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. The final implementation and evaluation results are published in the GitHub repository of Engel and Sikeler (2018).

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 - a so called binary classification. If the simplified model yields promising results, a more complex variation is worth investigating using the implemented model.

V. Background

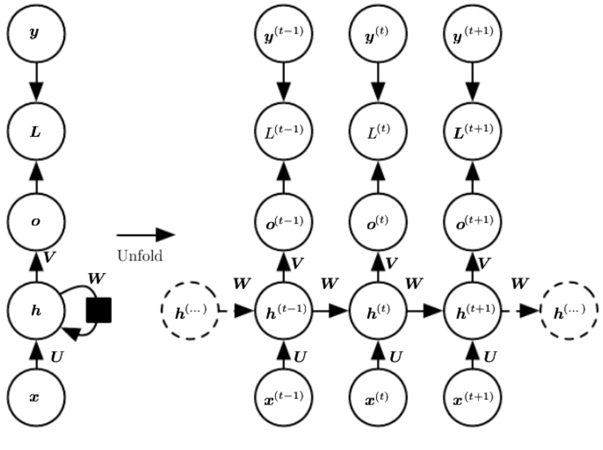
Deep learning architectures are complex and a lot of research is going on. In the next sections we provide some basic background knowledge about recurrent neural network. If further background knowledge about neural networks is necessary, please read Goodfellow et al. (2016).

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

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



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.

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

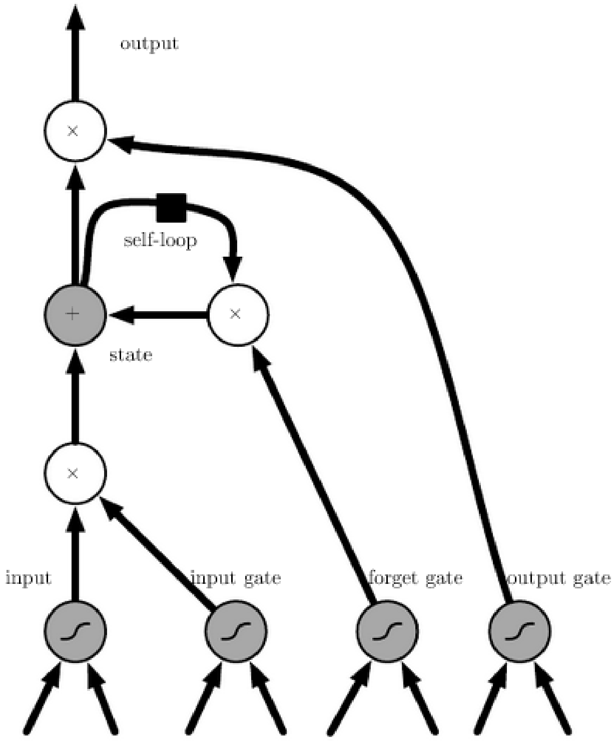
With the parameter α one could regulate how fast the information of the past is discarded. If α is near zero the information is kept for a long time. It would be the best if the network could learn this parameter and must not be set as a hyper parameter.

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)



D. *Dropout*

TODO: Daniel

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:

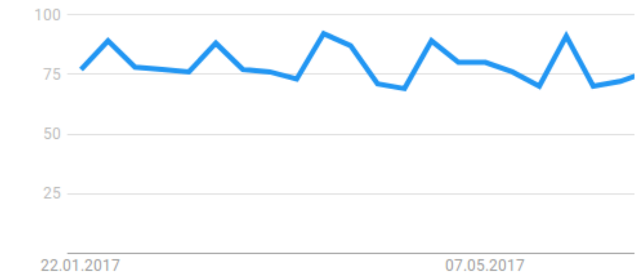
- **prices.csv** This file contains the daily stock prices from 2010 until the end of 2016. For newer companies the range is shorter. Unfortunately the prices are sorted by day and not stock and therefore we could not use this data without preprocessing it. Instead, we found a better suiting dataset.
- **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 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.

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



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 our 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.

For simplicity we split our dataset into two categories.

1) Training and Validation: This dataset contains 209 entries and can be found in *stockprediction/datasets/myDataset*.
2) Test: This dataset is used only for testing and never during training. It is located at *stockprediction/datasets/testData* and has 16 entries which is about 7.66% of the training and validation category.

