STOCK PRICE PREDICTION WITH VERY SIMPLY DESIGNED LSTM BASED NEURAL NETWORKS

~

A CRITICAL VIEW AND VALIDATION

Seminar Paper

by:

ALEXANDER PRINZ

PHILIPP NOSTITZ

Hochschule Anhalt

Anhalt University of Applied Sciences

Abstract

This paper is about very simply designed Long Short-Term Memory based Neural Networks for stock price prediction. Those can be found in very similar wise in some publications of various authors. We take a critical view on these works regarding the manner of their models and the validation. Thereby, we use two different model approaches with similar model architectures to validate the model validation concepts of those authors. We can show that the manner of model validation used by these authors is not meaningful to evaluate the model goodness. In addition, we show a more statistically representative validation concept and that, at least, those very simple LSTM based models are not work as reliable predictors.

Kurzfassung

Dieser Artikel befasst sich mit sehr einfach gestalteten LSTM basierten neuronalen Netzen zur Vorhersage von Aktienpreisen wie sie in sehr ähnlicher Weise in Publikationen verschiedener Autoren zu finden sind. Wir untersuchen diese Arbeiten kritisch hinsichtlich deren Modell und Modelvalidierung. Hierzu nutzen wir zwei Ansätze mit ähnlichem Aufbau der Modelle wie sie von den anderen Autoren genutzt werden um deren Vorgehen bei der Modelvalidierung zu bewerten. Wir können zeigen, dass das Vorgehen der anderen Autoren nicht sinnvoll ist um die Modellgüte zu bemessen. Zudem zeigen wir eine repräsentativere statistische Methode der Modelvalidierung und mit ihr, dass zumindest solche sehr einfach designte LSTM basierte Modelle nicht für eine verlässliche Prognose von Aktienpreisen geeignet sind.

Content

List of Figures

List of Tables

Symbols | Abbreviations |Acronyms

acc	Accuracy
acc_{sig}	Significant accuracy level
D	Timeseries Data of the Stockprice
$f(\cdot)$	Function of ·
h^t	State function of hidden unit
I	Index
λ	Eigenvalue
M	Timeseries of the predicted Stockprices
MSE	Mean Squared Error
μ	Value of expection
n	Number of predictions
o	Output
p	Value of possibility
σ	Standarddiviation
θ	Shared parameter state function hidden unit
$trend^i{}_{real}$	i'th real trend value
$trend^i{}_{pred}$	i-th predicted trend value
V	Matrix as shared parameter from hidden unit to output
w_i	i-th value of weights
$\vec{w}$	Weight Vector
W	Matrix as shared parameter
W^t	Matrix as shared parameter at time step t
x^t	External input at time step t
x_i	i-th values of an input vector
$\vec{x}$	Input vector
$\|\cdot\|$	Absolut Value of ·
z	z-Value for z-Transformation

Introduction

The ever faster and better becoming Internet makes stock trading very well possible for almost everyone interested. In addition, it offers by the numerous information platforms an enormously large source at information which can relate to the value of the stocks.

In particular, the CFD Trading has found in the last years a large spreading, which are surely also strongly connected with the possibilities of the Internet. CFD trading is based on betting on stock price behavior in relatively short periods of time, for example within a few hours or even seconds. It is therefore obvious that in the age of machine learning and information floods, algorithms are used to learn to predict the temporal price changes of shares and to automate stock trading.

Our study deals with a simple model based on a Long Short Term Memory Network (LSTM), as can currently be found in works of some authors, published on numerous platforms such as "Towards Data Science" or private blogs etc., but also platforms such as university publishing servers. It is about a critical examination of the methodology as well as result interpretation and presentation of these authors.

In addition, we use two different methods to test two long term memory based neural networks for their suitability as predictive models. We investigate a model that is supposed to predict CFD trades in relatively long-time ranges and a model that expresses a very short time range. While the long-term model will be validated visually, we will examine the short-term model in more detail using a suitable statistical method.

In this study, we can show that supposedly appropriate metrics and visualizations, as used by authors to verify their models, are not suitable and may even mislead readers.

Furthermore, we can show that these simple LSTM based neural networks have no significant predictive power, neither in a long-term modeling of stock price behavior, nor in a short term one.

The idea behind stocks and their trading dates back to 1288, the year in which the stock was first mentioned in a document (DUTHEL, 2013). A stock is a share of a company. The goal is to invest in a company to benefit from its profits proportionally.

Over a long period of time (months to years) the value of a stock should increase. Of course, this presupposes a successful management of the company. One speaks of a positive correlation between growth and the associated indicators (GOLDSMITH, 1969; MCKINNON, 1973). If a company has established itself in the market and is making progressive profits, an investment in such a stock is very likely to be profitable for the stock owner. The profit that can be generated from the company shares (dividends) as well as the stock value itself, however, increases only relatively slowly. Under certain circumstances (high share price, low growth rate) the ratio between investment and profit is not attractive enough in terms of the long time. A good managed company generates a long-term positive trend regarding the stock value. On small time scales, however, the stock value behaves very variably. It does not run constantly positive, but also loses value for a short time. This is called volatility (BOLLERSLEV & MIKKELSEN, 1996).

The following figure shows the stock price course chart of APPLE about a particular time range. As can be seen the trend of the stock price in a long-term view is increasing, but in short time ranges decreasing time ranges may occur.

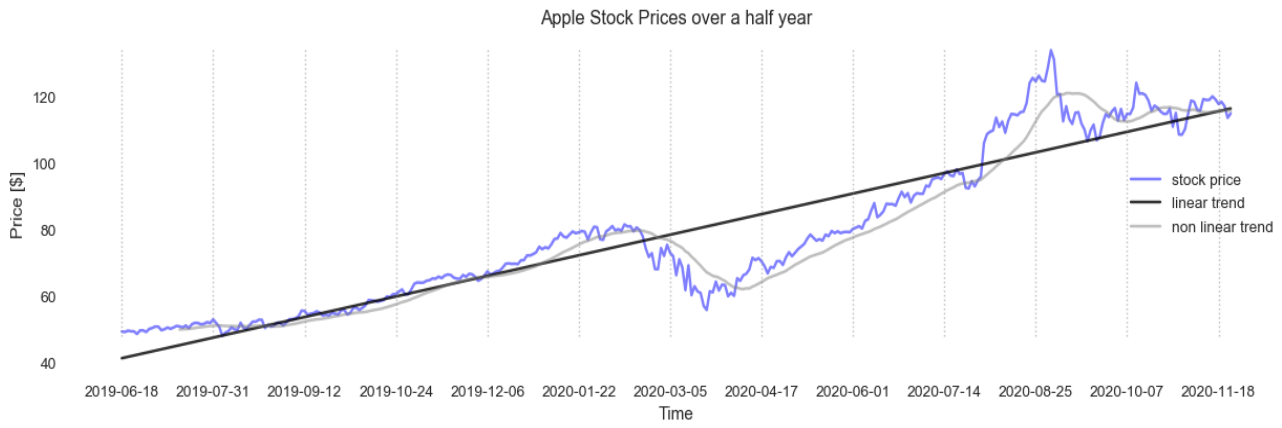

Fig. 1: ***Apple stock price behavior over a half year.*** The blue curve shows the raw data. It can be seen that the course is increasing in a wide range time scope, but there are time ranges, were the course is decreasing. The black curve shows the linear regression oft the price data as a positive trend. The grey curve is a rolling mean with a window size of 10 timesteps. It emphasizes the positive trend but also the volatility.

There are, however, financial methods that can lead to higher profits over the investment period, but also to possible losses. These methods take advantage of volatility and allow transactions to be made between a broker and the traders. The broker is a kind of middleman, who owns and can provide high amounts of capital. The trader approaches him and borrows stock shares from the broker. There are some options. On the one hand side, the trader can buy for his trade a stock share value, that is 1:1 corresponding to the value of the money he spent. But it is also possible -if the broker offers such option- to buy higher shares of stocks with another ratio between money spent and the stock share. One speaks of leveraged transactions, because a small bet of real money will be traded for a stock share with a higher nominal value. This can lead in high wins, but also in high losses, if the trade goes against traders bet. CFD trading also allows a short-term trading of the share, whereby it is even possible to bet on the loss of a stock in the predefined time of trading. These points make CFD trading very attractive for many. In addition, there are now apps from many brokers that make CFD trading easy and convenient "from the couch".

The principle is to define a period of trading and bet on a movement of the stock (value increases, value decreases) within this period. If the desired result is achieved, the difference in value of the stock minus some costs is credited to the trader's account. In case of opposite results, the trader loses all his bet. Dependent on the value of the used lever, the loss can be very high, but also the wins. It depends on traders' "sense" if he is successful in his work or not. In the following chapter, we discuss the meaning of that "sense" in the context of time series patterns of stock price courses.

Stock Price Prediction and Time Series Pattern

Due to the volatility and the principle of CFD trading, it is possible to use short-term share price fluctuations as a basis for betting. It can be bet on falling as well as rising prices in the defined time range of trading. So, it seems even more interesting whether there are ways to predict these fluctuations. Share prices are not subject to chance but are the function of a very large number of parameters. Just a press release about a company with negative news can cause its share price to drop for a short time.

The reason for this is that stock owners who got the message and then sell their stock shares in fear of losing money. But possible seasonal events may change stock prices, too. A company that sells gas will have more sales in colder times of the year than in warmer times. The sales may map directly to the share prices. In the colder times of the year, there may be a slightly larger increase in stock value.

Many traders are convinced that there are patterns in stock price movements and that these patterns, if recognizable, can be used as an aid to predict stock prices.

The following figure shows the time period of APPLE's stock performance in the form of a so-called candle plot of a trading day. The candles contain the closing prices of the respective trading period. The color codes whether the closing price has fallen (red) or risen (green) compared to the last one.

In addition, so-called Bollinger bands are seen as blue hatching. These Bollinger bands represent the second standard deviation of all stock prices within a rolling window (in this case with a window size of 30 timesteps) relatively to the corresponding rolling Average.

