UPPSALA
UNIVERSITET

Short term stock evaluation

Using LSTM/Transformer methods

Carlsson, Victor
Victortommy.Carlsson.9424@student.uu.se
Jacobson, Oscar
Oscar.Jacobson.9201@student.uu.se
Slagbrand, Johan
Johan.Slagbrand.0528@student.uu.se
Åsell, Martin
Martin.asell.0292@student.uu.se

Project in Computational Science: Report

February 15, 2023

PROJECT REPORT

Institutionen för informationsteknologi

Abstract

Forecasting stock market is considered a challenging task, due to stock prices being influenced by a wide range of complex and dynamic factors. The use of artificial intelligence in finance has gained significant attention in recent years due to its potential to enhance the accuracy of stock market predictions. The foundation of this work is based on the hypothesis that stock markets follow reccurring patterns and one can leverage fluctuations in price and conducted volume over time in order to make forecasts. The authors propose two neural network architectures for predicting short-term movements of a stock based on historical data. The explored approaches were a Bayesian long short-term memory model and a transformer based model. Due to design differences between the networks, separate evaluations were employed and therefore the models could not be directly compared. Both architectures performed significantly better than random guessing. However the findings of the study particular highlight the potential of the Bayesian aspect of the long short-term memory model and suggest that the integration of probabilistic techniques into deep learning models can lead to advancements into financial forecasting.

Contents

1 Introduction

When speculating in the stock market, a number of strategies can be applied. For instance, placing capital in a global index fund is a common *passive-investing* method that has historically paid off well over larger time horizons. Passive-investing in a S&P500 index fund often serves as a benchmark for evaluating other strategies during the same time period. However, to successfully *actively* manage ones capital over time, the actor requires high-quality information. Historically, fundamental and technical analysis have played a key role in providing this information.

To make a good informed decision regarding a stock one would have to be up to date on all major aspects that may affect the stocks price. For instance how the price have moved in the past, how the market values the stock etc. This might not seem as a big challenge but when considering a stock where there are a large number of factors to consider, the task becomes more complex. However, there are certain situations where all information is not publicly available, one example of this is whats called *inside trading*, where some people decide amongst themselves that they will either buy or short a certain stock. Some other examples that can influence the market in a negative way are natural catastrophe, war, politics and so on.

To accurately predict stock market movements, one would have to stay up to an immense number of news and events playing out in real time. This is of course hardly achievable to do by any human. In this project we are interested in developing two neural network models that can make decisions based on historical price data. This is called time series modeling and is very common in the field of stock market analysis. The two models that we developed for this project are a *Bayesian Long-Short Term Memory network* (LSTM) as well as a *Transformer network* which both show promising results.

2 Related Work

Stock market predictions have been a research topic since the birth of the traditional stock market. With the introduction of the internet, and technologies such as deep learning, this topic, like many others, has evolved quickly since the beginning of the AI boom we are currently living in. There are some papers which accompany the flourishment of AI and deep learning which consequently have impacted the field of neural network usage in stock market prediction.

Pagolu et al. [2016] is a research paper that investigates the use of regression analysis for predicting stock prices. Regression analysis is a statistical method used to model the relationship between a dependent variable, in this case stock prices, and one or more independent variables, such as economic indicators, market sentiment, and company-specific information. The results of the study showed that regression analysis can be used to accurately predict stock prices with great precision. However, the authors also caution that regression analysis should be used in conjunction with other analysis techniques when making decisions. This caution follows from the tendency of regression to closely *mimic* the last point when making predictions. Thus not introducing any significant new information or insights that is not already known.

Sahoo and Charlapally [2015] is a research paper that explores the use of sentiment analysis on Twitter data to predict stock market movements. Sentiment analysis is a technique that uses natural language processing and machine learning algorithms to analyze the tone and emotion expressed in text data, in this case, tweets. The authors of the paper aim to determine if the sentiment expressed in tweets about stocks and companies can be used to predict the movement of the stock market. The results of the study showed that sentiment analysis of Twitter data can be used as a tool when making predictions in the stock market with a significant accuracy.

Ghosh et al. [2020] investigates the use of machine learning algorithms, specifically LSTM networks and random forests, for predicting the directional movements of stock prices in an intraday trading setting. The results of the study showed that the LSTM and random forest models were able to accurately predict the directional movements of stock prices. The authors found that the LSTM model outperformed the random forest model in terms of prediction accuracy, but that the random forest model was more robust and less sensitive to overfitting.