In the following 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.

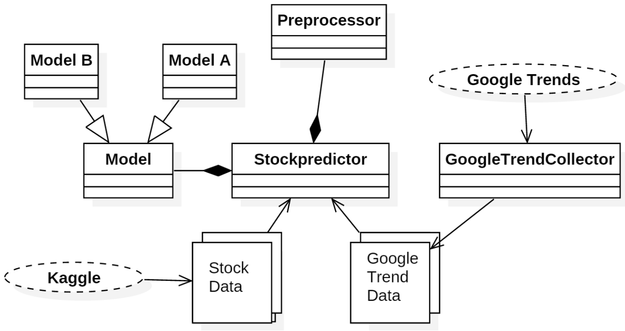
Another common preprocessing task is normalization. During normalization all values are mapped in a given interval. As the data from *Google Trends* already is between 0 and 100 we do not need normalization. For the stock data we do it neither because of the same reason as before.

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

The architecture for this project was designed to be easily expandable for future work. The raw design is shown in figure 4.

Fig. 4. A UML depicting the raw architecture implemented for this project.



The *GoogleTrendCollector* gathers all data regarding the information provided by the Google trends service as described in section VI-B. The stock data, however, is just downloaded from *kaggle*.

Within the *Preprocessor* class functions to read raw stock and Google trends data from csv-files as well as the zero centering are implemented. Additionally, it transforms the raw data into a more useful shape. The functionality to save, read and delete the preprocessed data is also provided by the *Preprocessor* class.

By extracting the learning model into a separate class the architecture becomes flexible regarding the use of different models. This also greatly reduces duplicity of code within the entire project. As there are basic functions used by all kind of learning models, an abstract class *Model* was defined. It provides basic functionality and enforces inheriting classes to implement other necessary methods. For example, the functions of saving and restoring a trained model is used by every kind of learning model, wherefore it is implemented in the abstract parent class. On the other hand, functions like training and predicting may vary between different types of models. Hence, such functions are only defined in the *Model* class and need to be implemented in any inheriting model class. The used model for this project is explained in more detail in section IX.

The *Stockpredictor* class serves as the central part of the whole structure. It uses an object of the *StockpredictorConfig* to configure settings like the learning rate and the number of epochs for the training as well as the used count of folds *k* for the cross validation. An object of the *Model* is also passed as a parameter to the *Stockpredictor* constructor. During the training process this model is being used. Also, an already trained model can be passed to the *Stockpredictor* for further training or prediction. The train function of the *Stockpredictor* partitions all available data. Also, the epoch loop as well as the loop of the k fold cross validation is implemented at this position. The train method of the respective model, however, executes the actual training. This also applies to the predict function. Although the *Stockpredictor* manages all available data, it delegates the preprocessing of raw data to the *Preprocessor* class.

For this project *TensorFlow*, an open-source library for machine learnig, is used. The previously described classes are implemented using the programming language *Python*. The access to *TensorFlow* is therefore facilitated by using its *Python*-API. Packages like *numpy* and *jsonmerge* are used additionally to simplify certain operations. Despite the benefits of using a framework like *Keras*, the native TensorFlow was used.

IX. Learning Model

The development of stock prices can be represented as a time series. The same applies to the Google trend data, where the number of searches for a specific term is used as y-value. These characteristics of both data types as well as the binary classification problem of predicting the rising or falling of the stocks urge the usage of a Long short-term memory network (LSTM). The necessary background for this type of network is explained in V-C. A common issue when training traditional RNN is the exploding and vanishing gradient problem. As LSTMs were developed to deal with those two problems, this advantage is another reason for using a LSTM.

As described in VIII, the model was separated in class. For higher flexibility an abstract base class *LearningModel* was defined. Within this class the abstract function *build_graph* is declared. This method serves as the point of definition for the TensorFlow graph of the model. In IX-A and IX-B the implemented models used for stock prediction are explained.