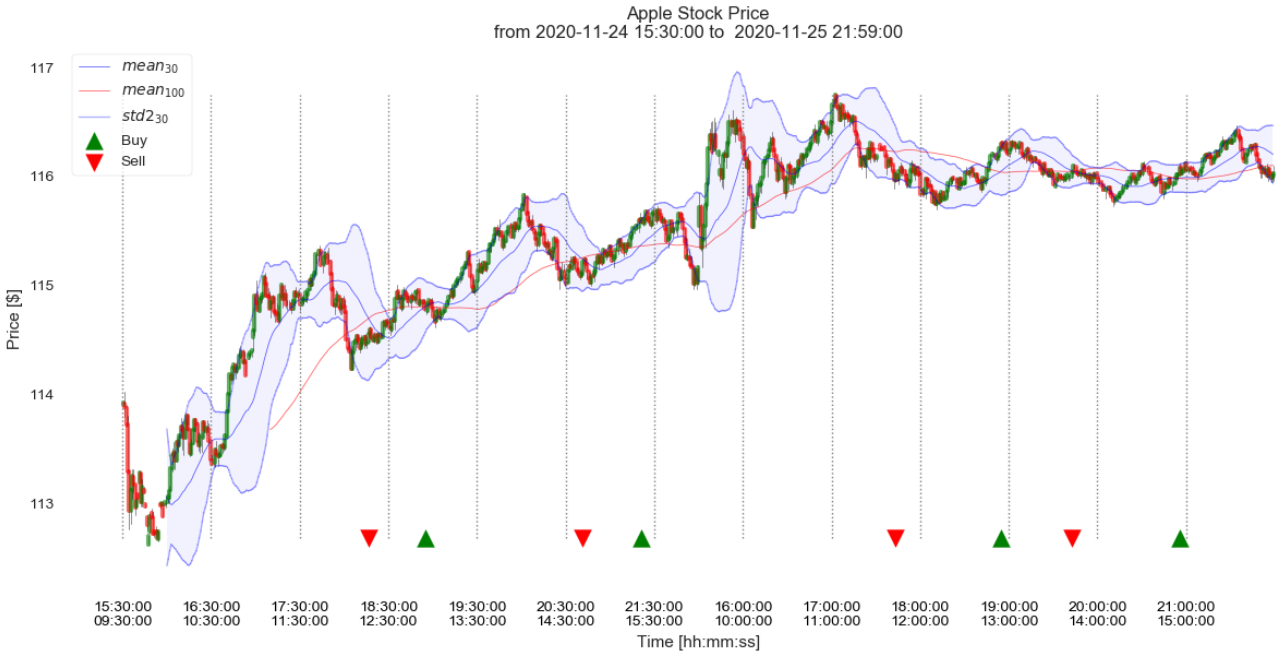

Fig. 2: ***Candlestick Plot of an Apple stock price course over a whole trading day and some containing patterns***. The red and green colored bars are called candles and represent the closing price as well the relation to the last before closing price at the related time step (in this case 5 min). The closing price is represented by an end of the candle. The color indicates whether the closing price of the last before timestep related stock price was higher (red candle) or smaller (green candle) than the currently one. The end of those candles is dependent on their color. A green candle has its end upside and red candles downside. The red curve represents a rolling mean with a window size of 100 timestep related stock prices. The blue curve is also a rolling mean with a window size of 30. The blue hatch represents the area of the so-called Bollinger band. Those bands were defined by the upper and lower second standard deviation of the stock prices within a rolling window. It can be seen, that in some cases, where the band is pierced by a candle, the course is inverted. In cases where both curves of the rolling means are crossed, there is also a stock price change to observe.

A common pattern is that in many points, where the Bollinger bands are pierced by a candle, the stock price trend for the next time steps is inverted at this time step. Thus, when a candle pierces the band, this could be an indication of a price correction and serve as a basis for speculation on it. This pattern seems to come true surprisingly often. However, if one assumes that this pattern is known to many traders, it should not be astonishing. In the end, it is a self-reinforcing effect. When many traders see that the Bollinger-band will be pierced soon, they will react corresponding, which leads in this effect itself.

Another pattern is the behavior of two rolling averages with different window sizes. In the case of this model, there is a rolling average with the window size 30 as a blue curve. And there is a rolling average with the window size 100 as a red curve. If both curves intersect and the red curve dips below the blue one, this is supposed to indicate a price increase (indicated by green triangles in the figure). If the red curve crosses the blue one, the price should fall instead (indicated by red triangles in the figure). This pattern also seems to work quite often.

There are several other patterns where for example price corrections and their distances regarding their stock price can be represented by Fibonacci numbers. But we do not want to go deeper in this. However, those and other patterns can serve as a basis for prediction. Therefore, it should even be possible to let algorithm learn such patterns to build prediction models based on these algorithms.

Artificial Neural Networks

The central unit of neural networks and therefor for Long Short-Term-Memory Recurrent Neural networks are perceptrons. Neural networks provide a robust approach to approximating real-valued, discrete-valued and vector-valued target functions. Over the course of time, they stand out to be one of the most effective learning methods. Neural networks are loosely motivated by biological neural systems like the human brain.

However, neural networks consist of an arbitrary number of connected perceptrons. A perceptron inputs a real-valued vector and calculate a linear combination of it. If this result is greater than a given threshold θ the perceptron returns 1, otherwise it returns -1.

$$o(x_1, \dots, x_n) = \begin{cases} 1 \; if \; w_0 + w_1 x_1 + \cdots + \; w_n x_n > 0 \\ -1 \; else \end{cases} \tag{1}$$

In the equation above w_i is a real-valued constant setting the contribution of an input x_i to the output. For simplification matters the threshold θ will referred to as w_0 so that it multiplied with factor x_0=1 to write a shorter decision rule:

$$\sum_{i=0}^{n} w_i x_i > 0 \tag{2}$$

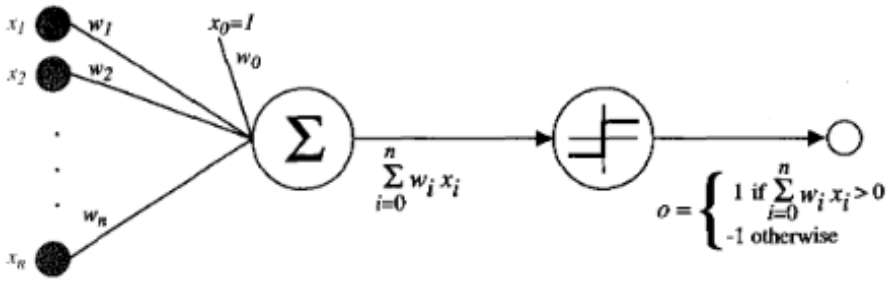

Fig. 3: *Scheme of a perceptron. Source: http://profsite.um.ac.ir/~monsefi/machine-learning/pdf/Machine-Learning-Tom-Mitchell.pdf*

To illustrate the matter geometrically, a perceptron is providing a decision surface in n-dimensional space. The output is 1 if the linear combination of its inputs is on one side of the plane and -1 if it is on the other side. The equation for this hyperplane is as follows:

$$\vec{w} * \vec{x} = 0 \tag{3}$$

This requires a set of training examples that are linearly separable for a single perceptron.

Training a perceptron adjusts the input weights according to the set of training examples. But this also means that the hypothesis space, the space where all possible target functions are in, is a set of all possible real-values weight factors.

The limitations of perceptrons are that they are only applicable to training set that are linearly separable. If the separation hyperplane becomes more complex a single perceptron is not capable of its representation.

For these kinds of applications Multilayer-Perceptrons (MLP) are used. They are also known as neural networks. MLPs can represent a rich variety of non-linear decision surfaces. The standard architecture consists of 3 parts. First the input layer where all the training data is inserted. The second part consists of an arbitrary number of hidden layers. The neural networks end with the output layer as the last part. Every single part of the neural network consists of neurons.

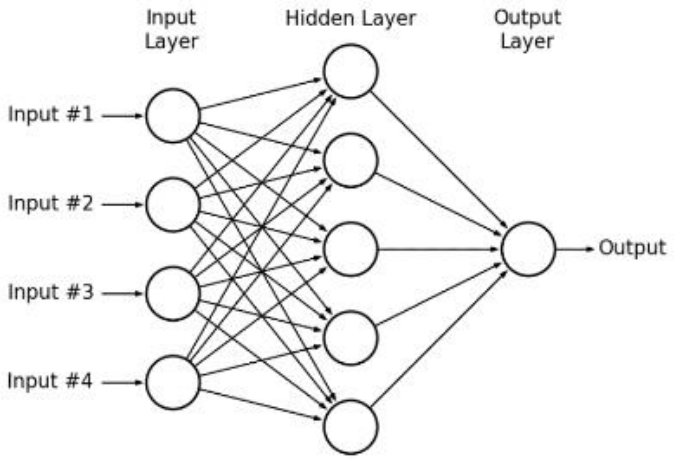

Fig. 4: *Scheme of a neural network*. Consists of three parts. First: input layer, second arbitrary number of hidden layers, last: output layer. Each part itself consists of an arbitrary number of perceptrons. Can represent a rich variety of complex functions. Source: https://en.wikipedia.org/wiki/Artificial_neural_network#/media/File:Artificial_neural_network.svg

If all outputs of the layers above serve as the input of the subsequent layer, the networks are called dense neural network. However, there are use cases where these dense connections are decomposed in favour of more selective connections. Every single connection of this network has its own weight w_i associated with it. Bespoken networks are acyclic.

As the structure of the neural networks become more complex so does the training rule. The most conventional rule is called the backpropagation. This algorithm is based on the gradient descent. The idea behind this is that the hypothesis space of possible weight vectors to find weights that best suit the training sets. Backpropagation lets the calculated error term travel backwards through the network. Therefor in the first step the error term for the output layer is calculated. Secondly, the error term for every hidden layer is calculated taking the results of the first step into account. In the last step every weight is updated. The main goal of the networks is to generalize over the given training sets (MITCHELL, 1997).

Recurrent Neural Networks

Recurrent neural networks are a subset of neural networks. They are specialized on processing sequential data. They are capable of handling longer sequences than neural networks without this kind of specialization. To understand what is necessary to turn a conventional neural network into a RNN the mechanism of parameter sharing must be explained beforehand. In the networks mentioned above every connection between layer is associated with a certain weight. But with parameter sharing every element that is calculated along a path in the ANN is calculated with the same update rule. It allows our trained model to enhance, apply and generalize to training examples of varying forms. In this case the varying form refers to its possible input lengths.

Without parameter sharing this kind of generalization is not possible. In addition, no statistical verifiable statement can be made when the information of interest can be found on different positions of the input sequence. Note that parameter sharing is of particular interest when the important information may appear on several positions in the input sequence.

In recurrent neural networks every element of the output is a function of the previous elements output. This means that every element is calculated with the same update rule as the

element before. It is common practise to train a RNN with mini batches. These batches are assembled in the manner that the input sequences do not have the same length.

The graphical representation of recurrent neural networks captures the concept of the computational graph and enriches it with cycles. Note that in multi-layer perceptron cycles are avoided. Cycles represent the influence of the current value of a variable on its own values in a later point in time. These kinds of graphs allow the definition of RNNs. Computational graphs allow the formalization of the structure of a succession of calculations. So, association between inputs and parameters on the one hand and between outputs and losses can be made.

