Deep Neural Networks

September 2021

Project

HELLENIC REPUBLIC

National and Kapodistrian University of Athens

Anastasia Rempoulaki (ID: DS1.20.0016)

Athanasios Polydoros (ID: DS1.19.0016)

Ilias Karabasis (ID: DS1.20.0005)

Supervisors

Dr. Haris Papageorgiou

Athanasia Kolovou

Table of Contents

Introduction

Stock price prediction is a popular problem that economists try to solve by applying technical analysis over companies and/or detecting known patterns in the stock price time series. Trading stocks is a common way of investing and a hobby for several people globally. There are several types of traders, with the main difference being on the period between the buy and sell order of an asset and the frequency of the trades.

The day traders are the one who buys and sells an asset in short periods looking to make a profit on the volatility of the market (from several minutes to several hours), while the long term traders act on a longer-term trying to maximize the earning based on company's vision, earnings or hype (from several days to months). Day traders use the intraday stock price in order to call on sell or buy actions while the long term traders mainly use the closing price of the stock for a reference. However, all the traders have the same goal, buy low and sell at higher prices.

Most people argue on whether the price of a stock can be predicted only by using the previous values since it's a more complicated problem that depends on several parameters (news, politics, performance of the company etc). Other solutions are trying to approach this problem by combining input from different sources including sentiment analysis on news and tweets, financial data of the stock or correlation between similar stocks.

We will try to apply deep neural networks techniques in order to predict the closing price of a stock just by using previous closing values of the stock for input.

Our project (Vision/Goals)

The goal of the project is to try different deep neural networks techniques in order to predict the closing price of the stock on the N+1 day given the previous N days closing price. We try to understand whether the price of the stock can be predicted by using its previous values and whether the use of deep neural networks is a great solution for this task.

The main techniques that will be compared are three. First, a simple neural network composed of fully connected layers.

Secondly, a Long-Short-Term memory network will be used which is a specific type of recurrent neural network (RNN). RNNs are suitable for predicting prices from time-series data.

Finally, a more sophisticated RNN, Random Long Short-Term Memory (RLSTM) [1] will be used which is a recently suggested solution for the problem of stock price prediction. Although we are going to describe the model in more detail later on, it is a newly (paper publication May 2021) introduced deep learning model aiming to mitigate weak generalization ability and overfitting in training. RLSTM is based on LSTM includes a prediction module, a prevention module and three fully connected layers. The input of the prediction module can be a stock or an index that needs to be predicted. That of the prevention module is a series of random data with uniform distribution.

All models will be trained over the same dataset (see Data Collection and Dataset Overview) and tested using different parameters. More details about each model can be found in the Models section.

The notebooks that have been used can be found on GitHub.

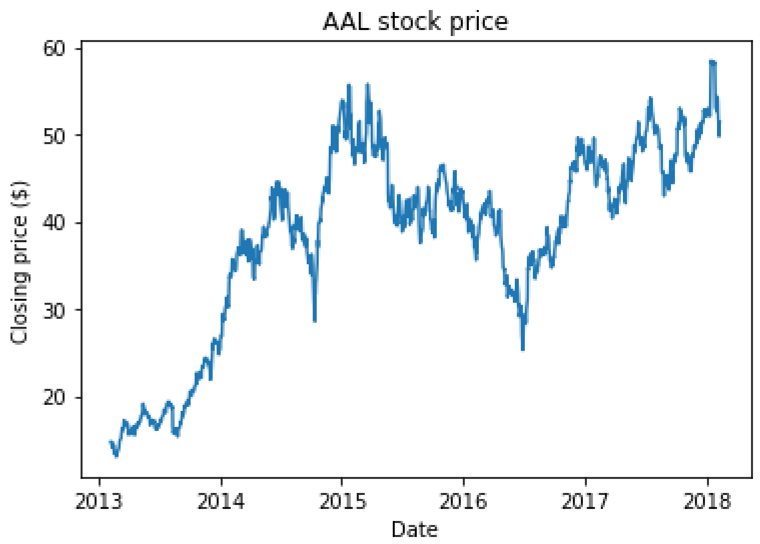
Data Collection and Dataset Overview

The dataset that has been used contains all stocks that are included in S&P 500 during 2013 -2018. S&P 500 is a list that contains the 500 biggest companies in the New York stock market exchange.

The dataset can be found easily online by most of the major financial websites. We have used data found on Kaggle.

Each row in the dataset holds data about a single day that contains the Name of the stock, open price, close price, highest price, lowest price, date and volume. We will only use the name and the close price.

This is the closing price graph of AAL stock between the above-mentioned period.



Some of the stocks have been added later in the index (new companies entering the index, or companies might close/rename) which means that some of the data might be missing.
There are 619032 rows with a mean closing price of 83$ (ranging from 1.59$ to 2049$) containing data for 505 different companies.

Data Processing

There are two main preprocessing steps that will be common across all models. The first is the regularization of the closing prices, while the other is a conversation of raw data into the input and expected output of the models.

The dataset contains the closing price of each stock in $(US dollars). It's common knowledge that neural networks perform best when fed with data within the range of 0 to 1. We normalize the closing price of the stock to be within this range by applying a Min-Max normalization.

Min-max normalization is one of the most common ways to normalize data. For every feature, the minimum value of that feature gets transformed into a 0, the maximum value gets transformed into a 1, and every other value gets transformed into a decimal between 0 and 1.

In our case, the closing price of the whole dataset ranges from 1.59$ to 2049$.
When training with one stock we use the min and max price of the stock over the train set and apply the min-max normalization over each data set separately (validation and test set). However, this might add some bias to our model since the min and the max price are given to the model.