3 Method

There are several ways to evaluate the performance of a neural network used for stock prediction, each with its advantages and limitations. First one need to consider the type of output from a network, that is regression or classification. If the purpose is to utilize the proposed models in a real life scenario, regression tends to mimic the stock price movements from the previous step, hence not supplying a significant amount of new information and therefore not a suitable candidate for forecasting. Because of this, classification of the close-price direction was chosen as the target for this study.

Due to differences in architecture, the implemented Bayesian LSTM model only predicted up or down movements, whereas the Transformer model also had a *sideways class*. The reason for different classification approaches is motivated by the the proposed Transformer model struggling with binary classification. This struggle was probably caused by the model getting trapped in local minimum. The sideways class refers to no significant movement in price. The threshold which defines *significant movement* determines the three classes and was selected with respect to the volatility of the stock.

During the project it was discovered that having the sideways-movement class being smaller than the other two classes was beneficial to the training of the neural network. This was due to the model heavily favouring the sideways-movement in the predictions. To avoid an overall directional bias in the data, the distribution of the "up" and "down" class was modified to be equal in size, by removing samples from the major class.

3.1 Data processing

To increase the trustworthiness of the models, a variety of different data sets were tested, all with the resolution of one hour. Stock market data is in the form of a time series consisting of a time stamp and the open, high, low and close prices as well as the trading volume for that period.

Apart from the six features previously mentioned, two of the data sets that are presented in this study also included a total of 16 features, adding information about the sector index of the corresponding stock and the SP 500 index. These indices are composed of exchange-traded funds (ETFs) containing a selection of companies representative for

its domain. The dates were also removed from the timestamp, leaving just the normalized intraday time with 0 and 1 representing the first respectively the last hour of a day.

The exact input itself to the neural network consisted out of overlapping continuous sequences of data points. The sequences were randomly shuffled to capture different market conditions, on which the model was trained and evaluated on. Through testing, the final Bayesian LSTM model got sequences of 14 data points as input with min-max normalization applied to each sequence. The final transformer model received sequences of 7 data point as input with a min-max normalization applied to the whole data set.

The data sets were retrieved from Yahoo Finance through the Python library yfinance and goes back approximately two years in time. A larger data set (2009-2022) with just the SP 500 index itself was downloaded from *backtestmarket.com*.

3.2 Recurrent Neural Networks

A recurrent neural network is a type of neural network that is particularly well-suited to processing sequential data, such as time series or natural language. In a traditional feedforward neural network, the input data is processed independently and passed through the network once, producing a single output. In contrast, a recurrent neural network processes the input data in a sequential manner, with the output of each processing step being fed back into the network as input for the next step. This feedback loop allows a RNN to capture dependencies between the input data and the output over time, making it suitable for tasks such as language translation or language modeling as well as time-series predictions.

3.2.1 Vanishing Gradient problem

While RNNs are well suited to learning dependencies in sequential data, they can be difficult to train because of implementational difficulties of gradients and gradient descent. The gradients of the loss function with respect to the weights of the network has a tendency to either vanish or explode as they are backpropagated (see. Rumelhart et al. [1986]) through time.

In Bengio et al. [1994] the authors argue that traditional gradient descent algorithms, which are commonly used to train neural networks,

struggle to learn long-term dependencies because of the inherent assumptions they rely on. The vanishing gradient problem is also described using equation 1 which represents the derivatives of a cost C_t at time t.

$$\frac{\partial C_t}{\partial W} = \sum_{\tau \leq t} \frac{\partial C_t}{\partial a_\tau} \frac{\partial a_\tau}{\partial W} = \sum_{\tau \leq t} \frac{\partial C_t}{\partial a_\tau} \frac{\partial a_t}{\partial a_\tau} \frac{\partial a_\tau}{\partial W} \tag{1}$$

With the assumption made in equation (2):

$$\tau \ll t \Rightarrow \left[\frac{\partial C_t}{\partial a_\tau} \frac{\partial a_\tau}{\partial W} \right] \to 0 \tag{2}$$

The problem proposed here is that a cost-term for when τ is substantially smaller than t, becomes incomparably smaller than a cost-term for when τ is close to t. This causes the gradient of the cost to heavily favour the terms in the *near past* reducing the ability for the gradient descent algorithm to make optimization based on long-term dependencies.