Another concept that is relevant for the understanding of recurrent neural networks is the unfolding of the computational graph. This results in a change of structure from a recurrent to a repetitive structure. With this unfolding parameter sharing takes place over a deep network structure.

Most of RNN use the following equation to describe the state of their hidden units:

$$h^t = f(h^{t-1}, x^t, \Theta) \tag{4}$$

Where h^{t-1} describes the state of the previous hidden unit, x^t describes the input in the current step and θ describes the shared parameters. Some recurrent neural networks add another architectural property like outputs for every calculation step. This means that if a model is trained to predict the future with respect to what happened in the past, h is a lossy summary of the task relevant aspects of past input sequences. This summary is lossy by default as it tries to map an input sequence of variable length onto a vector with a fixed length. Dependent on the training criteria some aspects are stored more precisely than others.

With these two concepts clarified 2 main design patterns can be described as shown in figure 5 & 6.

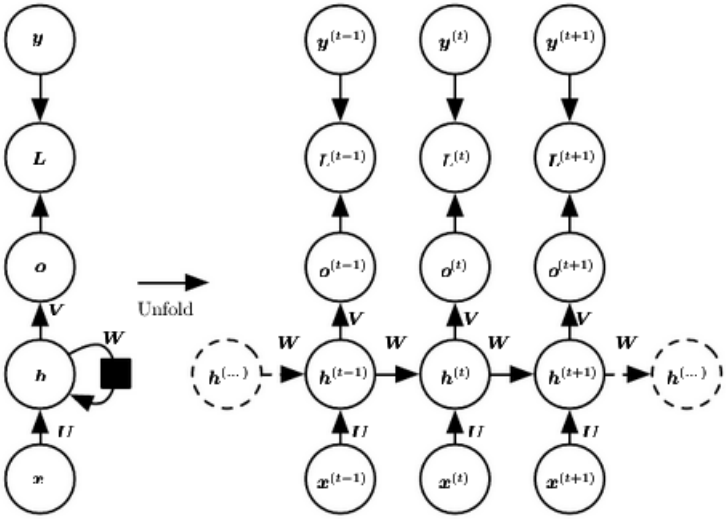

Fig. 5: *Design pattern 1 of a Recurrent Neural Network*. Recurrence takes place between hidden units. With W as the shared parameter matrix between hidden units, U as the shared parameter matrix between input and hidden unit and V as shared parameter matrix from hidden units to output units. Most powerful RNN but very costly to train. Source: https://www.deeplearningbook.org/contents/rnn.html

The first design pattern is shown in figure 5. It consists of recurrent connected hidden layers h, external input units x and output units o. With W describing the shared parameter matrices between hidden layers, U describing the shared parameter matrices for connections from input to hidden units and V as shared parameter matrices for connections from hidden units to output units.

These kinds of recurrent neural networks are the most powerful. They can learn every function that can be described by a turing machine. The gradient descend for these networks is very costly because it is calculated by a forward propagation through the net followed by a backwards propagation. Due to the sequential order of those two calculations the gradient descend is not parallelizable. This means that the computational power this design patterns offer comes with a high cost.

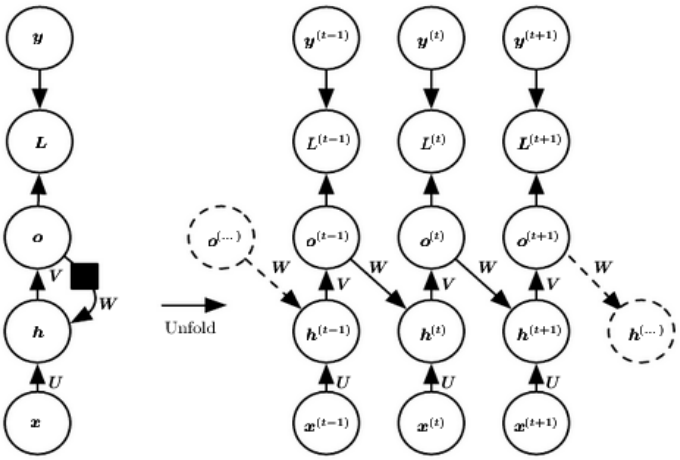

Fig. 6: *Design pattern 2 of a Recurrent Neural Network*. Recurrence takes place between hidden units. With W as the shared parameter matrix between hidden units, U as the shared parameter matrix between input and hidden unit and V as shared parameter matrix from hidden units to output units. Not as powerful as pattern 1 but way easier to train because training can be parallelized. Source: https://www.deeplearningbook.org/contents/rnn.html

The second quite common design pattern is shown in the figure x. In this model the recurrence connects the output unit with the hidden unit of the subsequent layer. With W describing the shared parameter matrices for connection from output units of one layer to the hidden units of the next layer, V describing the shared parameter matrices from the hidden units to the output units in the same layer and U describing the shared parameter matrices from external input units to the hidden units of the same layer. These kinds of recurrent neural network are not as powerful as the first pattern shown because of the missing connections between the hidden units. This means that die output units must capture all information from the already seen in the past. But it is very unlikely that they are capable of it. But the advantage of the avoided connection between the hidden units is that every loss-function that compares a prediction with the according training goal is isolated. On one hand this means that the training of these network is parallelizable and on the other hand this leads to a gradient that can be calculated separately for every step in the net. The training method used for this is called teacher forcing. In this method the model gets the known training label $y^{(t)}$ as input for the timestep t+1. This avoids the backpropagation through time algorithm which makes it less time consuming. The drawback to this method is that the trained model cannot be used in the open-loop-mode. The open-loop-mode is using the output $o^{(t)}$ as an input in timestep t+1. The problem with this is that the trained model may encounter sequence forms it never met while training. One way to avoid this problem is to randomly choose between the known values and the calculated values of the hidden units (GOODFELLOW, 2016).

The problem with recurrent neural networks in general is that they fail to learn in presence of time lags greater than 5 – 10 (GERS ET AL., 2000). This problem can be illustrated by a simple thought experiment. Given a computational graph that implies a multiplication with some random matrix W. After t timesteps this corresponds to a multiplication with M^t. Assuming M has an eigenvalue decomposition of:

$$W = (V\ diag(\lambda)V^{-1}) \tag{5}$$

It is clear to see that:

$$W^t = (V\ diag(\lambda)V^{-1})^t = V\ diag(\lambda)^t V^{-1} \tag{6}$$

All eigenvalues that are not near an absolute value of 1 will either explode (when greater than 1) or vanish (when smaller than 1). This problem is called the vanishing and exploding gradient problem. Recurrent neural networks use the same matrix in every time step. Long Short-Term-Memory-cells are based on the idea of creating paths in time where the gradient neither vanishes nor explodes. Therefore, connections weights are introduced that can change in every time step. LSTM-cells accumulate information over the course of time and learn when to forget them e.g., when it is 'used' or out of date. To realize this idea LSTM-cells introduce the concept of self-loops. Self-loops create the bespoken paths on which the gradient run for a long time without vanishing or exploding (GOODFELLOW, 2016).

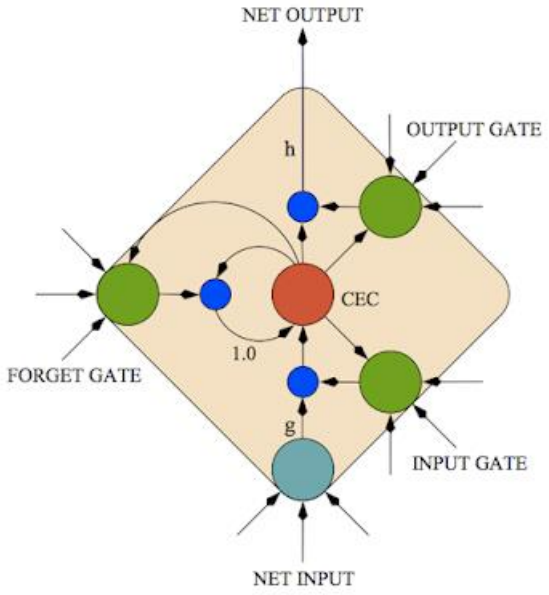

Fig. 7: *Scheme of a Long Short-Term-Memory-cell.* Based on Gated units. Replaces hidden units in RNNs. Information flow can be controlled by input and output gates. These keep noisy data out of the cell. Self-loop allows cell of accumulate information over a long period of time with a forget-gate attached. The forget-gate learns when to forget accumulated information. Typical forget criteria: information out of date or information 'used'. Source: https://ai.googleblog.com/2015/08/the-neural-networks-behind-google-voice.html

They are connected recurrently to themself. Neural units in the hidden layer are replaced by LSTM-cells. This means that LSTM-cells in recurrent neural networks have two recurrencies. Input features are computed by a regular artificial neuron and if the input gate allows it the value of the input neuron is accumulated in the state. The state unit has a self-loop with weights controlled by the forget gate. The output can be denied by the output gate. With this gated weighting of the self-loop, it becomes context dependent. Gating the input as well as the output controls information flow throughout the neural network. Noisy data and irrelevant data do not enter the LSTM-cell (GERS ET AL., 2000).

Long Short-Term-Memory cells can bridge time lags up to 1000 discrete time steps (HOCHREITER ET AL., 1997; GERS ET AL. 2000). In addition, they are capable of handling noisy data. Lastly, they are good in generalization and there is no need for parameter fine tuning.

Drawbacks of LSTM-cells are that with the gating mechanism additional units are needed which increases the number of weights that must be computed (HOCHREITER ET AL., 1997).

One use case in which LSTM perform significantly well is the subject of translation as shown by Sutskever, 2014. For this experiment, the WMT'14 dataset was used. 12 million of sentences were to be translated from English to French. Therefore the 160 000 most common words of the source language and the 80 000 most common words of the target language were considered. Every word that was out of vocabulary was replaced by an UNK token. In general, the main objective of translation is rather capturing the meaning of the source sentence rather than translating it word by word. Many important problems of machine learning are expressed with sequences whose lengths are not know a-priori. The approach presented required an end-of-sentence symbol.

Why does this problem is suitable for using Long Short-Term Memory neural networks? Translating sentences from one language to another basically is mapping an input sequence of variable input length into a fixed-dimensional vector representation which is one of the – for this subject – useful properties of RNN and therefore of LSTM-networks as well. This problem could also be solved by using conventional recurrent neural networks, but the problem is that these cannot handle long term dependencies. Remember that they are limited to 5-10 timesteps which, in many cases, are insufficient time steps when it comes to building sentences.

