the core concept of Angular is having *standalone components*, which have a template view, that gets rendered in the browser. Developers are able to manipulate and modify these templates according to the logic of the application. Classes in the application can be separated according to their roles:

¹² <https://learn.microsoft.com/en-us/archive/msdn-magazine/2009/february/best-practice-an-introduction-to-domain-driven-design>

¹³ <https://angular.io/>

components having a view, services providing data, model entities, structural or attribute directives that manipulate the DOM structure or the view of the component, etc.

Most of the classes use decorators, which provide metadata that tells Angular how to use them, for example, the metadata for a service class provides the information that Angular needs to make them available through dependency injection (DI). These classes are annotated with the *@Injectable* decorator.

Dependency injection is a programming pattern that enforces an object to receive all its necessary dependencies instead of instantiating them internally. This method helps developers to create loosely coupled programs, and also less boilerplate code is needed. As objects on the receiving end do not know the implementation of the injected dependencies, applications are more reusable and testable.

Angular gives an out-of-the-box solution for dependency injection, if the necessary metadata, through decorators, is provided. As modules are no longer necessary in Angular (they still can be used, but they have code structuring purposes only), one can directly define our components, which providers (services) they need, and what other imports Angular has to make. Thanks to DI, everything happens in the background, no unnecessary coding is required.

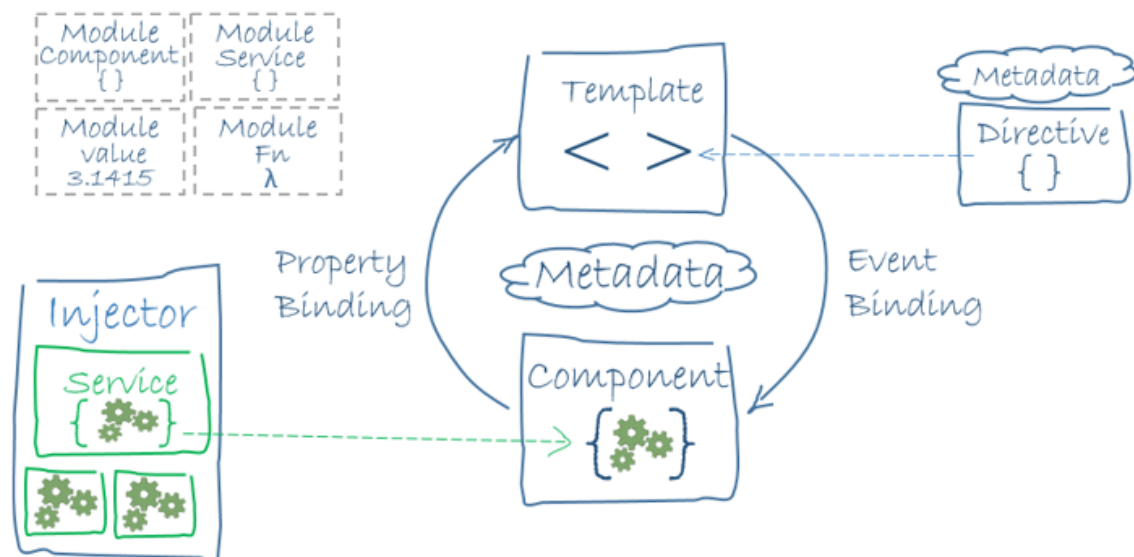


Figure 17 - Angular architecture and concepts [18]

I will not go into Angular's further details, that is only relevant from a developer's point of view, but I will mention some of the most interesting discoveries and solutions I came up with, which I feel is worth sharing.

I wanted to create a platform for traders to view their favorite stocks and see their interactive charts, so I had to integrate user handling in a way, that registration and login is made easy and is effortless. Therefore I implemented a social login with Google, where for login and registration users can opt to use their Google account, so no username and password is needed!¹⁴ This is the basic flow of authentication in my application:

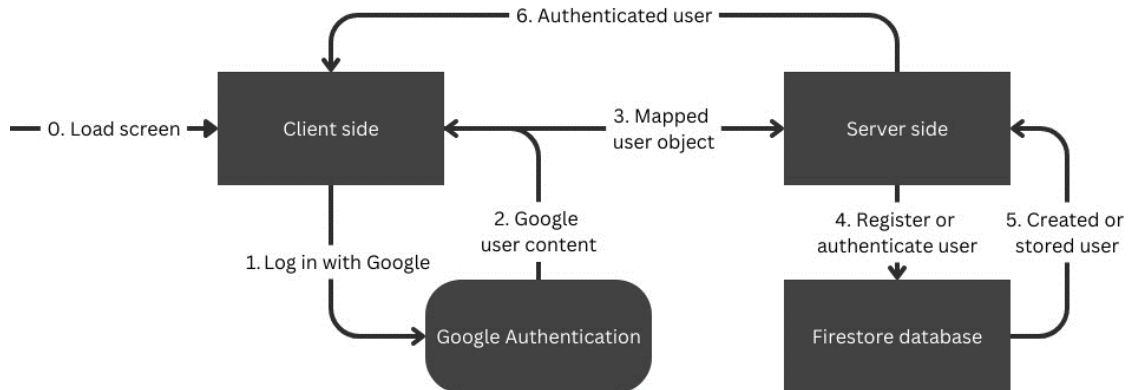


Figure 18 - Authentication flow

Angular has a different approach to asynchronous calls from other frameworks. It utilizes the RxJS library.¹⁵

RxJS, short for Reactive Extensions for JavaScript, is a library that enables reactive programming. Reactive programming is a unique and innovative programming paradigm that focuses on data streams and the propagation of change. This paradigm enables developers to easily express both static and dynamic data streams, providing a more versatile approach to handling data.