A uncommon secondary effect of the above explained problem is when the gradient descent ends up on, and remains on, the boundary between two attraction zones the limit of the gradient becomes infinitely large as τ becomes smaller than t, this is called the exploding gradient problem and is a edge case of the vanishing gradient problem.

The problem is more easily understood when visualized in Figure 1 as an exponential decay of gradients when moving away from current time. Causing an optimization algorithm to de-emphasize the long term dependencies far from the center (where the center represents current time).

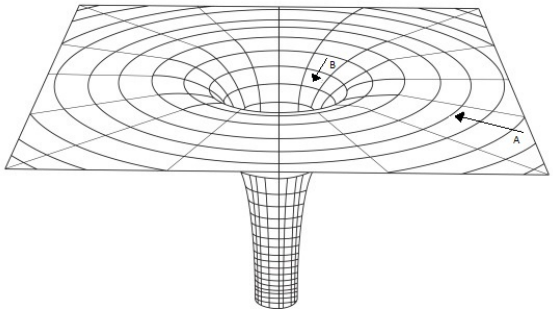

Figure 1: The drop-of in gradient visualized, larger distance to the central *convergence basin* gives smaller relative gradients.[1]

[1]Hortense Ardalan, via http://hortenseardalan.com/blackholes.html (2014-10-19)

3.3 Long Short-Term Memory (LSTM)

The paper *"Long Short-Term Memory"* by Hochreiter and Schmidhuber [1997] presents the LSTM model, a type of recurrent neural network that is capable of learning long-term dependencies in sequential data. LSTMs are designed to address the vanishing and exploding gradient problems that are commonly encountered when training traditional RNNs, which can make it difficult for the network to learn long-term dependencies.

LSTMs introduce a mechanism called gates which are able to control the flow of information in a network. These gates allows the model to selectively *remember* or *forget* information from previous iterations, allowing for the retention of long-term dependencies in the data.

3.3.1 Model structure

The topology of a LSTM network is highly malleable and is mostly up to the user to arrange freely like any other neural network layer.

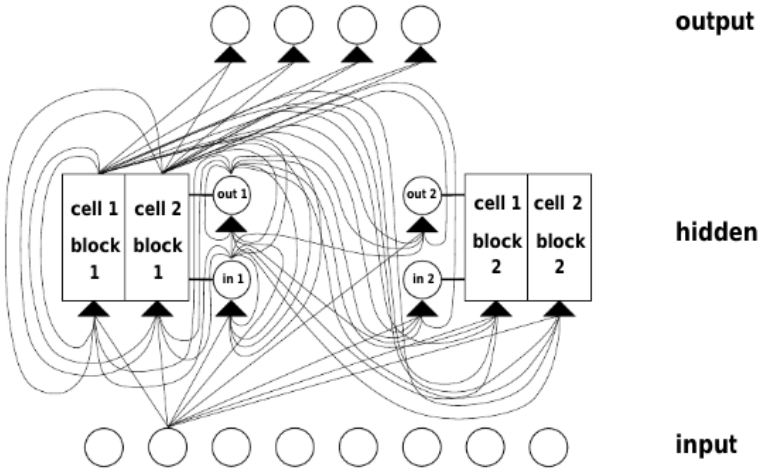

Figure 2: Example topology of a LSTM network layer.[2]

3.3.2 Gates and Memory cells

In a neural network, a memory cell is a storage unit able to store information over time, allowing the network to learn and make decisions

[2]Hochreiter and Schmidhuber [1997]

based on past events. A group of memory units enclosed by a input and output is called a *memory block*.

These inputs and outputs are called gates. There are two types of gates in a LSTM model. A multiplicative *input* gate and a multiplicative *output* gate. These gates are located at the input and output of the previously mentioned memory storage blocks. The *input* gates mainly shields the memory blocks from perturbation by irrelevant inputs. And the *output* gates analogously protect the other cells from being influenced by currently irrelevant information stored in the memory blocks. The combined unit of input, output and memory is called a *memory cell* and is integral to the functionality of the LSTM neural network.