So, in this experiment a model with a sequence-to-sequence architecture was built. This means that each sequence was associated with a LSTM-networks. The benefit of this approach is that it increases the number of model parameters at a negligible cost with the extra advantage that it makes it natural to learn other language pairs. The main goal of this use case it to estimate the conditional probability of an output sequence given the input sequence. The trained model consisted of 4 layers making it a deep neural network with 1 000 cells each.

This quite simple approach to machine translation performed well. Its performance was improved by reversing the input sequences. By doing so, many short-term dependencies were introduced that made the optimization problem much simpler. LSTM-networks trained on reversed datasets had less difficulties translating long sentences (SUTSKEVER, 2014).

Works about LSTM based stock price prediction models of other Authors

General

There are some Authors who are dealing with LSTM based stock price prediction models. They use platforms like „Towards Data Science“ or private blogs for publishing their works, but there are also a lot of authors who publish on serious platforms.

We took a view on their work containing models and methodologies in form of the model architecture and validation methods. We could see that the models in almost all works were very similar. We also could see that the method to validate and testing the model was in all works the same.

In following, we will discuss the issues of these works.

The same problem of all I – the mean squared error

It can be observed that all authors use the mean squared error as metric for valuing their model goodness and also as an optimization criterion to train their models. We will demonstrate that this metric is not suitable.

The reason is very simple. The mean squared error give only an information about the proximity of the model results to the real values. But in stock price prediction it is more important to predict the trend behavior of the stock price as exact as possible. We will discuss that problem more in detail in a later chapter.

It can also be observed in all articles of those authors, that their model architectures are very simple. The models contain only one or two LSTM layers followed by a dense layer. As mentioned in an earlier chapter LSTM are not able to keep many time steps and their related information. There is for instance the work of ZOU & QU (2020). They use 3 different but very similar model architectures. The following figure was taken from their paper.

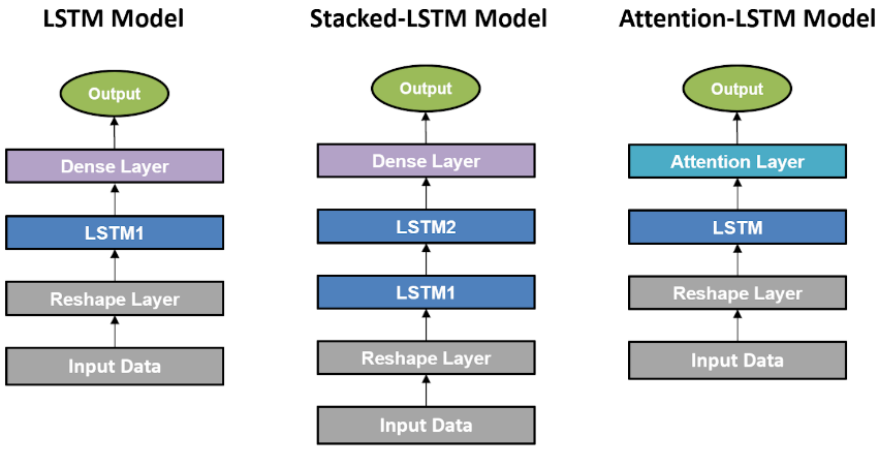

Fig. 8: *Original figure of the model schemes of Zou & Qu*. Original caption: "The architecture of three deep learning models"
Source: Zou & Qu, *Using LSTM in Stock prediction and QuantitativeTrading*, 2020

They give no information about the number of neurons, but it can be seen that there is no neural layer before the LSTM layer, only after. That means, their model is not able to keep pattern information in a very long term manner. An interesting aspect of the work of ZOU & QU (2020) is, that the extensional models (Stacked:LSTM and Attention_LSTM) compared with the simple one have no significant improvement to it.

However, note, that they used the mean squared error as metric for the model goodness. And this metric, as mentioned, is not suitable. Which leads to an assumption, that the model goodness of the two extentional models could be indeed better. But they must use another validation method to prove this.

We will propose and discuss a potential better validation manner in a later chapter where we validate our model replicas.

In no work can be read about the treatment of discontinuities in the time series of stock prices. Of course, the trading time of a stock is interrupted by some periodical, as well as aperiodical events like the end of working days, weekends, holidays and so on. These information gaps must be treated in a suitable manner.

The following figure demonstrates the time series information gaps in terms of the stock price course in a time range of a week. That figure suggests that the stock price course is based on a continual trading process due to the uninterrupted curve.

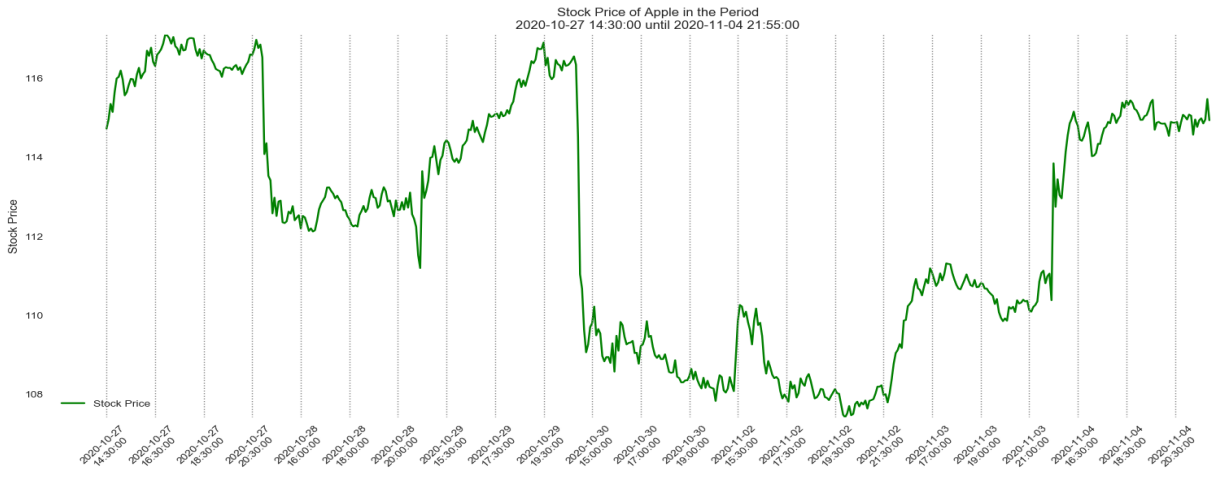

Fig. 9: *The Apple stock price course between 27.10.2020 and 04.11.2020*. The continuity of the curve lsuggests, that the trading process is a continues process without time outs in form of end of working day or weekends.

The following figure shows the same stock price data. However, the time gaps induced by normal periodcal events such as end of working day or the weekend are shown as colored fields without a curve. The curve is shown as an interrupted curve, and was located only in their related specific time ranges of the actually trading period

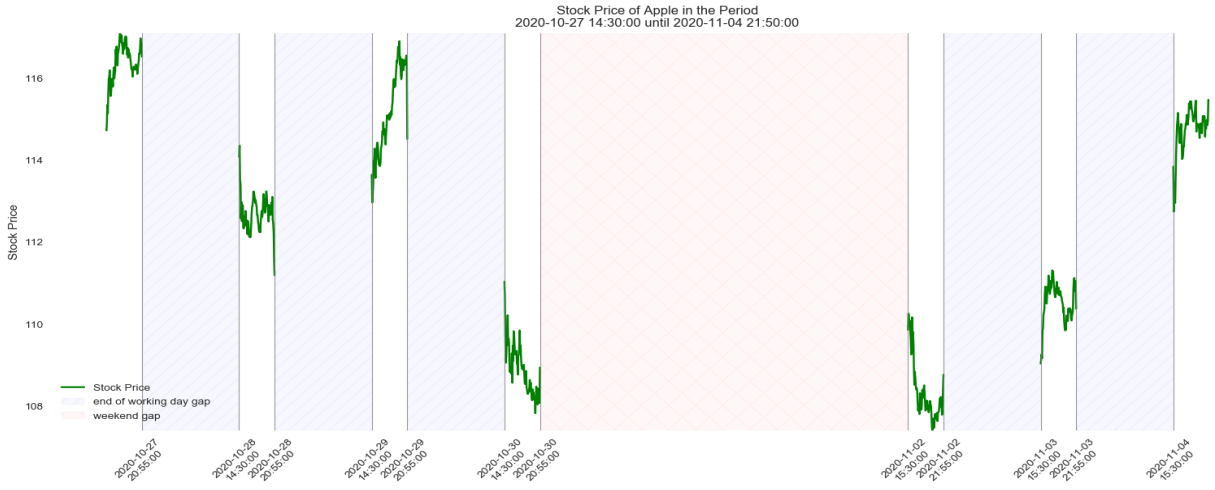

Fig. 10: *The Apple stock price course between 27.10.2020 and 04.11.2020 with time out definitions*. This figure demonstrates the time outs induced by events such as end of working day or weekends. It can be seen, that there are some large gaps between the time period of actually trading activities.

As can be seen in the last figure, the time series of stock prices actually have time outs or gaps without activities. It also shows the ratio between the actual activity time periods and the time outs. The time outs are much larger than the trading periods. Furthermore, it can be seen that after such a time out the first stock price can be somewhat different to the last stock price before the time out. A prediction model needs to be able to recognize such time outs and their impact on the stock price.

While ZOU & QU (2020) do not discuss a treatment of those problems, and we have to assume, that they did any suitable treatments of that, some authors did definitely not use a treatment of their time series. This means they trained their model on bad suited time series.

Our Study

Model I

The first model is as close as possible to the models we had seen in other works. We also use the mean squared error as the optimization criterion. However, we use a different approach to validate our model. In some other works either the validation is done by just a model result visualization and the visualization of the real stock price values or they simply used the mean squared error between the model outputs. The very urgent problems is that they use for each new time step prediction the last real value as feedback to the LSTM model and not the LSTM output.

Instead, we use each new predicted output value and feed it back into the LSTM instead of the real value. We have to keep in mind, that the model cannot know the real value, because they lie in the future and are unknown, of course. If we look more timesteps into the future, we cannot correct the model with the real values.

We show this principle in more detail in the methodology chapter.

Model architecture

We use a simple designed model as used by some other authors. We use TENSORFLOW and its framework KERAS for model design. The following code snipped represents the model definition:

```
# build the model
self.model = Sequential()
# we use one LSTM layer with feedback (bidirectional)
self.model.add(LSTM(
    self.seq_len,
    return_sequences=True,
    input_shape= (x_train.shape[1], 1))
)
# the next LSTM is unidirectional
self.model.add(LSTM(
    self.seq_len,
    return_sequences=False)
)
# a density layer with the a number of neurons equal to the windowsize
self.model.add(Dense(2*self.seq_len))
#
self.model.add(Dense(1))

#compile the model
self.model.compile(optimizer='adam', loss='mean_squared_error')
```

The first layer is a bidirectional LSTM layer followed by an unidirectional LSTM layer. The further layer is a dense layer as well as last layer, that represents the model output.