A. Simple Learning Model

The *SimpleLearningModel* class is derived from the abstract base class *LearningModel* and, therefore, needs to implement the *build_graph* function. A initial check if the graph has already been built prevents a possible exception. The first action when defining the TensorFlow graph is the creation of variables for the input X and the labels Y. Consequent, the variables for the weights and bias are defined and randomly initialized. The *SimpleLearningModel* uses just a single LSTM-cell with a configurable hidden size and forget bias. This simple model also facilitates the use of dropout. If dropout is specified, a *DropoutWrapper* is wrapped around the created LSTM-cell with a fixed dropout rate. The softmax function is used to predict the direction of future stock prices:

$$prediction = softmax(input \cdot weights + bias) \tag{9}$$

The loss is calculated by *build_graph* applying the summary metric *cross entropy* to the unscaled output. This metric was deployed, because the cross entropy should be optimally minimized and the softmax function is used for the prediction. Therefore, numerically unstable corner cases are covered in a mathematically suitable way. For optimization the *GradientDescentOptimizer* is facilitated by the model. Although this simple Optimizer requires more tuning of the learning rate to converge quickly, it does far less computations than the *AdamOptimizer*. For training the minimization operation is applied to the loss.

B. Only Stock Learning Model

In comparison to the *SimpleLearningModel* the *OnlyStockLearningModel* uses far less information. The whole data from *Google Trends* is discarded during runtime. As the *SimpleLearningModel* can be configured very easily, the *OnlyStockLearningModel* is derived from it. The only adaption is that the number of values (columns) used from the dataset is fixed to only the column with the stock data.

X. Evaluation

For evaluation of both models, which are described in section IX, the tool *Tensorboard* is used. This suite of visualization tools can be used to understand, debug and optimize *TensorFlow* programs. In the context of evaluation, *Tensorboard* is used to visualize the

- accuracy,
- loss and
- graph

of the model. As validation technique the k fold cross validation is being deployed. How this technique works and how it is integrated within the architecture is described in the subsection X-A. In X-B the basic setup used for executing different experiments are explained. The specific results of those experiments for the *SimpleLearningModel* are shown in subsection X-C, while the results yielded by the *OnlyStockLearningModel* are described in the subsection X-D. Finally, the results from both models are compared in subsection X-E.

A. K fold cross validation

The most simple method for cross validation is the *holdout* method. This technique splits the available dataset into two distinct sets - training and validation. In terms of variance this method is disadvantageous as it is not certain which data points end up in the validation set. This can lead to losing important patterns during training and, therefore, underfitting. The technique of k fold cross validation provides a method of using the available data efficiently for training as well as validation. The data is split into k subsets and the *holdout* method is applied k times to these sets. In each iteration a different set of the k datasets is used as validation set and the remaining $k-1$ sets are used for training. As a result, each data point is used multiple times for training at some point in the loop and once for validation. Consequently, most of the data is used for fitting so that the bias is greatly reduced and by using most data for validation this also reduces the variance. The swap of training and validation data for each iteration also contributes to the efficiency of this technique.

The evaluation of the model by the k fold cross validation is achieved by logging the accuracy and loss. Those two scalars are measured for every iteration during the cross validation and after the k iterations for the current epoch. During each of the k iterations the mean of the accuracy and loss for all batches is calculated and written to a *Tensorboard* chart. After the k iterations the mean of all previously calculated average accuracies and losses is determined and also written to a *Tensorboard* chart, whereas the x-axis displays the epoch of this result. The calculation for those mean values is done outside the *TensorFlow* graph. Although this is not best practice, the mean calculation for those scalars would have been quite challenging to do inside the graph as a definition for the epoch is not possible at the level of graph definition.

B. Experiments for evaluation

For the validation of the model implementation experiments are executed for both defined models, which are described in detail in subsection IX-A and X-D. The following parameters are varied for a different setup:

- dropout probability,
- hidden size of the LSTM cell,
- learning rate of the optimizer,
- count of folds for cross validation k,
- and the number of epochs

The mean accuracy and loss after each k fold cross validation as well as the mean overall accuracy and loss are being used as metrics for the rating of parameter configuration. These scalars are logged as described in the previous subsection X-A.

Overall, four different settings of those parameters are used, wherefore four experiments for each model type are executed. In table I these configurations are depicted. While the number of folds k for the cross validation as well as the number of epochs for training remain the same for all experiments, the other three parameters are varied. According to a scientific paper about dropout from the university of Toronto (Srivastava et al. (2014)) the minimal classification error is achieved for a fixed dropout probability p in the range $0.3 \leq p \leq 0.8$. Therefore, the experiments using dropout are executed with a minimal dropout probability $p = 0.3$. To examine the effects of the hidden size of the LSTM cell and the learning rate of the optimizer on the accuracy and loss, these parameters are also varied.

