Stock Prediction using Deep Learning and Google Trends Data

Michael Engel, Daniel Sikeler

TABLE OF CONTENTS

I. ABSTRACT

Neural networks experienced a steep rise in popularity over the last years. One reason for this trend is the versatility of these networks. They are being deployed in many domains such as computer vision and pattern recognition, predictions, robotics and self-driving cars and many more. However, in the financial domain only a few scientific papers are published which aim to predict the stock course development. This may be due to the difficulty as stock prices are influenced by a multitude of seemingly unpredictable and uncorrelated factors. Nevertheless, the development of a stock course depends strongly on the actions of traders.

In this paper a possible correlation between google trends search terms and stock courses is assumed and examined by using deep learning methods. Previous to the investigation the basic idea and necessary background is explained in more detail. Subsequently, as deep learning methods strongly rely on the data fed into this system, the deployed datasets as well as the executed preprocessing are described. Next, an overall description of the project architecture, including the used deep learning model, is given. In a final chapter the previously described implementation of the stock prediction model is being evaluated according to its accuracy.

II. INTRODUCTION

Neural networks experienced a steep rise in popularity over the last years. One reason for this trend is the versatility of these networks. They are being deployed in many domains such as computer vision and pattern recognition, predictions, robotics and self-driving cars and many more.

However, quite few scientific papers of neural networks are released in the financial domain with the objective to predict the development of stock prices. The task of stock course prediction is difficult as the stock prices are influenced by a multitude of seemingly unpredictable and uncorrelated factors. Nevertheless, the development of a stock course depends strongly on the actions of traders. Those traders could use the search engine google to gather information about stocks they are interested in right before a trade. The service *Google Trends* views various graphs displaying data of search terms typed in by users at the google search engine. This service is accessible by the public. Based on these thoughts the following hypothesis can be formulated:

A correlation between google trends search terms and stock courses exists.

In this paper the previous hypothesis is being investigated. In order to check the existence of a correlation between goolge trends data and stock courses, a neural network will be implemented and verified.

III. RELATED WORK

The stock market is commonly described as chaotic, volatile, complex and, therefore, seemingly unpredictable. However, the ability to predict the development of stock prices promises to yield much profit. Because of this there were many efforts to forecast the seemingly unpredictable. As a result of the hype for deep learning and its versatility of application, the scientific research to apply machine learning in the domain of stock market prediction was triggered. However, older scientific work does not provide encouraging results. Although this is rather unpromising, recent research work used the ever improving methods of deep learning to increase the precision of the forecast for stock development.

In Singh and Srivastava (2017), for example, the authors propose a combination of (2D)2PCA and Deep Neural Network and compare the results with other state of the art methods. Compared to the 2-Directional 2-Dimensional Principal Component Analysis (2D)2PCA and Radial Basis Function Neural Network (RBFNN) the proposed method achieved an improved accuracy of 4.8%. Also, the accuracy of the proposed DNN combination is about 15.6% higher than the achieved precision of a Recurrent Neural Network (RNN). This result encourages the use of modern deep learning methods to predict stock development.

A more comprehensive assessment was done in Chong and Park (2017). The authors analyzed advantages as well as drawbacks of different deep learning algorithms for predicting the stock development. They examined the fitness of three feature extraction methods, like the principal component analysis, on overall ability of the network to predict the development of the stock market.

In this paper a special form of RNN, the Long short-term memory network, is used to predict the future development of stock prices. Although the results in Singh and Srivastava (2017) discourage the use of RNN, the suitability of a LSTM was not investigated in the reference paper. Additionally, as the LSTM "remembers" values over an arbitrary interval of times, it is well-suited for predicting time series. By using Google Trends data as input it may prove as a valid model for predicting stock prices. The basic idea is described in more detail in IV.

IV. BASIC IDEA

As already stated in the Introduction II, this paper tries to implement a deep learning model to predict future stock courses by using Google Trend data. However, as the prediction of stock development is a challenging task, a simplified goal is being defined.