The model is dynamically parametrized by the window size of the rolling window. The window size directly determines the number of neurons of each layer. In more detail each LSTM layer has the same number of neurons like the window size. The first following density layer has twice the number of neurons as the window size. Larger numbers do not lead in better results. We do not use an activation function after the density layers, because the reference models of the other authors do not have those as well. The last dense layer has only one neuron that represents the output. In this manner we are able to try some different model parametrizations by alter the window size but keep the simplicity of the reference models.

Like some seen models, we use also the mean squared error as optimization criterion.

Methodology

We use some different stock price samples of the APPLE stock price which was taken by use the YAHOO FINANCIAL API. In detail, we have 5 different stock price courses of APPLE in different time ranges. We choose randomly one of them. For this model the time discretization is set to 1 minute. That means, the generated time series provides the stock price at each minute. The entire time range is 2 trading days without a treatment of the time outs as discussed in chapter above *The problem of discontinuities*. We use only the closing price of the timesteps for model training and usage of the model.

We divide the time series into 3 parts. The first part is to train the model. The second part is for the model testing and observation in terms of overfitting and the third part is for the model validation. That is the typical manner in the works from other authors and in general in supervised machine learning tasks. The size of these 3 parts is approximately 60% - for training. 20% for testing and 20 % for validation. But we use also other configurations. To have more different model scenarios.

As mentioned in the chapter about the architecture we use the mean squared error as optimization criterion. We observe the mean squared error while training process to see the training effect.

To validate the model, we let the model predict the stock price about some future time steps and compare them by visualizing them with the real values.

We do that to get a better overview of the model results. This method is something different to the model validation of some other authors. Some of them only use the mean squared error between the validation data and the model results. We will discuss this method and its weakness in this context in a later chapter.

The following figure shows the principle of the model prediction for the visual validation.

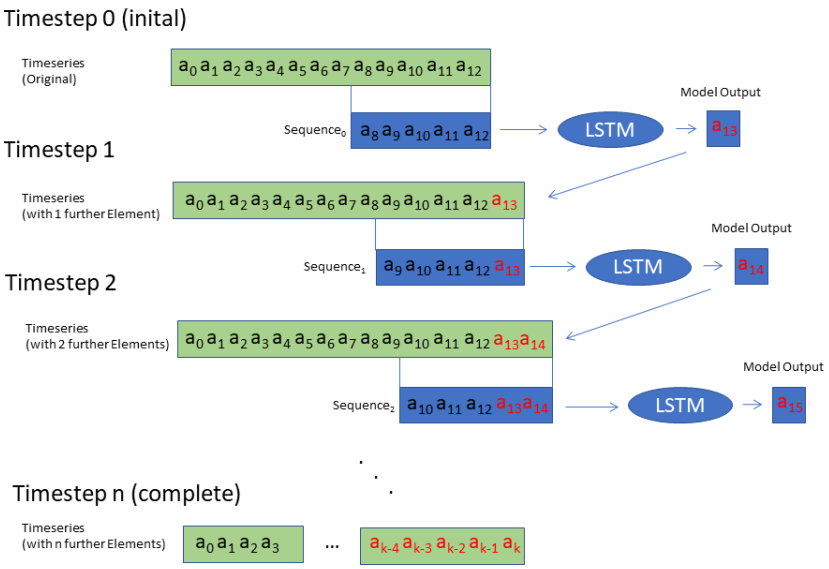

Fig. 11: *The model scheme.* After the model was trained with the time series data of the past, it tries to predict the future time step related stock prices. These will be feeded backwards into the rolling window (called Sequence in the figure) because they are part of the new present after the predicted timestep.

In other manner like some other authors, we use the model to predict some new timesteps and use the predicted new time step related stock price prediction as feedback into the model. That means, instead of in the models of those authors, our model is not corrected after each future timestep with the real value (the real one remains unknown to the model!).

After some generated forecasted future timesteps and their related stock prices we validate our model by a visualization of these result and the real values. If the model works

well, the curve of predicted values should, of course, be very close to the real values and following their trend.

Results

In this chapter we present some results of our model for validation purposes. It can be seen the entire time range of training-, testing and validation phase of the model. The red curve shows the model result across a particular time range. The fat plotted blue curve shows the real stock price values. The green curve shows the model results after the training process. The orange curve shows the model result in terms of the test time range. This result can be used to see whether the model is well fitted or overfitted. The curve would be more away from the blue curve compared with the green one to the blue, if the model would be overfitted. The violet-colored curve shows a pseudo validation of the model, how can be seen in some other works. It suggests that the model works very well and can predict some timesteps.

The second figure is related to the first one and shows the model results of the model in the manner of our validation method. We look some timesteps into the future and feed each predicted value back into the model. The red curve shows the result.

We show some other cases in that manner in the appendix. Of course, this is not representable in a statistical manner. However, we tried some cases and could see, that most of the results were negative. That means, there is a trend that lets assume, that this model is not able to do reliable stock price predictions.

An exemplary example result of a model.

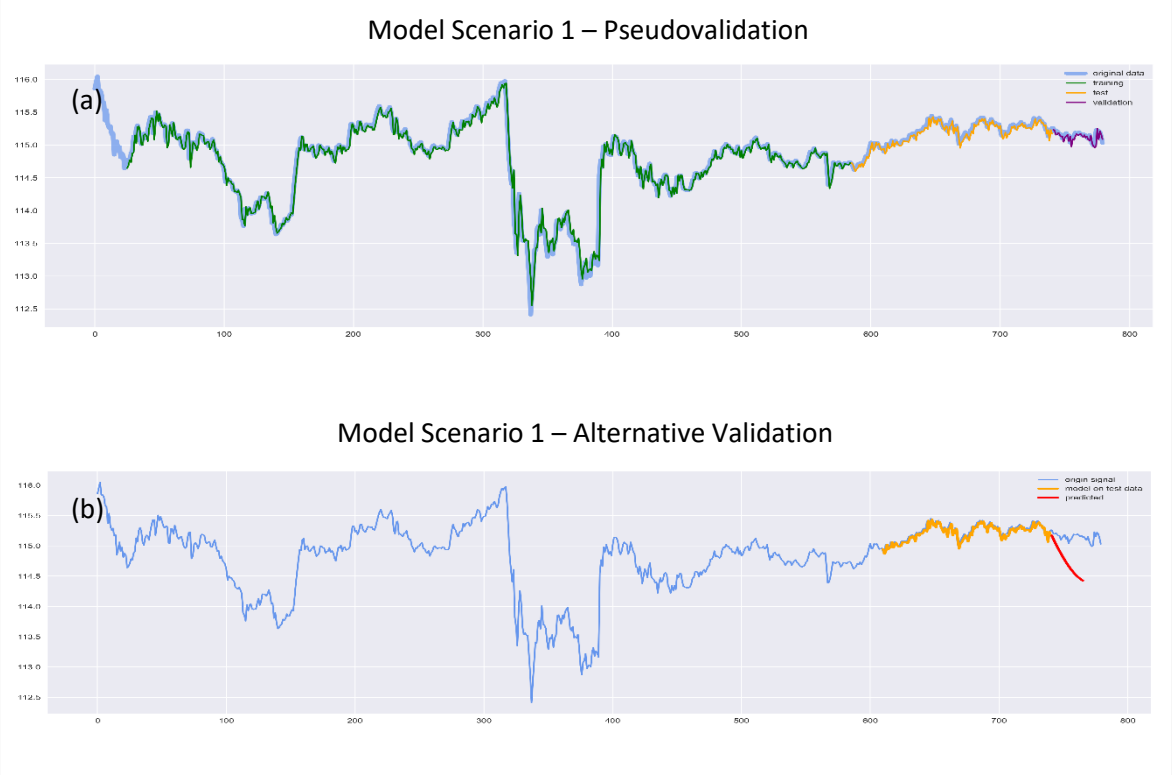

Fig. 12: *Model validation in manner of some other authors and the alternative manner.* Both charts represent the same stock price base data. a shows as blue fat plotted curve of the entire real stock price course. The green curve represents the model result after the training process in its related time range. The yellow curve represents the model results about the test data. The violet curve represents the model validation in manner of some other authors. All model related curves match the blue curve verry well. The well matching yellow curve shows that the model is not overfitted. Otherwise, it would be more divergent than the green one. In b can be also seen the real stock price as blue curve. The yellow one also represents the test results again. But instead of the model validation result in the manner of the other authors, we used our method and show the results as the red curve. It can be seen that the curve matches the not the blue curve.

Discussion

As can be seen, the model result as an example for some other model realizations we did, shows a large deviation from the real course. While the validation results in some authors works matches the real values very good, our model is diverging more and more with each new time step.

If we are interested in some future time steps and not in only one, we have to consider that the model cannot know the real values.

The validation manner of some authors seems not to consider this. They do not use the predicted timesteps and feed them back into the model to forecasting the new one. Instead, they use the real but actually unknown timestep related stock price. But of course, that make no sense, unless the model is only to predict exactly one timestep.

To consider this particular case, we designed a second model approach, which only predicts one timestep into the future. Please see for it the next sub chapter.

Model II

Model architecture

The model architecture in terms of the neural Network is the same as in the first model. We do so to ensure comparability of the validation approaches.

Methodology

The methodology is some different to the first model. Unlike the first model, this model is used to predict only one timestep into the future. In a real case, a trader can bet on the stock price change within the next 5 minutes. Those time ranges might by significantly in term of CFD trading. We examine only the trend of the predicted timestep. That means, we are only interested, whether the course relatively to the last value (the known present) will increase or decrease. That implies, there are two different kinds of trend. A positive trend if the course is increasing, and a negative trend, if the stock course is decreasing. A positive trend for the next timestep related stock price is than the case, if this stock price value minus the last stock price value before is a positive number. Otherwise, the trend is a negative. The advantage is, we can define two different vectors which contain the timestep related trend definition. The first vector contains all true timestep related trend definitions. And the second vector contains the predicted trends of the model.

That makes it very simple to validate the model in a statistical manner. We simply compare, if the real trend is equal to the predicted trend or not.