	Experiment setups			
	1	**2**	**3**	**4**
Hidden size	100	100	200	100
Dropout probability	0.3	0.0	0.3	0.3
Learning rate	1.0	1.0	2.0	0.5
Number of folds k	10	10	10	10
Number of epochs	650	650	650	650

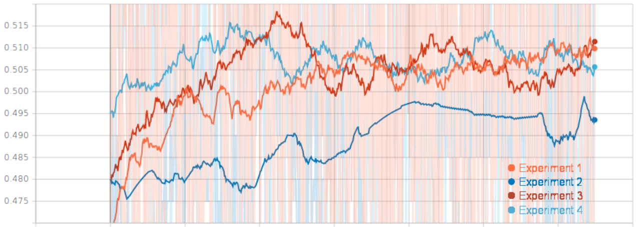
TABLE I
THE CONFIGURATIONS FOR ALL EXPERIMENTS.

Additionally, the trained model is used to predict the rising or falling of stocks unused during the training process. This prediction process is executed ten times and the maximum, minimum and average accuracy for the predictions are logged. Thus conclusions can be made about the applicability of the models for stock prediction.

C. SimpleLearningModel results

The visualized experiment results of the *SimpleLearningModel* are extracted from *Tensorboard*. Within the visualization suite the accuracy and the loss of all experiments were grouped within one graph for better comparability. In figure 5 the accuracy over 650 epochs are depicted.

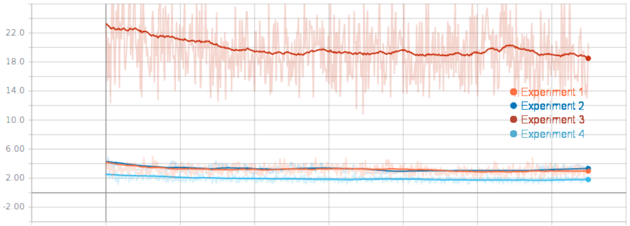
Fig. 5. The mean accuracy of all experiments over 650 epochs of the k fold cross validation with $k = 10$.



As expected, the experiment without dropout achieved the lowest accuracy of all experiments. While the other three experiments achieved quite similar results concerning the accuracy, experiment 2 needed about twice the time for the same number of epochs than experiment 1 and 4 with about 22 minutes. This is due to the doubled size of the hidden units, wherefore this setting is unsuitable for training and, also, strongly influences the extent of the loss which can be seen in figure 6.

Therefore, only experiment 1 and 4 yield acceptable results. In comparison, the accuracy graph of experiment 4 is overall higher than the accuracy graph of experiment 1. This implies that the configuration of experiment 4 is better than the settings of experiment 1. However, regarding the prediction results for a dataset unknown to the trained models, experiment 1 seems to work better. In table II these prediction results are presented.

Fig. 6. The mean loss of all experiments over 650 epochs of the k fold cross validation with $k = 10$.

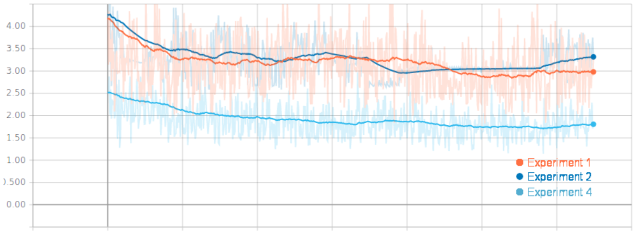


	Experiment prediction results			
	1	**2**	**3**	**4**
Minimum	50.00	43.75	43.75	31.25
Maximum	68.75	43.75	68.75	68.75
Average	61.25	43.75	50.00	51.25

TABLE II
THE MINIMUM, MAXIMUM AND AVERAGE PREDICTION ACCURACY IN % OF TEN PREDICTIONS FOR A DATASET UNKNOWN TO THE TRAINED *SimpleLearningModel*.