Usually price prediction focuses on forecasting the exact price for the following day. One key characteristic of stocks is that they are strongly volatile. In this context, volatility describes the amplitude of fluctuation for the price of a stock between one day and the following. According to Schieche (2007), stock courses are stochastically often modeled as the relative and logarithmic volatility:

$$change_{rel} = \frac{Value_{t1} - Value_{t0}}{Value_{t0}} \tag{1}$$

$$change_{log} = ln(Value_{t1}) - ln(Value_{t0}) \tag{2}$$

Due to this feature and the overall complexity of predicting the exact stock price for the next day, a simplifying assumption is made. The stock development can be abstracted to an increase or decrease of the price for a given stock. The case of an unchanged stock price for one day is unlikely, wherefore only these two cases are relevant. In this paper the main goal is, therefore, to predict an increase or decrease for a given stock for the following day. Thereby, the task of predicting the exact price of a stock for the following day is transformed into a simple classification problem with only two classes. If the simplified model yields promising results, a more complex variation is worth investigating using the implemented model.

V. BACKGROUND

A. Recurrent Neural Network

There are many architectures specialized on different tasks. Convolutional networks are used for processing data that is organized as grid, like images. This kind of nets can also be used for time-series data, which are a 1-D grid. But often the data can not be interpreted as a 1-D grid and then another architecture is better suited. Recurrent neural networks (RNN) are specialized on processing sequential data.

The hidden units of a RNN have recurrent connections. So the

Fig. 1. recurrent neural network with hidden-to-hidden connections (Goodfellow et al., 2016, p. 373)
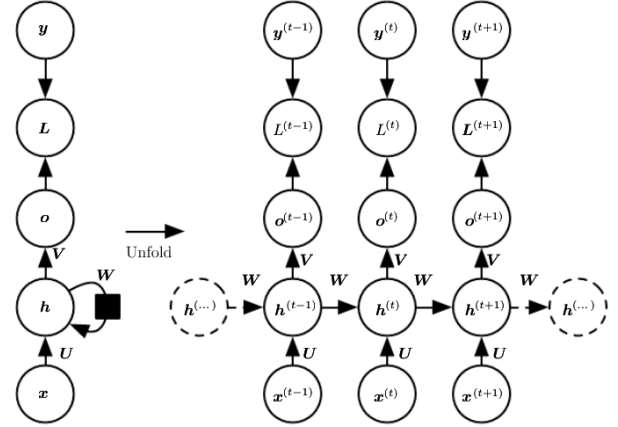
result of a hidden unit at time step t is used in the following time step $t + 1$. It is important to notice that it is the same unit, which uses the result. As a consequence the parameters are shared during time. This is a key benefit, because it enables us to process sequences of different length. The update rule is also the same as it is the same unit.

There are different patterns, which combine the units in a different way. A RNN with connections between hidden units and a output at each time step (see figure 1) can compute the same as a turing machine. In this sense the network is universal. It produces a sequence of outputs of the same length as the input. It is also possible to only produce a single output after the last hidden unit.

Like in any other neural network different output and loss functions are possible. Normally the softmax is used for the output. One common activation function is the hyperbolic tangent. The following equations show the update process of the forward propagation.

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)} \tag{3}$$

$$h^{(t)} = tanh(a^{(t)}) \tag{4}$$

$$o^{(t)} = c + Vh^{(t)} \tag{5}$$

$$y^{(t)} = softmax(o^{(t)}) \tag{6}$$

"where the parameters are the bias vectors b and c along with the weight matrices U, V and W, respectively, for input-to-hidden, hidden-to-output and hidden-to-hidden connections."(Goodfellow et al., 2016, p.374)

The generalized back-propagation algorithm can be applied to

the unfolded computational graph. It is called back propagation through time (BPTT). You can see the unfold operation in figure 1. Therefore, we need to store all the sequential states because they are used in the computation of the gradient. If we increase the number of recurrent layers the computational and memory cost increases as well. So for deep recurrent neural networks we have to meet the challenge of efficient training.

B. Long-Term Dependencies

When we are training sequences of only length 10 to 20 the stochastic gradient descent (SGD) algorithm has already some problems. The gradients either tend to vanish, if it is in the interval $[0, 1[$, or tend to explode. The problem is that we reuse the unit in every time step and therefore the gradient is up to the power of the number of time steps.