The memory cell avoids conflicting input weights by allowing the input cell to control error flow into the memory cell and the output cell to control error flow from the memory cell. This allows the gates to control when to keep and/or override information in the cell as well as control when the rest of the network can access the contents of the memory cell.

Errors inside the memory cell cannot change, but additional information flowing into the cell at a different point in time can superimpose with the constant errors inside the cell. The output cell therefore has to learn when to trap errors and the input cell has to learn when to release errors. Both of these actions are done by multiplicative scaling (amplification or reduction) of error flow.

3.3.3 Information Peephole

In Hochreiter and Schmidhuber [1997] the concept of a *information peephole* is also introduced and explored as a part of the LSTM network. This peephole is an extra connection between the *input*, *output* and *memory* cell. This connection allows for the gates to use the current state of the memory cell as an additional variable together with their respective input weights when determining the future state of the memory cell. This has been proven to improve the LSTM networks ability to make decisions based on long-term dependencies and is a standard component in most modern LSTM implementations.

3.4 Bayesian Neural Networks

In a neural network, the weights fully determine the connections between each layer. These weights can be defined in a multitude of ways.

In Mackay [1991] the groundwork for using Bayesian probability theory to define the weights of a neural network is laid out. The orthodox way to build a neural network is to use static weights and backpropagation in training as per Rumelhart et al. [1986].

The key idea in Bayesian neural networks is to view the weights of a neural network as random variables, rather than numerical values, and to use Bayesian inference to produce a posterior distribution fitted to the trained data. Using a Gaussian distribution as the prior, the conjugate gradient method can be used to approximate the posterior.

Calculating the loss for Bayesian weights can be a challenging task and often require extraction of statistical properties of the posterior. Using the evidence lower bound (ELBO) loss one can measure the performance of the posterior and train a Bayesian neural network.

The Bayesian implementation of a neural network comes with some inherent benefits. The user can incorporate previous knowledge about the distribution of the weights in the learning process and the model is often more robust to overfitting compared to a normal implementation because of the lack of single point estimates. The Bayesian model also provides a way to estimate the uncertainty of the output, a feature which is usable in our case where we can make sure to only *take action* in the market when the model is gives an answer with a high probability of success.

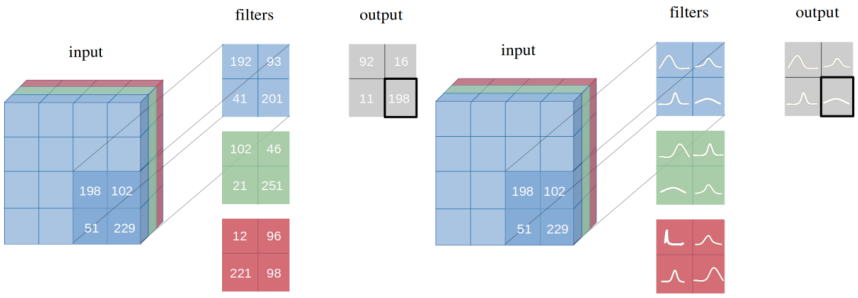

Figure 3: Static weights (left) in a Convolutional Neural Network compared to Illustration of the Bayesian weights (right).[3]

[3]https://github.com/giacomodeodato/vinn (February 15, 2023)

3.4.1 Bayesian LSTM

The application of Bayesian LSTMs have previously only seen limited research in the application of stock market predictions and this study explores the possible usefulness of this kind of implementation. The Bayesian implementation allows for a way to estimate the uncertainty in model parameters, which in turn allows the model to select the most informative samples as well as discard samples with high uncertainty. Since the output of the model is probabilistic, multiple predictions need to be made on the same input. If the resulting median of its certainty is below a specified threshold, that sample will not be considered. Several architectures were investigated, however the one presented here consisted of just one LSTM layer with 40 hidden units and one linear layer and was highlighted due to its robustness in terms of delivering stable results.

3.5 Transformer

The transformer-based model in this project is heavily inspired by original transformer method first introduced in Vaswani et al. [2017]. In contrast to LSTM and many other models used in time-series classification, the transformer is not a recurrent neural network but rather relies on an attention mechanism. The transformer was not the first model to utilize the attention mechanism but the first to solely rely on it. See Figure 4 for the model architecture used in this project. Each part of the network will be explained in parts.