As described in the previous section X-B, these results are collected by using a dataset unknown to the trained models and executing the prediction function ten times. For this validation the settings of experiment 1 achieve a 10% higher accuracy than the model of experiment 4. The smaller learning rate of experiment 4 in combination with the presented results could imply that this model needs more epochs to converge to the optimum. However, the losses of all experiments quickly converge to a minimum. In figure 7 the graph of the third experiment is removed due to its magnitude so that a more precise view of the remaining loss graphs can be seen.

Fig. 7. The mean loss of the experiments 1, 2 and 4 over 650 epochs of the k fold cross validation with $k = 10$.



D. *OnlyStockLearningModel results*

In figure 8 you can see the accuracy of the four experiments described in table I. The loss can be seen in figure 9. The model without using dropout (experiment 2) performed best. This seems to be reasonable because if the dropout drops the only input, like in the other three experiments, the model can not use any data for learning. The model with the biggest hidden size (experiment 3) is not performing better than the other models and therefore it does not make sense to increase the hidden size. The model of experiment 4 has the lowest learning rate and therefore it should be possible for it to find a better minimum. But as the accuracy shows there is a great variance and the accuracy is not converging against any value.

Fig. 8. A Graph depicting the mean accuracy of the OnlyStockLearningModel over 650 epochs of the k fold cross validation with $k = 10$. The orange line shows the accuracy during training for experiment 1. Its mean accuracy is about 49%. The dark blue graph is presenting the results of the second experiment. It has the best mean accuracy of about 52%. The red line showing experiment 3 has about the same mean accuracy as the forth experiment (light blue line). The mean accuracy is at about 48% accuracy.

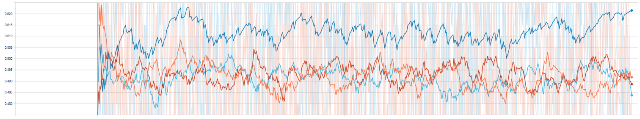
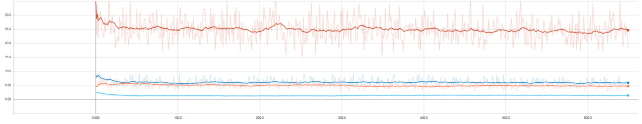


Fig. 9. A Graph depicting the mean loss of the OnlyStockLearningModel over 650 epochs of the k fold cross validation with $k = 10$. The orange line depicts the loss of the first experiment. It has nearly the same mean loss of about 4.7 as the second experiment (dark blue line). Experiment 3 has a much higher loss, depicted with the red line, of about 24.6. The loss of experiment 4 is shown in the light blue line. Its mean value is the lowest with about 1.4.



In table III you can see how the models performed on data not seen during training. The test dataset contains only 16 entries, so the results have to be watched carefully. The model without using dropout (experiment 2) has a lower accuracy on unseen data than it had during training, whereas the model of the first experiment is even better on this test data. The forth model has like during training the biggest variance and is performing equally well as the second model. The third model, although it has the biggest hidden size and can store the most information, has the lowest accuracy on unseen data. All four models do not have an accuracy during training and testing which is good enough to make me belief that they can predict the trend of a stock better than an financial expert could do.

	Experiment prediction results			
	1	**2**	**3**	**4**
Minimum	50.00	50.00	37.50	37.50
Maximum	56.25	50.00	56,25	62.50
Average	53.63	50.00	46.88	50.00

TABLE III
THE MINIMUM, MAXIMUM AND AVERAGE PREDICTION ACCURACY IN % OF TEN PREDICTIONS FOR A DATASET UNKNOWN TO THE TRAINED *OnlyStockLearningModel*.

E. *Comparison of model results*

Comparing the results of the SimpleLearningModel X-C with the results of the OnlyStockLearningModel X-D can help us to decide whether the data from google trends is useful in predicting the stock course.

First of all we want to compare the differences between the two models according to the changes made to the configuration. As already mentioned in subsection X-D dropout does not help the OnlyStockLearningModel, whereas the SimpleLearningModel improves with using dropout. Varying the hidden size of the LSTM cell does not lead to an improvement neither in the SimpleLearningModel nor in the OnlyStockLearningModel. The same can be said about the reduction of the learning rate.

During training the second experiment with the OnlyStockLearningModel had the best accuracy, but on unseen data it is slightly worse than nearly all experiments (expect experiment 2) with the SimpleLearningModel. On unseen data the first experiment of the SimpleLearningModel has by far the best accuracy with 61.25%. But this has to be watched carefully because of the small number of samples in the test dataset.