The following expressions show the principle of those trend vectors in terms of the definition of their elements. Expression 1 defines the i-th element of the trend vector regarding the real trends. Expression 2 defines the i-th element of the trend vector regarding the predicted stock price trends. i is the timestep index. |D| and |M| each mean the number of containing elements in these data sets. M is related to the model output regarding the time range of model validating. That means, we use furthermore the validation time range for this kind of model validation.

$$trend^{i}_{real} = \begin{cases} positive, & if \quad D_{|D|-|M|+i+1} - D_{|D|-|M|+i} \geqq 0 \\ negative, & else \end{cases} \tag{7}$$

$$trend^{i}_{pred} = \begin{cases} positive, & if \quad M_{i+1} - D_{|D|-|M|+i} \geqq 0 \\ negative, & else \end{cases} \tag{8}$$

As can be seen, each vector contains only the binary values *positive* and *negative*. These two values are sufficent enogh to describe the model goodness in our way.

After the normal model training by using only the time series data related to the time range of model training, the model predicts the stock price related timesteps for the testing time range as well as for the validation time range. We use the testing time range again, to ensure that the model is not overfitted. Than we compute based on the real data the corresponding trend vector as well as the trend vector of the model data. Due to the fact that we consider only one time step for our prediction, we can apply this method. We do not feed the predicted stock price value back into the model, but use the corresponding real value instead. That is the diffrence to the first model. There, the manner to use the real stock price value as feedback signal instead of the predicted one, was not meaningful. But in this case, we consider only the ability of predicting one timestep.

That is why we can apply a better manner of statistical validation by comparing both trend values.

We do not use e metric like the mean squared error, which can only give an comparsion in form of the similarity of the two well ordered data sets, but we use the directly comparsion of the two values and consider the frequence of a hit of the model. We consider the 4 cases which can be happen and classify them as:

TP := True Positive, TN := True Negative, FP := False Positive and FN := False Negative.

In more detail:

- **First case:** The real trend value $trend^i{}_{real}$ at the i-th timestep is positive and and the corresponding predicted trend value $trend^i{}_{pred}$ is also positive

  → TP

- **Second case:** The real trend value $trend^i{}_{real}$ at the i-th timestep is negative and the corresponding predicted trend value $trend^i{}_{pred}$ is also negative

  → TN

- **Third case:** The real trend value $trend^i{}_{real}$ at the i-th timestep is negative but the corresponding predicted trend value $trend^i{}_{pred}$ is positive

  → FP

- **Fourth case:** The real trend value $trend^i{}_{real}$ at the i-th timestep is positive but the corresponding predicted trend value $trend^i{}_{pred}$ is negative

  →FN

To get a well understandable overview of the validation result, we use a confusion matrix to visualize them. The following figure shows the scheme of a confusion matrix for this purpose.

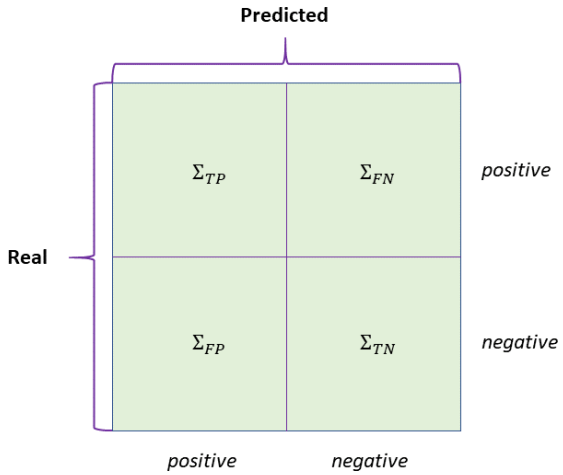

Fig. 13: *Scheme of a confusion matrix adapted to this validation purpose.* This matrix consists of two column and two rows. Each so definable field (element) represent the sum of the related cases (TP, TN, FP, FN). It gives a good overview about the ratio of the frequency of all 4 cases. The major diagonal consists both cases when the model prediction was correct (TP and TN), whereas the 2 other elements represent both possible cases when the prediction was incorrect. The configuration of these 4 values gives on overview over the balancing of the not correctly prediction to the correct prediction and the balance between the two different kinds of correct predicted cases and the two kinds of not correct predicted cases.

The confusion matrix design, which we use, is defined by a 2 x 2 matrix. Each of the four possible cases is related to a corresponding row and column. Thereby is the row and column related value the sum of all occurred cases across all validation results.

For example, let the number of all validation cases be N. Thus, is the length of the two trend vectors N, so we can iterate over these two vectors and compare the both values at the i-th iteration or element index. The comparison is mapped to one of the 4 cases (TP, TN, FP, FN). The frequency of these cases across all validation value pairs $({trend^i}_{real}, {trend^i}_{pred})$ defines the confusion matrix entry at the related column and row. If the number of all true positive cases of the N validation value pairs is X, the matrix element Σ_{TP} would be X. The same principle applies to the other 3 cases and their frequencies.

The benefit of this matrix is, to get a well overview of the balance between the frequencies of those cases. Thus, we also see how the frequencies are distributed to see possible relationships like a model bias etc. As example of such bias, the frequencies of the major diagonal would be similar (all correct predicted cases are well balanced) but the frequencies of the negative cases would be significant different.

A reliable indicator for an invalid model would be a significant difference between the frequency of TP and the frequency of TN.

As a further method to define the goodness of the model, we use the accuracy as a well known metric to quantify the model goodness of classification models. The accuracy is defined as the number of correct classified or predicted values divided by the total number of classifications or predictions. In the case of our model validation the accuracy is defined as the equation (3) below:

$$acc = \frac{\Sigma_{TP} + \Sigma_{TN}}{\Sigma_{TP} + \Sigma_{TN} + \Sigma_{FP} + \Sigma_{FN}} \tag{9}$$

Since the accuracy is the ratio between the total number correct predicted trends and the total number of all cases, we get a possible number range of 0 to 1. Whereas 0 would be the case, that no prediction was correct, 1 would imply, that each prediction was successful. That means the closer the accuracy is to 1, the better is the model. However, if the model has an equal distributed stochastic noise around the true value, we have always to expect an accuracy of about 0.5 if the model behaves randomly. Furthermore, accuracy works only as a meaningful metric if both majority cases (positive trend, negative trend) are well balanced in their total numbers. That means, we should have approximately the same number of positive trends as well as the same number of negative trends across the entire validation time range.

To ensure that the prediction is significant and not caused by coincidence, we need a minimum level of accuracy, from which the model can be assumed as trustful. Therefore, we compute the corresponding significance level of each model scenario. Since we assume an equal distribution of the both majority cases (positive trend, negative trend) if the model behaves randomly, we can assume this H_0-hypothisis: $p = 0.5$. That means, each case has the probability of 0.5. The expected value is defined as $\mu = n \cdot p$, whereas n is the number of all predictions by the model. By computing the standard deviation σ with following equation:

$$\sigma = \sqrt{n \cdot p \cdot (1 - p)} \tag{10}$$

we can define the 0.95 confidence area by using the corresponding z value multiplied with σ as:

$$\mu \pm z \cdot \sigma \tag{11}$$

Due to the fact, that we consider only the minimum value of accuracy, we need only to consider the positive interval. Thus, we can define the significant accuracy level as:

$$acc_{sig} = \frac{\mu + z \cdot \sqrt{n \cdot p \cdot (1-p)}}{n} \tag{12}$$

The corresponding z value is 1.65 (taken from: http://eswf.uni-koeln.de/glossar/zvert.htm). So, we can generalize:

$$acc_{sig} = 0.5 + \frac{1.65 \cdot \sqrt{n \cdot 0.25}}{n} \tag{13}$$

Each accuracy value that is greater than this significant accuracy level, would deny the H_0-hypothesis and can be assumed as very sure sign, that the prediction ability of the model is given.

Results

By applying this validation approach, we are able to validate our model in a statistical representative manner. We consider some prediction cases related to different time series to have a large sample size for the validation. In following we present 3 model scenarios and their validations. The differences are defined by the chosen timeseries. For each of such a model run, we show the related stock price course chart with the time range visualization as seen before in the first definition and how it can be seen in the publications of other authors. In this manner we can see if the model is well fitted or overfitted and see the time range of the validation part. Also, the comparison with the visual validation manner used by other authors to the statistical manner can be seen directly. We show the related confusion matrix with the frequencies of the prediction results. In addition, we show for each model scenario the corresponding mean squared errors of the 3 phases (training, testing, validation), to make a comparison between the validation manner of the other authors which use the mean squared error as metric for the model goodness and our validation manner. We show the accuracy as well as the scenario related significant accuracy level to see, if the model can be assumed as a good predictor.

Model Scenario 1

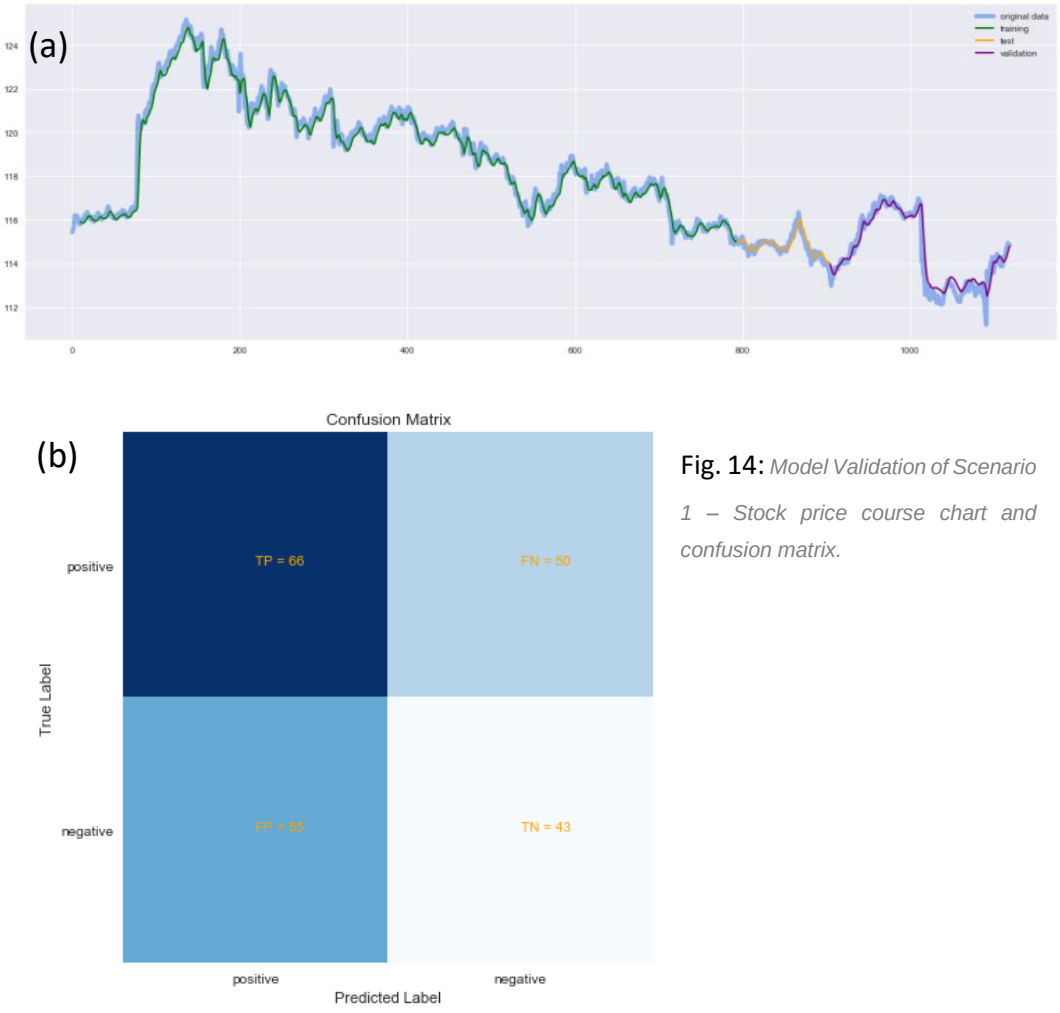

Fig. 14: *Model Validation of Scenario 1 – Stock price course chart and confusion matrix.*