Another problem is that the long-term dependencies have smaller magnitude on the weights than short-term dependencies have. The gradient is hidden by the smallest fluctuations arising from the short-term dependencies. As a consequence it takes very long time to train long-term dependencies. To overcome this problem we could introduce skipping connections where time step t also uses the result from time step $t - 2$ or even further in the past. So we have multiple time scales, one fine grained and one distant past.

Leaky units are a different way for remembering information about the past. A leaky unit is similar to a running average. A linear self-connection is used in the update rule.

$$y^{(t)} = \alpha y^{(t-1)} + (1 - \alpha)v^{(t)} \tag{7}$$

With the parameter α one could regulate how fast the information of the past is discarded. If α is near zero the information is kept for a long time. It would be the best if the network could learn this parameter and must not be set as a hyper parameter.

C. Long Short-Term Memory

The Long Short-Term Memory (LSTM) is a gated recurrent neural network. Gated in this case means that the weight of the self-loop is controlled by another hidden unit. LTSM performed very good in handwriting and speech recognition, for example. The basic idea is that it might be useful to forget information of a leaky unit. As always it is better to have the network learn when to forget and not to decide manually.

In a LSTM network the hidden units consist of so called long short-term memory cells. A LSTM cell has three gating units (see figure 2). The input gate is a sigmoid unit and controls the input feature. The input feature is a regular artificial neuron. The forget gate has the same task as the parameter α in a leaky unit. As there are only values between 0 and 1 are allowed, a sigmoid unit is used again. It controls how fast the information of the past is discarded. In other words it controls the weights of the state unit. The state unit has a self loop similar to the leaky units. With this self loop an internal recurrence is added to the outer recurrence of the LSTM cell. The output gate is also a sigmoid unit. It can shut off the output of the whole LSTM cell. So a Long Short-Term Memory network is far more complex than a simple recurrent neural network.

Fig. 2. long short-term memory cell (Goodfellow et al., 2016, p. 405)
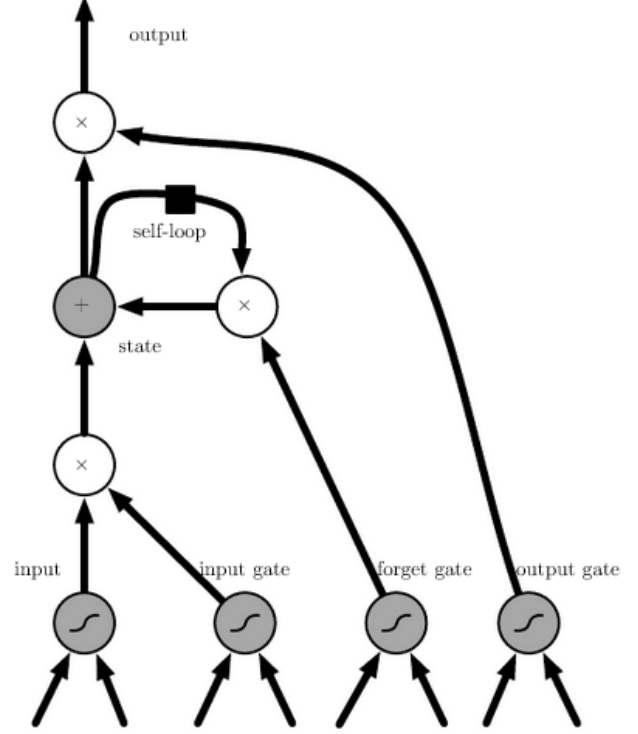
VI. DATASETS

There are many databases, which provide a lot of datasets. One of those is *kaggle*. *kaggle* does not only provide datasets, they also organize competitions in analyzing datasets or improving already invented algorithms. We found a lot of data about to *New York Stock Exchange*. Unfortunately we did not find one single dataset which satisfied all our requirements. There was no data about how often a search term was requested at *Google*. So we had to collect the data ourselves from *Google Trends*. In the following chapters we describe our datasets and why we are using them.

A. Stock data

We combined data about the *New York Stock Exchange* of two datasets from *kaggle*.