One of the core ways that RxJS is used is in handling asynchronous events, particularly within the context of Angular applications. Following the observer pattern, RxJS offers a range of operators such as *map*, *switchMap*, *filter*, *concat*, *merge*, etc. These operators are tools that help developers manage both synchronous and asynchronous code, making the handling of events much more streamlined and efficient.

¹⁴ Package used for implementation: <https://www.npmjs.com/package/@abacritt/angularx-social-login>

¹⁵ <https://rxjs.dev/>

The observer pattern is a software design pattern, which enables the multicasting of data from an object, mostly called as subject, to its dependents, called observers. Observers get notified, when the subject's data changes, therefore data flow is continuous and automatic. In RxJS, the *Subject*¹⁶ class is the core element of subjects. In my project I used *BehaviorSubject*, which is a special subject with an initial value, which is emitted whenever it is subscribed to. The benefit of this is that value initializations will cascade automatically.

A central feature of RxJS is its provision of an API for creating and working with observables. Observables are lazy Push collections of multiple values, meaning they don't compute their held values until they're subscribed to. This approach allows for a more efficient use of resources, as values are only computed when needed. In many ways, observables fill a gap in JavaScript, providing an effective way of handling multiple values over time.

	Single	Multiple
Pull	Function	Iterator
Push	Promise	Observable

Table 7 - Observables are lazy Push collections of multiple values [19]

In the Angular framework, RxJS is extensively used to handle HTTP requests and responses. It enables developers to handle these requests and responses in a more efficient and scalable manner, as the use of observables allows for effective management of multiple, possibly asynchronous, data streams. As observables are lazy type collections, data can be pulled to the UI with async pipes.

Moreover, RxJS provides observable operators that allow for the transformation of observables. These operators can be used to manipulate the data streams in various ways, such as filtering out certain values, combining multiple streams, or mapping values to new forms. Additionally, RxJS also allows for actions to be scheduled for future execution, providing even more flexibility in handling asynchronous events.

As Angular 16 was released in May this year¹⁷, I wanted to use the most up to date technological advancements it can provide. One of the most interesting feature that

¹⁶ <https://rxjs.dev/api/index/class/Subject>

¹⁷ <https://blog.angular.io/angular-v16-is-here-4d7a28ec680d>

was released with the new version are Signals. They allow the developer to define reactive values and dependencies between them. Therefore, changes are cascaded automatically throughout the application and one can write his/her code more declaratively. In the application I handled the logged in user with Signals, as some components behaving differently when the user is logged or not. In Angular, this used to be solved with the RxJS library the following way:

```
private authService = inject(AuthService);
public authenticatedUser: Observable<UserInterface | null>;
constructor() {
  this.authService.authenticatedUser
    .pipe(takeUntilDestroyed())
    .subscribe(user => {
      this.authenticatedUser = user;
      // other logic
    });
}
```

The authenticated user gets stored into the variable through the subscription. When components get destroyed, unsubscribe needs to happen, if not, that can cause memory leaks. Therefore I use the *takeUntilDestroyed()* in the pipe method, which unsubscribes when needed. Otherwise, one has to save the subscription into a variable, and can unsubscribe at the component's *onDestroy*¹⁸ lifecycle event. I find the first approach more scalable and maintainable.

With Angular Signals however, this can be solved in one line of code:

```
authenticatedUser: Signal<UserInterface | null> =
  computed(() => this.authService.authenticatedUser.value);
```

As the authentication service's variable stayed the same *BehaviorSubject*¹⁹, these two implementations are identical. You can see that with Signals it is more readable, and declarative.

There are some scenarios, where RxJS is more optimal than Signals. I came across the problem that I needed to make a call asynchronously according to the result of another asynchronous call: requiring historical stock data according to the ticker the component receives. This can be solved in nested subscriptions, but that is unsustainable. RxJS has an operator for this, called *switchMap*, which does exactly what is needed: a callback

¹⁸ <https://angular.io/api/core/OnDestroy>

¹⁹ <https://rxjs.dev/api/index/class/BehaviorSubject>

function can be defined that executes when the Observable result arrives, and the return value will be the next Observable.

```
private activatedRoute = inject(ActivatedRoute);
public stockMetaData$: Observable<StockMeta>;
constructor() {
  this.stockMetaData$ = this.activatedRoute.paramMap.pipe(
    takeUntilDestroyed(),
    switchMap(params => {
      this.ticker = params.get('ticker') || '';
      return this.stockMetaService.getStockByTicker(this.ticker);
    })
  );
}
```

The component receives the stock's ticker in the URL of the page, that can be fetched from the injected `ActivatedRoute` object, and the historic data can be only called when the result has arrived.