3.5.1 Model architecture

The model used in this paper consist of an initial time-embedding layer followed by multiple attention-blocks, see Figure 4 which is then followed by a *linear* layer and a final *softmax* layer. This is heavily inspired by the encoder parts of the transformer model proposed in Vaswani et al. [2017]. Each attention-block contains skips-connections to avoid vanishing gradient and also layer-normalization as described in Ba et al. [2016]. Much like batch-normalization, layer-normalization has been found to decrease training times but has the added benefit of being more fitting for RNN:s and transformer models.

3.5.2 Time Embedding

The purpose of the time embedding layer is to give the transformer a notion of time. It makes intuitively sense that recent stock movements have a bigger influence on current stock movement. The transformer

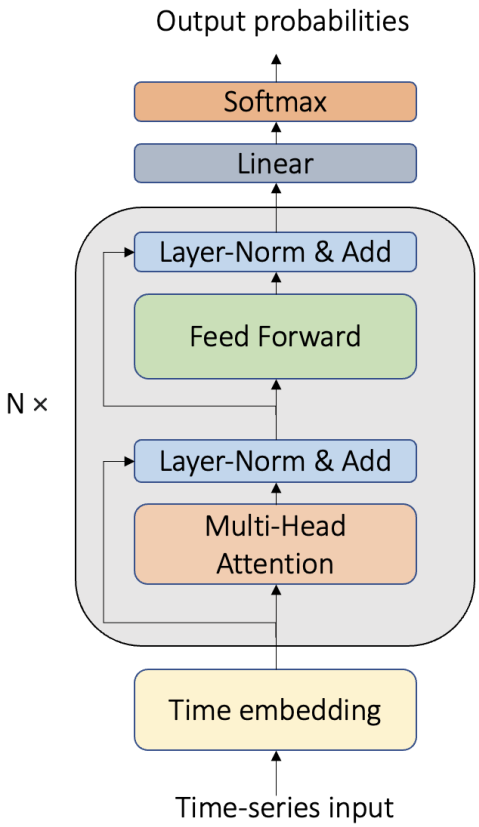

Figure 4: The proposed transformer model architecture used in this work.

does however look at the entire time-series at the same time and might treat earlier stock movement the same as more recent. To avoid this, the time embedding proposed in Kazemi et al. [2019], Time2Vec is used. The Time2Vec embedding layer tries to capture both the periodic and the non-periodic patterns by using the following expression

$$t2v(\tau)[i] = \begin{cases} \omega_i \tau + \phi_i, & \text{if } i = 0 \\ F(\omega_i \tau + \phi_i), & \text{it } 1 < i < k \end{cases} \tag{3}$$

where $F(x) = sin(x)$, τ is the time-series and $t2v(\tau)[i]$ is the i^{th} element. $\omega_i \tau + \phi_i$ is set to capture the non-periodic time-features and $F(\omega_i \tau + \phi)$ the non-periodic by using a `sin` activation function. Both ω_i and ϕ_i are learnable parameters. The result of the using Time2Vec is a vector representation of the periodic and non-perodic time-features.

It should be noted that Time2Vec is not applied to all time-series

features but only on the closing price (since that is what we are trying to predict). The time embedding is performed on the closing price and the resulting periodic and non-periodic features is concatenated to the original time-series and therefore contains 18 features instead of 16 after the time embedding.

3.5.3 Attention

Attention or self-attention is the main working principle of the transformer. There are multiple ways of implementing attention into a neural network but in our model we utilize "Multi-head attention" which was proposed in Vaswani et al. [2017]. This method is a extension of the Scaled Dot-Product Attention method. The idea behind self-attention is to capture dependencies within a sequence and teach the model how different stock movements relate to each other. The purpose is to give the neural network a sense of what parts of the time-series sequence is important to focus on and which parts that may not be as important when making a prediction.

See Figure 5 for a diagram for the scaled dot-product attention method and equation (5) for the equation.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{4}$$

The input of attention mechanism consists of 3 parts; Query, Key and Value (denoted Q, K & V). All of these inputs have been created by running them through 3 separate dense layers with trainable parameters. The query is then multiplied with the transposed keys which results in an attention matrix. The attention matrix predicts which areas of the time-series neural networks focuses on. The attention matrix is then divided by length of the initial dense layer (not shown in Figure 2) to avoid exploding gradients and then run through a softmax function. The attention matrix is then multiplied with the value matrix. The resulting matrix is the value matrix but with the values being proportional to the attention matrix. This matrix is called the attention weights which is then propagated through the rest of the neural network.