There is only a small variance of the accuracy in all experiments during training. The loss is also changing very slowly after about epoch 150. In contrast the variance during testing is really high especially for the forth experiment of both models. If we consider the results of the first experiment of both models as a spike because they have the best average accuracy on unseen data, the three other experiments leaded to nearly the same results for both models. As a consequence the data from google trend does not really help to predict the stock course.

XI. Conclusion

In this paper + results of the project to predict rising or falling stocks using deep learning methods are presented + the basic framework for - retrieving the google trends data - preprocess the data - learning models implemented + simple learning model, which uses a lstm, achieved an accuracy of 53% , just above simple guessing + after ca. 150 epochs with k fold cross validation (k=10) and dropout the accuracy reached its max - longer training was contra productive

+ simplified approach therefore does not encourage further research of predicting stocks based on google trend data + however as only the lstm, and only a single cell, was used, other models and model combinations could yield better results + as the architecture for this project was implemented flexible for future extensions, it could serve as a starting point / core for this future research. for example, another model could be derived from the abstract class and used instead

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?
- how fast must it be. Finance is fast changing market

XII. Lessons Learned

A. Daniel Sikeler

The idea of predicting the trend of a stock price is very interesting because I did some investment in stocks and could use it myself. Unfortunately it was difficult to get the data we needed. It took us a long time to get the scripts running for data collection from *google trends* and then we ran into the problem of limit number of requests. So in the end we had to collect data every day. Additionally, we often got data from *google trends* with a lot of zeros and so the data was unfeasible for us. We had to review the data and remove the keywords with useless data.

Deep learning is complex and needs a lot of background information to understand what is going on during training a model and how to improve the performance and accuracy of the model. At the beginning it was a lot of "try and error" because tensorflow further increased the complexity. The tensorboard documentation is so rich on information that you can spend a lot of time there if you do not know exactly what you are looking for. But tensorboard also has some advantages as it hides a lot of the internals of an LSTM cell. If you are ready building the model there is still a lot to do. The training process is very time consuming. You need a lot of data, and in my opinion we did not have enough, to achieve good results with the trained model. We had to decide if we spent more time on collecting further data or try to improve our model. We went for understanding our model and try to improve it. We trained the model with different learning rates, numbers of hidden layers in one LTSM cell. We also tried some other activation and loss function which did not really improve the model.

Evaluating the model is not simple. The accuracy and loss can be easily logged but if there is a lot of variation it is hard to say how to improve the training. For me it is useful to understand what is going on inside the model, but it is nearly impossible to say what the weights are representing. And so we can not really say for what the model is looking for.

Summarizing my experience I can say that it was a interesting task but I should have spent more time on tensorboard and collecting data which was not possible because of other classes. I like to keep myself busy on the topic of deep learning and machine learning in general.

B. Michael Engel

Predicting the development of stock prices is an interesting research topic as stocks are still considered to be unpredictable. The fact that stock prices are greatly influenced by the actions of the traders led to the basic idea of using user generated data and feed it to a deep learning model. Although the binary classification was solved just about 50% of the time, this result could have been worse. The approach of using user generated data for stock prediction could therefore yield quite promising results if aspects like the quality and quantity of data are enhanced.

Unfortunately, there were a few unexpected obstacles that made the complex task of developing a deep learning model for stock prediction even harder. The non-existing API for the Google trend service entailed an time consuming analysis of the http request send by this website to retrieve the needed information. Also, the used machine learning library TensorFlow is comprehensive and initially overwhelming for a beginner in the field of deep learning. The correct shape of data across the multiple operations and their correlations caused some trouble. However, after this initial training many features of TensorFlow, like the DropoutWrapper, could be used quite easily. The use of a framework like Keras would have accelerated the whole development process nevertheless. The efficient use of data as well as the interpretation of any results was unexpectedly difficult concerning the improvement of the prediction results.

Overall, deep learning is a fascinating research topic. It's versatility and applicability in a multitude of domains contributes much to this fascination. Additionally, it is delightful to read scientific papers which were written in the last few years instead of the last century. During the task of using deep learning to predict the allegedly unpredictable stock prices the essentials of deep learning were learned and the desire to do more projects in this domain has been awakened.

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.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2627435.2670313.