Angular components can also be lazy loaded, as users don't need all the available parts of the application at once. Therefore, pages can be loaded with just the necessary elements, which causes faster loading speeds and better user experience. In Angular, this is how you can load a component lazily:

```
{
  path: 'stockhistory/:ticker',
  loadChildren: () =>
    import('./stock/component/history/stock-history.component')
    .then(m => m.StockHistoryComponent),
}
```

In my project, I utilized lazy loading like in this example: when users navigate to the `/stockhistory/AAPL` page (where Apple's historical data get loaded) only then `StockHistoryComponent` gets loaded as you can see in the developer tools of the browser:

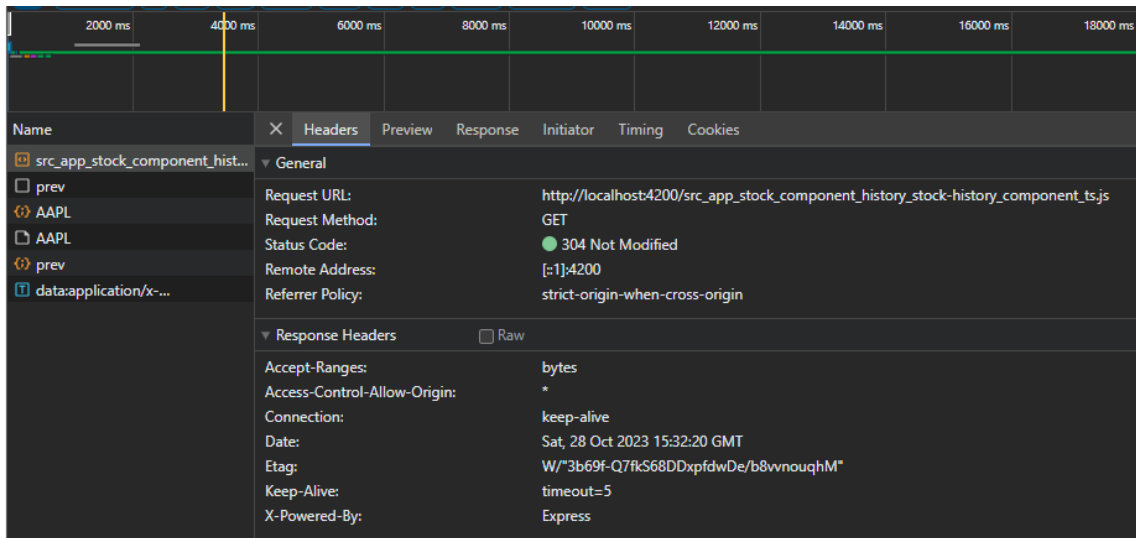


Figure 19 - Lazy loading mechanic in developer tools

One drawback of this approach is that the application’s network activity is increased as data loading is distributed in time, but this is the cost of faster load times.

For presenting the stocks’ historical data I obtained Syncfusion’s²⁰ community license, which provides countless components for multiple platforms. In addition, it is free of charge for individuals. I had to explain my project’s goal and what I would use the components for, they were helpful and friendly, shoutout to them!

Generally styling the application I used a first-party library called Angular Material²¹, which also provides out-of-the-box components, and for further customization, I used Tailwind CSS²². You can see the final form of the application in the following pictures. You can access the repository, the code and more screenshots on the following link: <https://github.com/GuTory/profitprophetfrontend>