Multi-head attention, which was used in our model, is a extension of the scaled dot-product attention. The basic scaled dot-product attention could therefore be seen as single-head attention. With multi-head attention, the same querys, keys and value are used as previously but are divided by features and then fed into multiple different attention

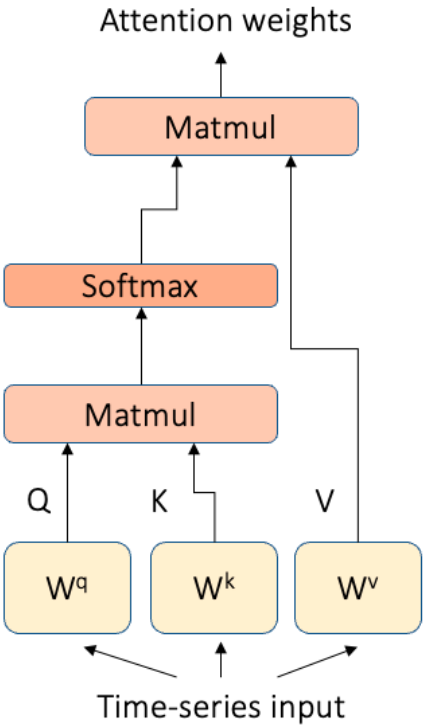

Figure 5: Transformer model proposed in Vaswani et al. [2017]

layers. These different attention layers are called *heads*. These heads are able to pick up different patterns and dependencies. After the heads are passed the attention matrix is concatenated back to the correct output dimension. The attention matrix is then passed through a dense layer and then passed throughout the rest of the network.

3.5.4 Differences from original transformer model

The application of the transformer model propose in Vaswani et al. [2017] is machine translation. Since translation is a sequence transduction task with the output being a sequence of vectors our implementation was adjusted such the output was a single vector containing our prediction. This means that the decoder was removed from the transformer model in Vaswani et al. [2017] since that is what is used to create an output sequence.

4 Results

To evaluate the performance of the neural networks, two different cases were considered:

- Two stocks from different sectors to minimize the correlation with each other: Amazon (AMZN) and Procter & Gamble (PG) together with their sector indices and the S&P 500 index with hourly data from the last 2 years.
- The S&P 500 index with hourly data between 2009 and 2022.

The Bayesian LSTM model solves a binary classifiction problem and predicts if the stock/index is expected to up or down. As the binary classification problem was not suited for the transformer, it instead predicts if the stock is expected to go up, sideways or down with sideways referring to no meaningful movement.

4.1 Bayesian LSTM

To evaluate the performance of the Bayesian LSTM model the accuracy was calculated and multiple runs were performed to ensure stability of results. The results from a single run are also visualized in confusion matrices to display how the predictions were distributed among the two classes. Due to the probabilistic output, 100 repetitions of predicting a given sample were conducted on each sample to calculate the median of the result. A threshold was applied to remove samples that the model was less sure of. In general the higher the threshold the higher the accuracy but at the cost of less eligible samples. In the results shown below the threshold was chosen to discard the bottom two thirds of samples.

The network was trained with the Adam optimizer using a learning rate of 0.001 and a batch-size of 16. Early stopping was practiced when the validation loss did not decrease any further after 20 warm-up steps.

4.1.1 AMZN Stock

For the Amazon stock, an accuracy of 58.36% was achieved with a threshold. As seen in the table 1, the threshold significantly improves the performance. The models predictions for a single run are not completely evenly distributed among the two classes. One can easily see from the confusion matrix, table 2, that the model performs better on the up class.

Table 1: Results for predicting movement of AMZN stock over 5 runs.

	Mean accuracy	**Standard deviation**
Under threshold	0.5265	0.0075
% of samples over threshold	0.3421	0.0915
Over threshold	0.5836	0.0147

Table 2: Confusion matrix for the results of a single run on AMZN stock with threshold.

	True Down	**True Up**
% of samples over threshold	61	50
Predicted Up	59	85

4.1.2 PG Stock

The P&G stock falls short of Amazon stock with an accuracy of 56.29% with a threshold , see table 3. From table 4, one can also see that for a single run that the model favor somewhat the up class, however the model managed to get a higher accuracy for the down class.