1) NYSE: Gawlik (2017) collected information about historical prices of the S&P 500 companies and additional fundamental data in his dataset. Standard and Poor's 500 (S&P 500) is a stock index of the 500 biggest US companies listed on the stock exchange.

The dataset has 4 files:

- **prices.csv** This file contains the daily stock prices from 2010 until the end of 2016. For newer companies the range is shorter. Unfortunately the prices are sorted by day and not stock and therefore we could not use this data without preprocessing it. Instead, we found a better suiting dataset.
- **prices-split-adjusted.csv** Approximately 140 stock splits occurred during that time. This file has the same data with adjustments for the splits. We do not use this data neither.

- **fundamentals.csv** The file summarizes some metrics from the annual SECC 10K fillings. The metrics are useless for us, too.
- **securities.csv** Additional information for each stock can be found in this file. The most important data is the mapping from the ticker symbol to the full name of the companies. We use this data as an input for collecting data from *Google Trends*.

2) S&P500: The second dataset from *kaggle* has also information about the daily stock prices from the S&P 500. Nugent (2017) ordered the prices by stock and are therefore much more useful for us. Each entry consists of the date, four prices, the volume and its ticker symbol. The dataset has the prices for the last five years starting at 2012-08-13 until 2017-08-11. For the weekends there are no entries because the stock market is closed and no trading happens. For our task we use the opening and closing price to calculate if the stock went up or down on one day.

B. Google Trends data

Google provides a public accessible service called Google Trends. It offers many different kinds of information about all search terms typed into the Google search engine. For example, for a given search term, like "Commerzbank", a graph is being displayed that shows the number of searches for this exact term over the course of time. The figure 3 depicts a sample of this graph. Additionally, there is other data like the

Fig. 3. Resulting graph of a google trends Google search for the financial company "Commerzbank", which is tradeable at the stock market.

interest grouped by region, related topics and similar search terms available. Especially the similar search terms could be used to traverse all related Google Trends data starting from one single term like a stock course name. Therefore, the simple and automated extraction of Google Trend data is essential for the success of the proposed deep learning model for stock prediction. Unfortunately, there are some problems regarding the automated retrieval of it. This is explained in detail in the following subsection VI-B1.

1) Hacking Google Trends: The amount of available data from Google Trends is massive. Unfortunately, unlike many other services of Google, it does not provide a public API for retrieving the trend data. Due to the needed quantity of data, manual extraction is not a practicable option. Therefore, an initial analysis of the sent requests from the web browser was executed.

The Google Trends website sends an explore-request to collect a JSON-file containing several tokens and further information. These tokens are used by the Google Trends servers to validate a request from the requesting machine to a specific service. The multiline-request is used to retrieve the time line data of a search term. The data for related topics and similar search terms can be collected by the same URL, whereas the respective token needs to be put as value into the token-parameter to distinguish between those two types of information. Based on this analysis, a python script is developed that gathers automatically the needed Google Trends data starting from the stock names. If too many requests are send to the server, the HTML error 429 may occur. This is being take care of by simply waiting a short period of time between the requests.

However, even the working python script was limited in a way that the Google Trend server only allows a limited number of requests. If that upper bound is reached, the server blocks the requests and redirects to another web address. On this website the user needs to prove that he is not a robot by clicking the parts of a picture that contain a certain object. Although the use of selenium, a framework to automate software tests for web applications, could serve as a possible workaround of this problem, it would imply a huge overhead with uncertain results. Therefore, this option was not further investigated.

The implemented python script for the automated Google Trend data retrieval, as well as the other related work of this paper, is published in the github repository of Engel and Sikeler (2018).

C. Merged dataset

From the three previously described datasets we used only two directly. The NYSE dataset (see VI-A1) was only used to collect data from *Google Trends* (see VI-B). We combine the data of 14 search terms with the data from S&P500 (see VI-A2). We integrated the label in our dataset. The label is calculated from the opening and closing price.

$$y = sign(p_{close} - p_{open}) \tag{8}$$

The label can be found in the first column so that we can easily extract it in our algorithm. We separate our data in two classes. A label of 1 means that the stock went up on this day. The other class is labeled with -1 and says that the price went down. If the value did not change during the day the label will be 0. We interpret it the same as the label -1 because only up going prices are good if you invested in this stock.