²⁰ <https://www.syncfusion.com/>

²¹ <https://material.angular.io/>

²² <https://tailwindcss.com/>

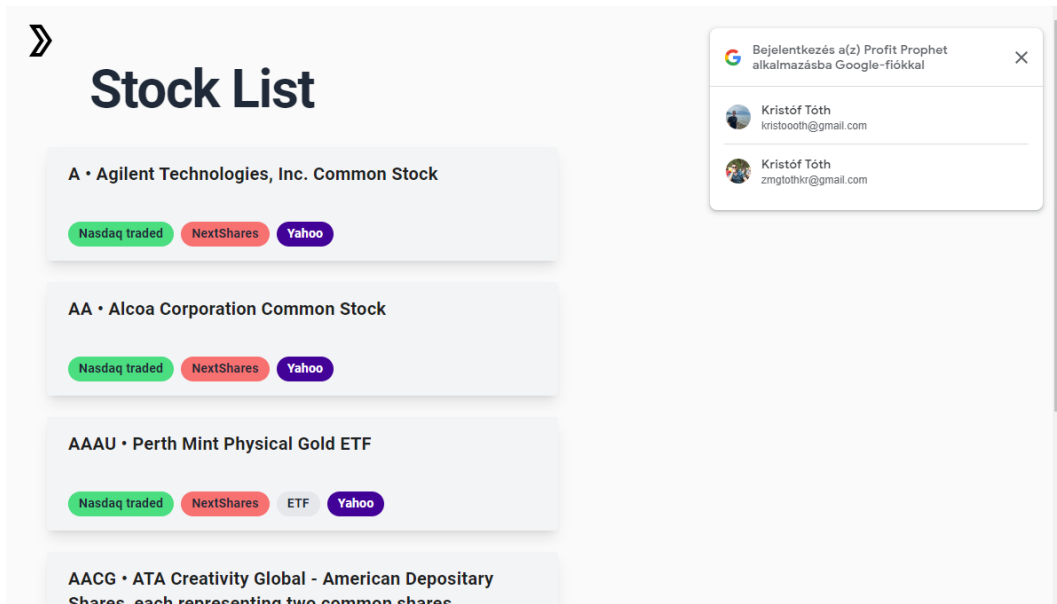


Figure 20 - Landing screen, Google authentication is appearing by default

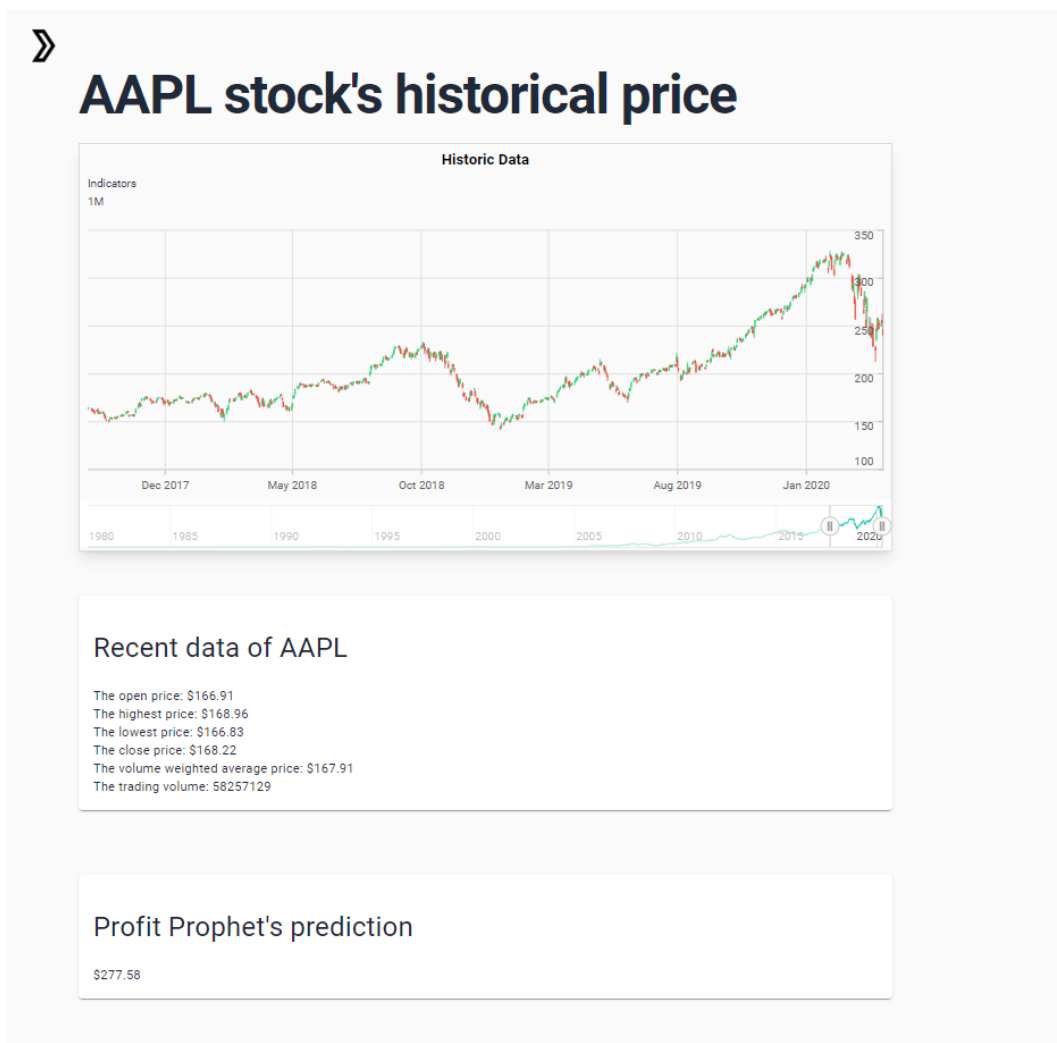


Figure 21 - Historical, and recent data of Apple, recent data is fetched from <https://polygon.io/>, Profit Prophet's prediction is the neural network's prediction presented on the UI