Table 3: Results for predicting movement of PG stock over 5 runs.

	Mean accuracy	**Standard deviation**
Under threshold	0.5047	0.0062
% of samples over threshold	0.3038	0.0429
Over threshold	0.5629	0.0203

Table 4: Confusion matrix for the results of a single run on the PG stock.

	True Down	**True Up**
Predicted Down	93	61
Predicted Up	73	89

4.1.3 S&P 500 index

A similar accuracy was achieved with threshold on the S&P 500 index as before, see table 5. From table 6 one can see that the predictions are not evenly distributed among the two classes as the down class is heavily favored.

Table 5: Results for predicting movement of S&P 500 index over 5 runs.

	Mean accuracy	**Standard deviation**
Under threshold	0.4887	0.0121
% of samples over threshold	0.2803	0.0519
Over threshold	0.5628	0.0117

Table 6: Confusion matrix for the results of a single run on the S&P 500 index.

	True Down	**True Up**
Predicted Down	1775	1370
Predicted Up	626	740

4.2 Transformer

To evaluate the performance of the transformer the accuracy for each class was calculated since the total accuracy can be misleading when the classifier contains 3 classes. Multiple runs were performed to ensure stability of results. We also visualize the confusion from a single run to show how the predictions are distributed among the classes.

For testing a batch-size of 16 was used and the Adam optimizer was used with the parameters $\beta_1 = 0.9$ and $\beta_2 = 0.98$. The learning rate was scheduled the same as in Vaswani et al. [2017]:

$$l_{rate} = d_{model}^{-0.5} \cdot min(S_n^{-0,5}, S_n \cdot W_s^{-1.5}) \tag{5}$$

Where d_{model} is the dimension of the model, S_n is the step number and W_s is the warm up step. As with LSTM, early stopping was used to ensure neural network was not overfitted. A label-smoothing of 0.2 was also used.

4.2.1 AMZN Stock

For the Amazon stock a total accuracy of 42.36% was achieved with accuracy being quite distributed among the three 3 classes, see table 7. The sideways class manages to achieve a slightly higher accuracy of about 50%.

Table 7: Results for predicting movement of AMZN stock over 5 runs.

Class	Mean accuracy	Standard deviation
Down	0.3958	0.0580
Sideways	0.4904	0.0245
Up	0.4100	0.0087
Total	0.4236	0.0205

Table 8: Confusion matrix for the results of a single run on the AMZN stock.

	True Down	True Sideways	True Up
Predicted Down	99	31	84
Predicted Sideways	58	63	14
Predicted Up	82	43	89

4.2.2 PG Stock

The P&G stock reaches a similar result as the Amazon stock with a total accuracy of 41.56%, see table 9. In similar fashion, the accuracy is quite evenly distributed with only the sideways class managing to reach a bit higher accuracy.

Table 9: Results for predicting movement of PG stock over 5 runs.

Class	Mean accuracy	Standard deviation
Down	0.3969	0.0442
Sideways	0.4512	0.0918
Up	0.3848	0.1180
Total	0.4156	0.0058

Table 10: Confusion matrix for the results of a single run on the PG stock.

	True Down	**True Sideways**	**True Up**
Predicted Down	93	61	67
Predicted Sideways	28	64	32
Predicted Up	73	59	89

4.2.3 S&P 500 index

While achieving a similar total accuracy as before, for the S&P 500 index the prediction accuracy is not evenly distributed and heavily favors the down and sideways class.

Table 11: Results for predicting movement of S&P 500 index over 5 runs.

Class	**Mean accuracy**	**Standard deviation**
Down	0.5299	0.0299
Sideways	0.5174	0.0327
Up	0.2236	0.0200
Total	0.4154	0.0058

Table 12: Confusion matrix for the results of a single run on the S&P 500 index.

	True Down	**True Sideways**	**True Up**
Predicted Down	3026	1458	1478
Predicted Sideways	1583	2440	491
Predicted Up	3076	1453	1413

5 Discussion

As seen in the results, both models performed better than random guessing. The accuracy may at first not seem very impressive but given the vast competition by institutional and retail traders one encounters in speculating in the stock markets, the models can be considered to be successful.