MSE_{train}	MSE_{test}	MSE_{valid}	n	σ	$\frac{pos\ trend\ [\%]}{neg\ trend\ [\%]}$	acc_{sig}	acc_{mod}
0.0009	0.0003	0.002	214	7.3144	54/46	0.56	0.51

Tab 1.: *Model Validation of Scenario 1 – Table of model result statistics.* MSE_{train} is the mean squared error between the real data and the model results in the time period of the model training. MSE_{test} is the mean squared error between the real data and the model results in the time period of the model testing. MSE_{valid} is the mean squared error between the real data and the model results in the time period of the model validation. n is the number of predicted values within the validation time period (sample size). σ is the standard deviation of the sample. $\frac{\sim pos\ trend\ [\%]}{\sim neg\ trend\ [\%]}$ is the percentual ratio between both shares of positive trend values and negative trend values. acc_{sig} is the significant accuracy level. acc_{mod} is the accuracy of the model.

Model Scenario 2

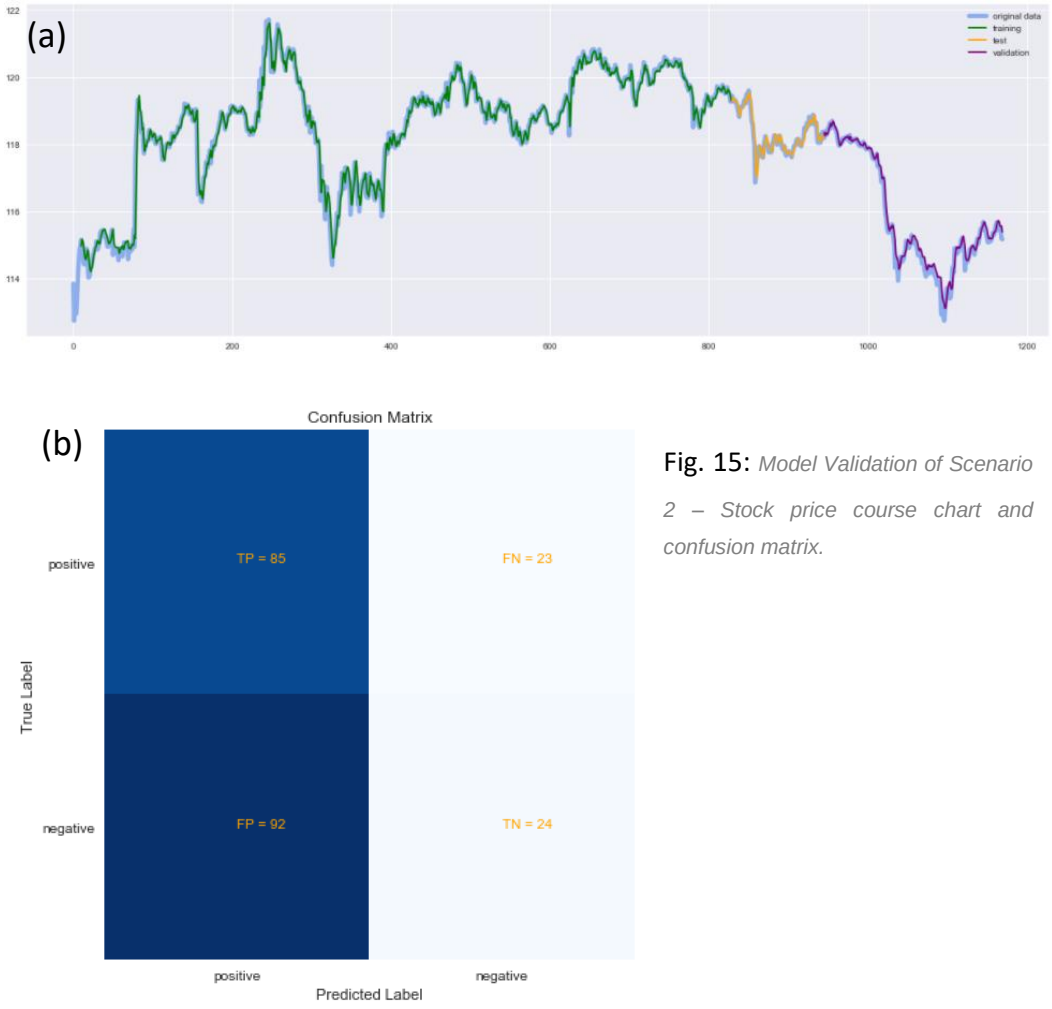

Fig. 15: *Model Validation of Scenario 2 – Stock price course chart and confusion matrix.*

MSE_{train}	MSE_{test}	MSE_{valid}	n	σ	$\frac{\sim pos\ trend\ [\%]}{\sim neg\ trend\ [\%]}$	acc_{sig}	acc_{mod}
0.0011	0.0005	0.0006	224	7.4833	48/52	0.56	0.49

Tab 2.: *Model Validation of Scenario 2 – Table of model result statistics.* MSE_{train} is the mean squared error between the real data and the model results in the time period of the model training. MSE_{test} is the mean squared error between the real data and the model results in the time period of the model testing. MSE_{valid} is the mean squared error between the real data and the model results in the time period of the model validation. n is the number of predicted values within the validation time period (sample size). σ is the standard deviation of the sample. $\frac{\sim pos\ trend\ [\%]}{\sim neg\ trend\ [\%]}$ is the percentual ratio between both shares of positive trend values and negative trend values. acc_{sig} is the significant accuracy level. acc_{mod} is the accuracy of the model.

In all of these 2 model scenarios can be seen that the predicted stock price values are very close to the related real values. We can see very similar results as shown in publications of other authors in terms of the result curves and the mean squared errors.

The first model scenario has a greater value of the MSE about the validation time range that we can see in the publication of ZOU & QU (2020). The ratio between the percentual shares of positive trend values and the negative trend values shows with 54 / 46 a well-balanced value base. We can be sure that the accuracy can be used as a representative metric. The corresponding significant accuracy level is 0.56. Since the model accuracy is 0.51 that is smaller than the significant accuracy level, we can assume the H_0-hypothesis. The model has no significant prediction ability.

The second model scenario has a similar value of the MSE about the validation time range that we can see in the publication of ZOU & QU (2020). The ratio between the percentual shares of positive trend values and the negative trend values shows with 48 / 52 a well-balanced value base. We can also be sure in this model scenario, that the accuracy can be used as a representative metric for the model goodness. The corresponding significant accuracy level is 0.56. Since the model accuracy is 0.49, that is smaller than the significant accuracy level we can assume the H_0-hypothesis. The model has no significant prediction ability.

Conclusion and Outlook

To predict the stock price behavior in very small time range of only one time step, we are interested if the stock price relative to the present one will be increasing or decreasing. But if we use only the mean squared error as distance information, we lose the information of those trends. The mean squared error as optimization criterion is sufficient for a model which should only be close to the real value. But in the context of stock price prediction in a very short term manner (only one future time step), it is not enough to be close to the real value. We can show that models also with a small mean squared error are not able to make reliable predictions. Thus, mean squared error cannot be sufficient to optimize and validate models in short term manner. That is why we must consider more detailed models for the prediction of stock prices. To show the models results as curves those matching really good the real value curve suggests very well performing models. But as we could show, this is not representative and may lead to wrong assumptions, that the model would be able to predict across a long-term period with good results.

For long term predictions (predictions about more than one future time step), the mean squared error might be a suitable metric and an optimization criterion. However, we can show with model 1, that long term predictions probably do not work, too. The problem may be caused by the strong influence of failed predictions relative to all predictions in the prediction time range. Since each predicted value is fed back into the model as new value of present for the next timestep, the prediction error of this prediction is influencing the next predictions. This principle acts as a chain effect. The prediction of each further timestep is depended to all predicted timestep in the past and thus to their prediction errors.

Neither the models nor the manner of model validation and presentations of some authors can be seen as good. Their publications suggest that their models are able to predict stock prices. But they are not able to do that. They show only their results as curves close to the real value curve, but as we can show, this is not a sufficient evidence for a proper model especially when the model is corrected by real future values during the prediction phase.