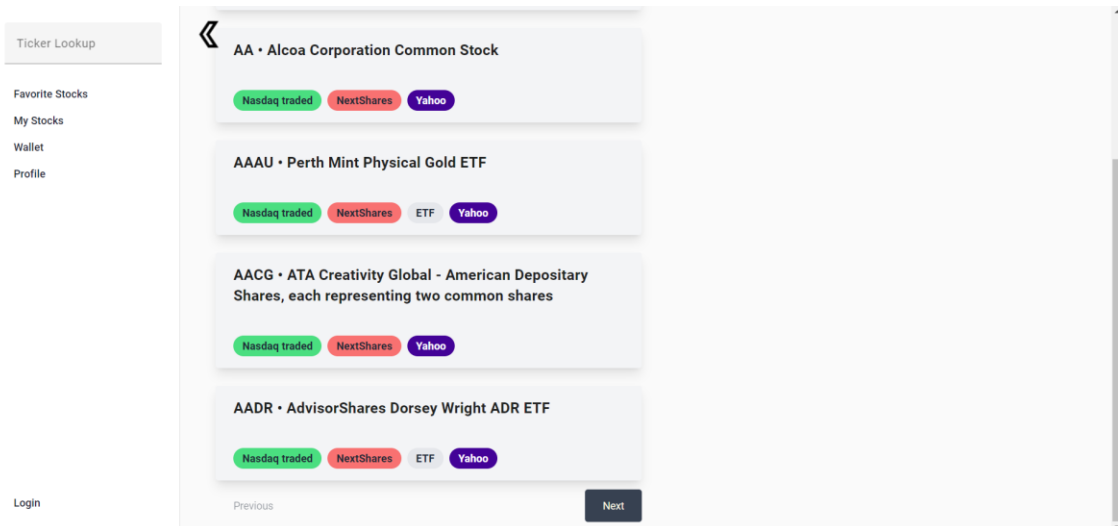


Figure 22 - Previous and next buttons for paging the data

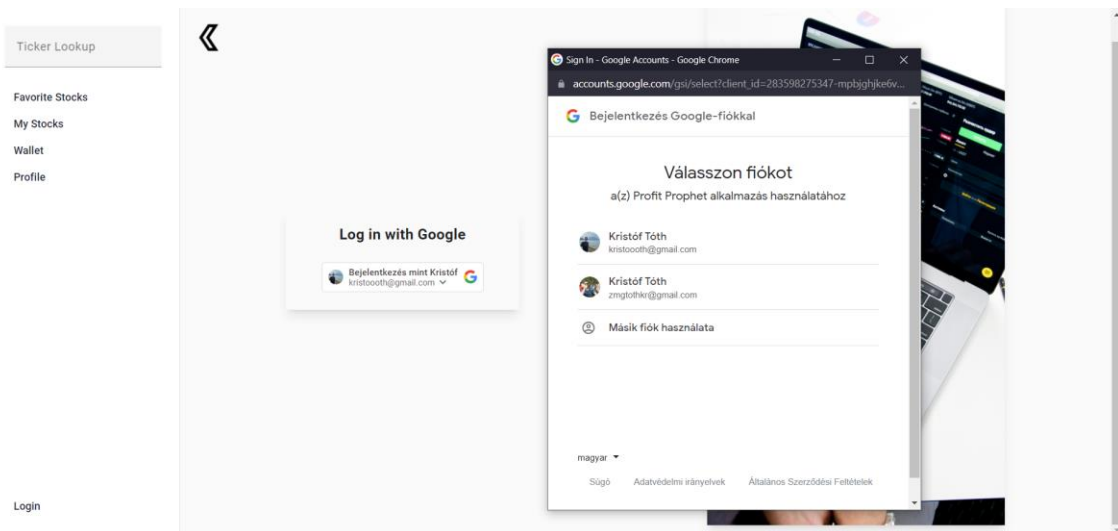


Figure 23 - The application's authentication page with Google authentication

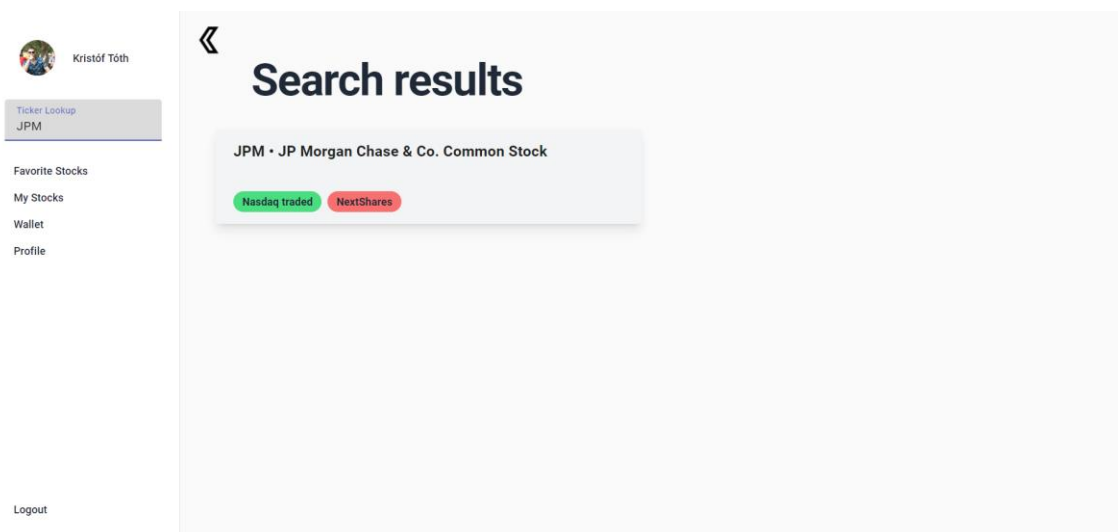


Figure 24 - Searching for symbols is enabled, the profile shows up on the top left side of the screen

5.2.2 Nest.js

Nest.js is used for developing backend applications with a modular architecture in its primary language TypeScript. The similarities with Angular became evident in the first stages in development: NPM packages, decorators, layer separation (module, controller, service), etc. [20] It also utilizes [dependency injection](#) in the same way as Angular.