The transformer model was superior in predicting sideways movements compared to up or down movements which might be explained by less volatile periods also producing lesser price action. Perhaps the most remarkable finding is the effectiveness of increasing the accuracy by implementing a Bayesian approach to the LSTM model and could be a field worth examine further in future research. There were limitations on accessing free market data. The further back in time, the lower the resolution of data. It is plausible that better results could be achieved on a larger data set containing 16 features.

There are shortcomings with only predicting the direction of the price of the next data point. For instance, even if an established trend is present, which may give a hint of the overall price movement in the near future, the direction of the next price movement does not necessarily have to align with the trend, due to the stochastic nature of the stock market. Also, the initial direction of the stock movement might be correctly guessed but then reversing in the same time period.

One way to increase the performance of our models is to simplify the problem. Predicting the actual stock price is widely considered an easier task than predicting its movement. This however is usually not that useful since that predicted price is usually very close to the price of the previous hour. Such an algorithm is also not that interesting from an investors point of view. Another way to simplify the problem is to apply some smoothing effect on the training data which will help the network find patterns. This does however also limit the actual usefulness of the model.

Initially the transformer model was constructed to solve a binary classification problem predicting if the stock market is expected to go up or down. However it became clear that this task was not suitable for the transformer as it would get stuck in a local minima and only predict up or down. The solution was to introduce a third class, the sideways class which refers to no meaningful stock movement. This solved the problem of the network constantly predicting the same class but unfortunately makes it hard to compare the performance of the Bayesian LSTM and the transformer model.

6 Future Improvements

There are many approaches in trying to improve the model. Something that was considered but not implemented was transfer learning. Transfer learning essentially means that the weights of a pre-trained model is used to simplify the training. Transfer learning is known to increase performance in many neural network as longs as the underlying model is the right fit for the given task Zhuang et al. [2021].

The selection process of the threshold which the Bayesian LSTM model rejects or accepts samples could be improved. The performance did in general increase with a rising threshold up to a certain point where the accuracy declined. However, that point could differ between the data sets. As an enhancement, while training the network, multiple thresholds could be tested on the validation data after each epoch, to obtain a perception of an optimal boundary.

In the current models, only one stock at a time is considered for training the networks on and predicting. Let say that most of the time that a stock might have a stochastic movement, thus making forecasting meaningless. High probability setups for predicting right may still occur, just very rarely. If one instead, for each time step, had a vast selection of stocks to choose from, the model could just predict on the stock that has highest current probability based on some type of scoring system.

It is also possible that, while very effective for language processing, the base transformer model is not well suited for this time-series problem. A major change in architecture while still utilize the attention-mechanism might increase performance for the transformer model. The Bayesian LSTM model could also benefit from a modified architecture, since the model presented here were rather primitive.

Since stock markets are not solely influenced by historical data, one approach to increase the performance of the neural networks, is to incorporate current news and events regarding a stock or areas connected to it. This could be done via extracting information from news articles, Twitter, Reddit and so forth via sentiment analysis. We believe that such an approach is necessary to reliably increase the performance of the model.

7 Acknowledgements

We would like to thank Prashant Singh for continuous support during the carrying out of this project, supplying professional help and expertise in the field of neural networks and machine learning.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181.

Pushpendu Ghosh, Ariel Neufeld, and Jajati Keshari Sahoo. Forecasting directional movements of stock prices for intraday trading using lstm and random forests, 2020. URL https://arxiv.org/abs/2004.10178.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. Time2vec: Learning a vector representation of time, 2019. URL https://arxiv.org/abs/1907.05321.

David J. C. Mackay. A practical bayesian framework for backprop networks. *Neural Computation*, 1991.

Venkata Sasank Pagolu, Kamal Nayan Reddy Challa, Ganapati Panda, and Babita Majhi. Sentiment analysis of twitter data for predicting stock market movements, 2016. URL https://arxiv.org/abs/1610.09225.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323: 533–536, 1986.

Dr. P. K. Sahoo and Mr. Krishna Charlapally. Stock price prediction using regression analysis. *International Journal of Scientific Engineering Research*, 6(3):1655–1659, 2015.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017. URL https://arxiv.org/pdf/1706.03762.pdf.

Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2021. doi: 10.1109/JPROC.2020.3004555.