Stock Prediction using Deep Learning and Google Trends Data

Michael Engel, Daniel Sikeler

TABLE OF CONTENTS

ABSTRACT

Irgendein Abstract... zusammenfassend vll.

I. 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.

II. RELATED WORK

+ stock market description: chaotic, volatile, complex and seemingly unpredictable + result: few research teams try to find a way to predict stock development + the hype of deep learning also triggered the research in the field of stock market prediction, as dl models find application in a huge variety of domains (so why not prediciton of stock prices) + however: older research only hit about 5% accuracy at best, even more discouraging for future work + nevertheless: recent research work used the improving methods of dl by using RNN and RBFNN and therefore the accuracy was improved

As a result of the stock market being chaotic, complex and volatile, there are few published research papers to facilitate the work for this paper.

Nevertheless, the few scientific papers exists which try to predict stock market development.

There are few published research papers in the fields of deep learning and neural networks regarding the prediction of price development. Additionally, most of the available research's results are discouraging. The accuracy of price prediction is quite low with about 5%.

Although the number of publications are few, the

In Singh and Srivastava (2017) the authors use a combination of (2D)2PCA + Deep Neural Network (DNN) and compare this model with the current state of the art (2D)2PCA + RBFNN.

In Chong and Park (2017) the authors have done some shit.

III. BASIC IDEA

google trends to stocks simplified approach: only predict if the stock will increase or decrease prediction of exact values are likely to have 0% success rate

However, one key characteristic of stocks is that they are strongly volatile. Describe concept of volatility: (https://de.wikipedia.org/wiki/Volatilit%C3%A4t) absolute volatility:

$$change = Value_{t1} - Value_{t0} \tag{1}$$

relative volatility:

$$change = \frac{Value_{t1} - Value_{t0}}{Value_{t0}} \tag{2}$$

logarithmic volatility:

$$change = ln(Value_{t1}) - ln(Value_{t0}) = ln(\frac{Value_{t1}}{Value_{t0}}) \tag{3}$$

in 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

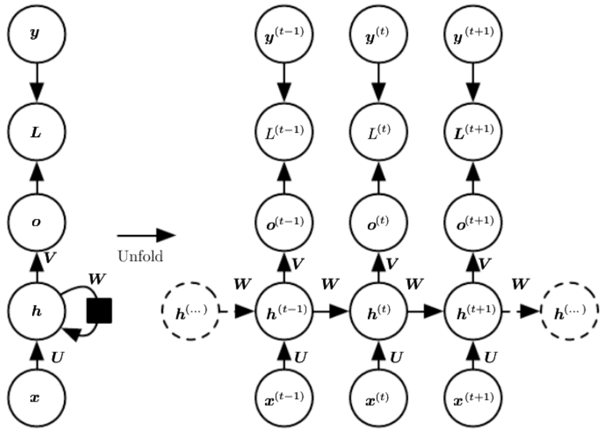
IV. Background

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 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.

Fig. 1. recurrent neural network with hidden-to-hidden connections (Goodfellow et al., 2016, p. 373)



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.

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)} \tag{4}$$

$$h^{(t)} = tanh(a^{(t)}) \tag{5}$$

$$o^{(t)} = c + Vh^{(t)} \tag{6}$$

$$y^{(t)} = softmax(o^{(t)}) \tag{7}$$

"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.

$$y^{(t)} = \alpha y^{(t-1)} + (1-\alpha)v^{(t)} \tag{8}$$

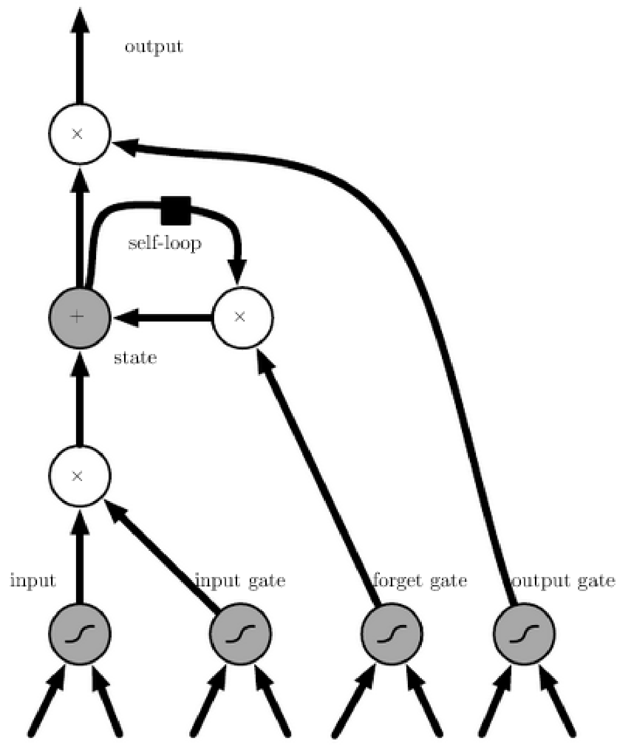
With the parameter α one could regulate how fast the information of the past is discarded. If α is near zero the information is kept for a long time. It would be the best if the network could learn this parameter and must not be set as a hyper parameter.

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

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



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.

V. 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 trends data

1) Hacking Google Trends:

- google trends has no api unlike many other google services
- therefore, an analysis of the request url was done to send prepared requests for data collection
- some additional problems like max. number of requests, request blockade etc.

C. Merged dataset

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

$$y = sign(p_{close} - p_{open}) \tag{9}$$

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. The dataset can be preprocessed. Our preprocessing is described in the following chapter.

VI. Preprocessing

zero centering

- csv format
- zero centering
- ...

VII. Model definition

used model (kind of lstm), tensorboard

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

VIII. Architecture

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

IX. Evaluation

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

X. Conclusion

- 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.

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.

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

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.