As a backend application one has to define endpoints which translate to certain functions, which's results get sent back. This can be done simply by annotating the functions in the controller:

```
@Controller('auth')
export class AuthController {
  constructor(private authService: AuthService) {}
  @Post('/')
  public async authenticateUser(@Body() user: UserInterface) {
    return await this.authService.authenticateUser(user);
  }
}
```

If *AuthController* is imported to the application's module, the application has now a POST²³ endpoint: */auth/*. This example code is my project's authentication endpoint, where the user's data is being sent in the HTTP body, the service layer handles the login/registration request with Firestore in the following way:

- Query if user exists in Firestore database
- If yes, return that user
- If not, save user to database and return it's value

Firestore is cloud database provided by Google Firebase to store and sync data for client- and server-side development. It offers seamless data synchronization and robust offline support. Firestore's data is stored as collections of documents, with the capability to nest fields in documents in the form of sub-collections for more complex hierarchical data structures. It also provides powerful querying, transaction operations, and real-time updates. Firestore can be imported with NPM to any application with the following package: <https://www.npmjs.com/package/@firebase/firestore>.

²³ <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST>

In the project's application I stored stocks' historical data locally, as they take up to more than 6GB of data when stored in a JSON format. Reading these files synchronously can be done with the file system library²⁴ Other data, such as stock metadata (symbol, security name, etc.) and users are stored in the Firestore database. As I have more than 8000 stock's data in the database I also implemented pagination into my application, because fetching these data all at once is unnecessary. Firestore is a NOSQL database, so to be able to implement pagination I had to save the document reference of the first and last element of a page, to be able to define with the *startafter* or *endbefore* function in the query, which documents are wanted.

```
const q = query(
  this.collection,
  orderBy('Symbol'),
  limit(this.pageSize),
  startAfter(this.lastStock),
);
```

In this example, *q* is a Firestore query on the collection, which is defined in the constructor, documents are ordered according to their Symbol, with the *limit* function one can define the page size, and documents will be fetched starting from the *lastStock* document reference. After defining the query, one has to execute it to fetch the actual data. You can find my project's pagination implementation in [this](#) link.

The whole project's repository can be seen on this page: <https://github.com/GuTory/profitprophetbackend/>

5.3 Model integration

After initializing the model and fitting for the right data, if one finds the model good enough, he or she can choose to persist it. Luckily, in Keras, it is only a function call on the model: *model.save(filename)*. One has to define the filename, and everything is taken care of the framework. From a personal perspective, I've also saved the scaler I've used for training and testing the model, to know how much I have to inverse transform them.

²⁴ <https://nodejs.org/api/fs.html>

I successfully managed to integrate the created model into my web application. My goal was to have a REST endpoint on the backend, which consumes the stock's symbol, and according to the symbol the script gives the adequate data to the model.

As my backend is a Node.js based application, I am not able to run direct python code. Luckily, the `child_process`²⁵ package provides a service, which spawns a shell then executes the command within that shell, buffering any generated output. The command string passed to the `exec` function is processed directly by the shell. Down below you can see a simple example of executing a script in a Typescript environment.

```
import { exec } from 'child_process';

exec('python3 script.py', (error, stdout, stderr) => {
  if (error) {
    console.error(exec error: ${error});
    return;
  }
  console.log(stdout: ${stdout});
  console.error(stderr: ${stderr});
});
```

Besides the python script I added the model, and the scaled values to the backend directory, and also the data which needs to be processed. The script chooses the file according to the symbol it receives as a parameter.

However, my python script takes a couple seconds to finish, let alone the creation of a shell. As you can see one can define a callback function which gets called when the script exits, but the web application does not wait for it's execution to finish. Therefore, I had to “promisify” it by bundling it into a Promise object. The `'util'` library gives an easy way for that: `util.promisify(function)`. Whatever function is added as a parameter, the result will be an asynchronous function, which can be called like I did in my project:

```
import { exec } from 'child_process';
import * as util from 'util';

const asyncExec = util.promisify(exec);
const childResult: { stdout: string; stderr: string } = await asyncExec(
  `C:\\Users\\Dell\\AppData\\Local\\Programs\\Python\\Python311\\python.exe
  ${this.pythonScriptPath}
  ${ticker}`
);
```

²⁵ https://nodejs.org/api/child_process.html#child_processexeccommand-options-callback

The `await` keyword assures that the result will be awaited, and then it can be captured, in my implementation the result appeared in the `stdout` variable, as the `promisify` function had a signature like the above for its return. After parsing the result the backend can give back the final prediction of the stock or signs of error. I decided to only give back the last data point of predictions as it is the most relevant.