The second column contains the opening price we already used to calculate the label. We did not include the closing value because it is the same as the opening price of the following day and could be computed. We include one price so that out network can use it to compare the trend of the stock price with the trends of the search terms. The value may be used to not only predict if the stock goes up or down, but to predict the exact next value.

The columns 3 to 16 contain the values of the search terms. It is important that there are only a few values of zeros. One problem is to find good search terms which are really

correlated to the trend of the stock price. It is easily possible to replace them and test the network with other search terms. This may improve the prediction. We always include the search terms for the ticker symbol and the company name. The remaining twelve search terms depend on them.

Unfortunately it was not possible to collect data from *Google Trends* for all search terms for the last five years. Therefore, we had to discard some of the stock prices and start including them in our dataset as soon as there are enough search terms with a different value than 0. That is why the trends are starting at a different date. Our dataset contains TODO entries. In the followin chapter we describe how and why we preprocess our dataset.

VII. PREPROCESSING

Because we use our own data set (created by our self), we do not need much preprocessing. We write the preprocessed data back to the file storage as python arrays (.nyp-files), so we do not have to do the preprocessing for each new training configuration. The csv-files contain the data as a two-dimensional array but we actually need a three-dimensional array. So we split the original data in blocks of dimension 30×16. If the number of entries in a csv-file could not be divided by 30 without remainder, the remaining entries are skipped.

For each block we apply zero-centering. Zero-centering is the method of subtracting the mean value of a column of each column entry. We do this only for the columns containing the data from *Google Trends*. These values are all positive and therefore the gradient would also contain only positive or negative values, which makes training more difficult. We do not apply zero-centering to the opening price of the stock although we have the same problem with these values. But if we like to predict the new stock price and not only if the stock goes up or down, we need the original, unmodified price.

We have another preprocessing step which is not stored in the python array. The labels -1, 0 and 1 have to be converted in a array of dimension 1×2 as we only have to classes. The labels -1 and 0 are treated as the same class. If the first entry of the array is bigger than the second, the stock price will rise. If it is the other way round the stock price will fall.

VIII. ARCHITECTURE

+ uml of project structure + describe every part of the uml and the connection between the parts +

IX. DEEP LEARNING MODEL

used model (kind of lstm), tensorboard

- logical representation of the model
- lstm/rnn: why?

X. EVALUATION

- describe how the implementation is being evaluated
- execute the evaluation
- describe and interpret the results

XI. CONCLUSION

simplified approach reached xx% of accuracy, therefore it would be worthwhile to enhance the current model by predicting ranges of values, e.g. 50 up or down etc. as described in VIII, only minor changes would only be necessary

- summarize the results (implementation? evaluation results?)
- does it work? why? why not?
- how could the current solution be improved?
- possible further research?

REFERENCES

C. Chong, Eunsuk andHan and F. C. Park. Deep learning networks for stock market analysis and prediction. *Expert Syst. Appl.*, 83(C):187–205, Oct. 2017. ISSN 0957-4174. doi: 10.1016/j.eswa.2017.04.030. URL https://doi.org/10.1016/j.eswa.2017.04.030.

M. Engel and D. Sikeler. Github repository - stock prediction, 2018. https://github.com/engelmi/DLA.

D. Gawlik. New york stock exchange, 2017. URL https://www.kaggle.com/dgawlik/nyse.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Google. Google trends. https://trends.google.com.

C. Nugent. S&p 500 stock data, 2017. URL https://www.kaggle.com/camnugent/sandp500.

M. Schieche. Grundlagen optionspreismodelle, 2007. http://markus-schieche.gmxhome.de/files/Grundlagen_Optionspreismodelle_MRZ2007.pdf.

R. Singh and S. Srivastava. Stock prediction using deep learning. *Multimedia Tools Appl.*, 76(18):18569–18584, Sept. 2017. ISSN 1380-7501. doi: 10.1007/s11042-016-4159-7. URL https://doi.org/10.1007/s11042-016-4159-7.