However, we cannot say, that LSTM based models in general do not work. Perhaps it is possible, that LSTM based models with more complexity and maybe coupled with other techniques like CNN or other and more suitable optimization methods could have an ability to predict stock prices. Of, course, it is very attractive to spend more time in it to investigate those model architectures. The next seminar task will come…

Bibliography

GERS, F.; SCHMIDHUBER, J.; CUMMINGS, F. (2000): "Learning to forget: Computational prediction with LSTM", *Neural Computation*, 12(10), 2451 – 2471, DOI: 10.1162/089976600300015015

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. (2016): "Deep Learning- Das umfassende Handbuch", ISBN: 978-3-95845-700-3, mitp-Verlag

GOOGLE BRAIN TEAM (2020): "Tensorflow"

GOOGLE BRAIN TEAM (2020): "Keras"

HOCHREITER, S.; SCHMIDHUBER, J. (1997): "Long short-term memory", *Neural Computation*, 9(8), 1735-1780, DOI: 10.1162/neco.1997.9.8.1735

LOUKAS, S. (2020), "*Time-Series Forecasting: Predicting Stock Prices Using An LSTM Model*", Towards Data Science | https://towardsdatascience.com/lstm-time-series-forecasting-predicting-stock-prices-using-an-lstm-model-6223e9644a2f, access: 10.12.2020 14:58

MITCHELL, T. (1997): "Machine learning", McGraw-Hill, New York, ISBN: 978-0071154673

ROONDIWALA, M.; PATEL, H.; VARMA, S. (2017), "Predicting Stock Prices Using LSTM", International Journal of Science and Research (IJSR)ISSN (Online): 2319-7064

SUTSKEVER, A; VINYALS, O.; QUOC, L. (2014) "Sequence to sequence Learning with Neural Networks", NIPS'2014, arXiv:1409.3215

ZOU, Z.; QU, Z. (2020), "Using LSTM in Stock prediction and Quantitative Trading", CS230: Deep Learning, Winter 2020

Appendix

Model 2

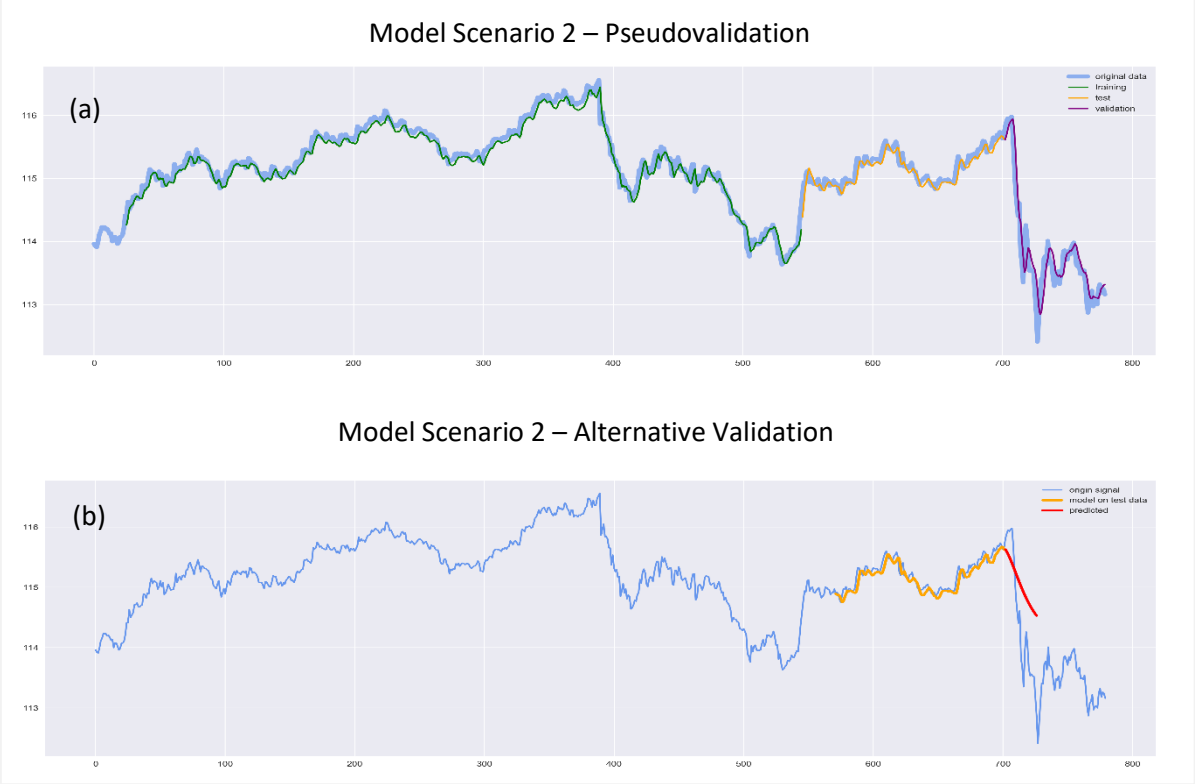

Fig 16.: *Model validation in manner of some other authors and the alternative manner.* The prediction model output is right in terms of the stock price trend in the specific time range but does not match the curve of real values well.

Model 3

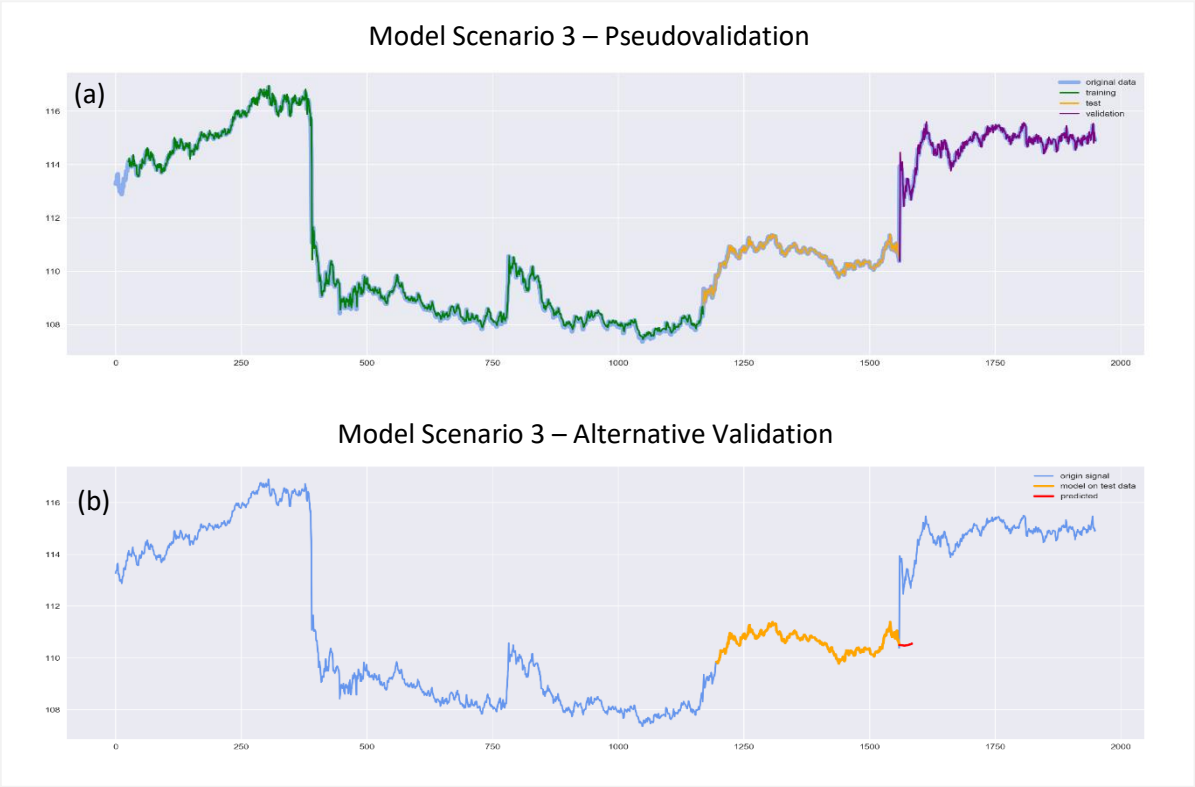

Fig 17.: *Model validation in manner of some other authors and the alternative manner.* The prediction model output is right in terms of the stock price trend in the specific time range but does not match the curve of real values well.

Model 4

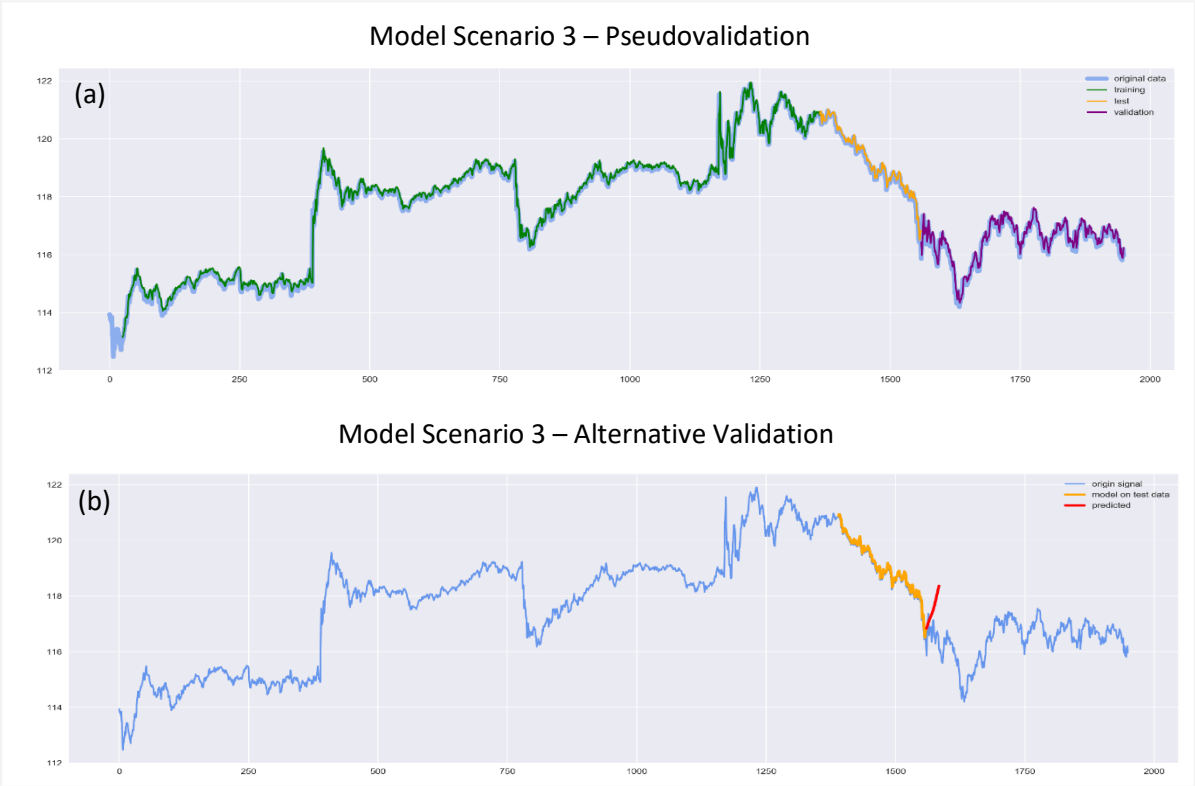

Fig 18.: *Model validation in manner of some other authors and the alternative manner.* The prediction model output is right in terms of the stock price trend in the specific time range but does not match the curve of real values well.

Declaration of Authorship

Hereby, I declare that I have composed the presented paper independently on my own and without any other resources than the ones indicated. All thoughts taken directly or indirectly from external sources are properly denoted as such.

Köthen, Halle, 21.12.2020

Alexander Prinz, Philipp Nostitz