You can ask why my project does not call python scripts with `python3` command, that is because I was struggling with running the script for hours, because giving the `python3` command instead of `C:\...\Python311\python.exe` the script loaded a previous version tensorflow (2.11.0, and 2.14.0 would have been required), therefore modules were not found (`tensorflow.contrib`) and the script failed.²⁶ I also found out, that the issue was because with the `python3` command the Command Prompt was using python version 3.7.9, whereas I needed 3.11.5. With better care for environment variables, this problem can be avoided.

Although, predictions were successfully integrated into the application, running the script is pretty time-consuming, which needs some further enhancements.

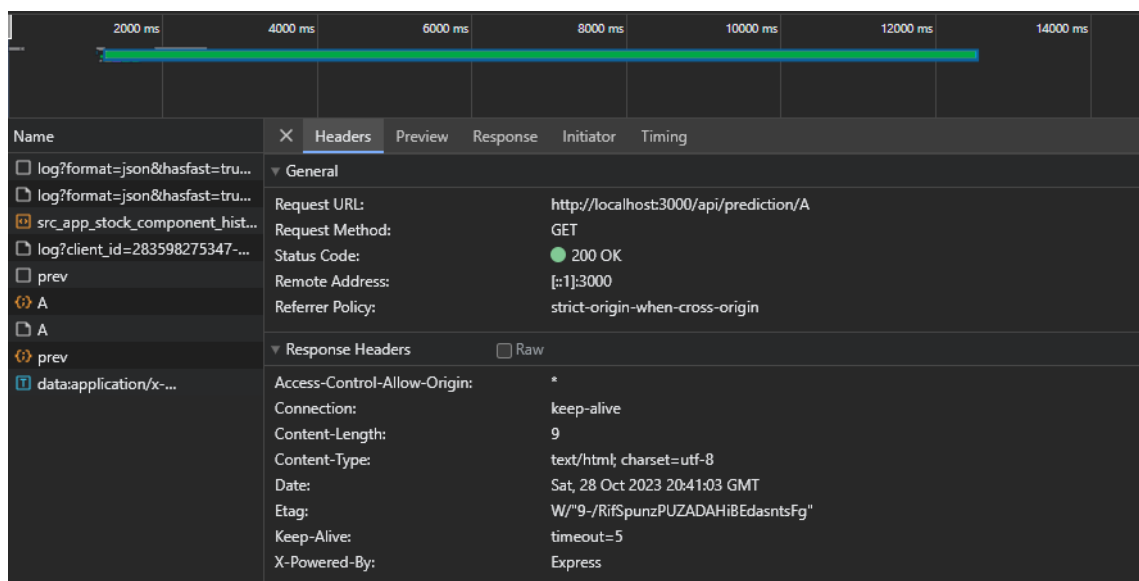


Figure 25 - Prediction takes more than 10 seconds to finish

²⁶ <https://github.com/tensorflow/tensorflow/issues/30794>

6 Conclusion and future work

In my paper, I investigated the possibilities of using neural networks for predicting stock prices based on historical data and integrated that model into a web application. Although my models are barely or not even above the 50% baseline for hitting the right direction, which is almost the same as flipping a coin, classification models were able to go up as high as finding the right direction of movement 57% of the time, but only for a few selected stocks, so I do not have the confidence yet to trade according to the models expect the stock to move.

For actual stock prediction, I used only LSTM models, whose predictions converged to the actual stock price, but the measure of movements was highly swinging according to the sliding window of past days I provided for each data point, and even the right direction was defective close to 50% of the time, predictions were lagging behind, that is how. Convolutional and regular neural networks performed better in predicting upward or downward movement. Also it would be beneficial to research, whether ETFs can be predicted more precisely, as they are less volatile, they are a basket of certain stocks, so more diversified.

Ideas are constantly coming to my mind for further improvements that could be made in this application. It is for sure, that further fine-tuning is required to the model, so that it could predict better, on the other hand however, without the information and impact of external events, relying only on the past performance of the paper is a naive approach and one cannot expect any kind of breakthrough. It is like a table having a maximum of three legs, rather two... I have also not explored the potential of transformer models which could further enhance the model's performance. [21]

To consume the economic and political events one could create a webscraper to scan the news, and classify them regarding emotions, outlook and insights for the market. However, people can react in a different manner for the same events, so our job would not done yet, one can only assume the possibility of reactions. Where human psychology is involved, actions get less logical.

In addition, the model could process the given stock's balance sheet to further improve the perception of the company.

Finally, regarding the neural networks I am curious whether there are direct methods to find out what inner architecture is suitable for certain problems instead of trying out and seeing results, I would like to be able to calculate how many layers, how to combine them, which activation functions and parameters are optimal.

Nonetheless, for a cleaner and more scalable architecture regarding the web application, the model could be hosted on a cloud platform, rather than storing it locally. That however, is more costly, and would have created such an implementation overhead, that I've decided that it is not necessary for now.

7 Acknowledgements

This research was supported by the the European Union project RRF-2.3.1-21-2022-00004 within the framework of the Artificial Intelligence National Laboratory.

8 Bibliography

- [1] B. C. Kat Tretina, „9 Types Of Investment Assets,” 2022. [Online]. Available: <https://www.forbes.com/advisor/investing/types-of-investment-assets/>. [Accessed: 29 October 2023].
- [2] B. Graham, *The Intelligent Investor*, 1st Edition szerk., New York, NY: Harper & Brothers, 1949.
- [3] Y.-T. L. Y.-J. L. T. O. Brad M. Barbera, „The cross-section of speculator skill: Evidence from day trading,” 2 July 2014. [Online]. Available: [https://pdf.sciencedirectassets.com/271966/1-s2.0-S1386418114X00029/1-s2.0-S1386418113000190/main.pdf?X-Amz-Security-Token=IQoJb3JpZ2luX2VjEOz%2F%2F%2F%2F%2F%2F%2F%2F%2FwEaCXVzLWVhc3QtMSJGMEQCIBM5J%2Ft4Q6qIYE5JjT0FufIusiEOSxbCIYjS5UR%2BqZoUAiBvzwWace9h](https://pdf.sciencedirectassets.com/271966/1-s2.0-S1386418114X00029/1-s2.0-S1386418113000190/main.pdf?X-Amz-Security-Token=IQoJb3JpZ2luX2VjEOz%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FwEaCXVzLWVhc3QtMSJGMEQCIBM5J%2Ft4Q6qIYE5JjT0FufIusiEOSxbCIYjS5UR%2BqZoUAiBvzwWace9h). [Accessed: 22 October 2023].
- [4] „Fibonacci retracement,” [Online]. Available: https://en.wikipedia.org/wiki/Fibonacci_retracement. [Accessed: 15 October 2023].
- [5] TobiasSchädler, “<https://www.researchgate.net/>,” 25 December 2018. [Online]. Available: https://www.researchgate.net/profile/Tobias-Schaedler/publication/330089334_Measuring_Irrationality_in_Financial_Markets/links/5c2cb67ea6fdccfc707804d3/Measuring-Irrationality-in-Financial-Markets.pdf. [Accessed 10 October 2023].
- [6] „Activation function,” [Online]. Available: https://en.wikipedia.org/wiki/Activation_function. [Accessed: 15 October 2023].
- [7] D. Kalita, „A Brief Overview of Recurrent Neural Networks (RNN),” 2023. [Online]. Available: <https://www.analyticsvidhya.com/blog/2022/03/a-brief-overview-of-recurrent-neural-networks-rnn/>. [Accessed: 20 October 2023].

- [8] P. J. Werbos, „Generalization of Backpropagation with Application to a Recurrent Gas Market Model,” May 1988. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/089360808890007X>. [Accessed: 23 October 2023].
- [9] J. S. Sepp Hochreiter, „Long Short-term Memory,” *Neural Computation*, 1997.
- [10] S. Saxena, „What is LSTM? Introduction to Long Short-Term Memory,” 2023. [Online]. Available: <https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/>. [Accessed: 26 October 2023].
- [11] C. Olah, „Understanding LSTM Networks,” 2015. [Online]. Available: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>. [Accessed: 22 October 2023].
- [12] L. H. J. A. C. A. S. a. K. A. Mintarya, „Machine learning approaches in stock market prediction: A systematic literature review.,” *Procedia Computer Science*, p. 96–102, 2023.
- [13] H. L. S. C. J. L. a. D. W. Bendong Zhao, „Convolutional neural networks for time series classification,” *Journal of Systems Engineering and Electronics*, pp. 162-169, 2017.
- [14] V. K. R. a. S. R. Balas, *Recent Trends and Advances in Artificial Intelligence and Internet of Things*, Springer Nature, 2019.
- [15] N. Shahriar, „What is Convolutional Neural Network — CNN (Deep Learning),” 2023. [Online]. Available: <https://nafizshahriar.medium.com/what-is-convolutional-neural-network-cnn-deep-learning-b3921bdd82d5>. [Accessed: 23 October 2023].
- [16] K. a. B. A. Pawluszek-Filipiak, „On the Importance of Train–Test Split Ratio of Datasets in Automatic Landslide Detection by Supervised Classification,” 15 September 2020. [Online]. Available: <https://doi.org/10.3390/rs12183054>.

- [17] J. B. Diederik P. Kingma, „Adam: A Method for Stochastic Optimization,” 30 January 2017. [Online]. Available: <https://arxiv.org/abs/1412.6980>. [Accessed: 26 October 2023].
- [18] Google, „Introduction to Angular concepts,” 2022. [Online]. Available: <https://angular.io/guide/architecture>. [Accessed: 25 September 2023].
- [19] Developers of RxJS, [Online]. Available: <https://rxjs.dev/guide/observable>. [Accessed: 30 September 2023].
- [20] G. Sharma, „LinkedIn,” 2023. [Online]. Available: <https://www.linkedin.com/pulse/why-nestjs-better-choice-building-nodejs-applications-gautam-sharma/>. [Accessed: 28 October 2023].
- [21] N. S. N. P. J. U. L. J. A. N. G. Ł. u. K. a. I. P. Ashish Vaswani, „Attention Is All You Need,” *31st Conference on Neural Information Processing Systems*, 2017.