Finally, we need to pre-process the raw data to input and output for the models. In order to achieve this, we are sliding a window of size N over the time series of the closing price using N values as input to our model and the N+1 value as the expected output. Then we slide the window by 1 and apply the same slicing.

For example given the window size to be 10 and having an array that contains value from [0..100], the first input to our model will be:

1. X = [0..9]   and Y = 10
2. X = [1..10] and Y = 11
3. X = [2..11] and Y = 12
4. …

When using the models for inference the same process should be applied. The data should be normalized using the min and max of the provided data (not the trained data). Similarly, the sliding window technique should be applied again using the same window as train data.

In order to translate the output of the model to the real stock price, the reverse Min-max normalization should take place. Simply multiply the output O with the max-min and then add the minimum price. **O * (max-min) + min**

Models

This section will focus on the topology and a brief description of the models that will be compared.

Simple Neural Network

Regarding the first model, the topology is quite straightforward.
The network consists of 2 dense layers and an output layer has been used.
The 2 dense layers consist of 64 units with RELU activation function. The output layer is a single fully connected that outputs the normalized price of the stock. As loss function, while training the mean squared error has been used alongside with adam optimizer for backpropagation.
The input of the network depends on the size of the window (as mentioned above) while there is always a single output.
As mentioned in several articles and papers, a feedforward neural network on stock price prediction based on time series doesn’t perform well on unseen data. After experimenting with different neural networks we found that there is no significant difference in the output. Thus, only a single setup has been used.

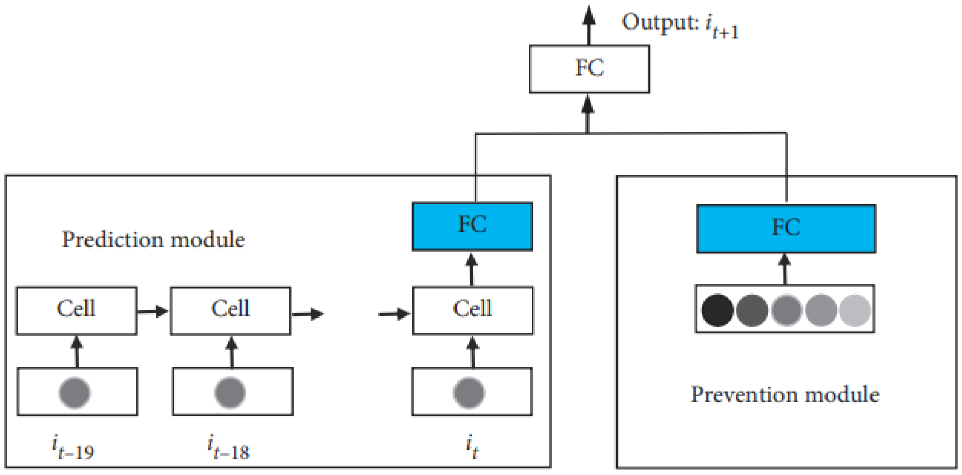
LSTM

LSTM is a network model proposed by Schmidhuber et al. in 1997. It is a network model designed to solve the longstanding problems of gradient explosion and gradient disappearance in RNN. It has been widely used in speech recognition, emotional analysis and text analysis because this model has its own memory and can make relatively accurate forecasting. As we saw in the previous model, a traditional neural network also known as a feed-forward neural network, it has its input layer, the hidden layer and the output layer. An RNN has a looping mechanism that acts as a highway to allow information to flow from one step to the next. This information is the hidden state, which is a representation of previous inputs. The LSTM can solve the problem of short memory traditional RNN has.
More specifically, in our case in order to build our model, some modules from Keras were imported. Firstly the Sequential for initializing the neural network, Dense in order to add a densely connected neural network layer, LSTM for the Long Short-Term Memory layer and Dropout for adding dropout layers for preventing overfitting.
Firstly, we add an LSTM layer. For the LSTM layer, we choose 50 units which is the dimensionality of the output, return_sequences = True which choose if the full sequence will be returned or the last output and finally we include the input_shape as the shape of our training set.
Then, for Dropout layers, we choose 0.2. This means that 20% of the layers will be dropped.
Then we add the Dense layer.

Our model is compiled with the adam optimizer and the loss is set as mean_squared_error. This function estimates the mean of the squared errors. The model runs for 100 epochs and the batch size is 32.

RLSTM

Random Long-Short-Term Memory (RLSTM) is a model architecture proposed by Hongying Zheng, Zhiqiang Zhou and Jianyong Chen at [1] which improves the generalization ability of a single LSTM through adding an additional module described as “prevention module”. The latter provides random variability so that the model can show better performance of stock prediction. The architecture of RLSTM can be seen in the below figure:



As we can see it contains two parts. One is the prediction module which is composed of an LSTM and a fully connected network layer. The input of this module is the prices of the stock we need to predict. The other is the prevention module which is only a fully connected network layer. The input of the second module is random data that are randomly extracted from the uniform distribution within the range between the highest and the lowest prices of the target stock. Above the two modules, there is the third fully connected layer in the architecture. Its inputs are just the outputs of the two modules. The final result of the architecture can be obtained after the computation of the third fully connected layer. A linear activation function is used in all three fully connected layers.

The randomness of the input data in the prevention data has no relation with the real stock price, thus it can be seen as noise in RLSTM. During the training stage, the noise interferes with the model which can help against overfitting for the LSTM module.

The above architecture is not entirely a breakthrough. It is similar to the ModAugNet-c. [2] ModAugNet-c also requires additional reinforced data to aid the target stock data. The reinforced data is composed of some other stock data which have a strong correlation with the target stock data. Such selection of the reinforced data highly depends on personal experience. Different persons may have different selections and thus may result in different performance of

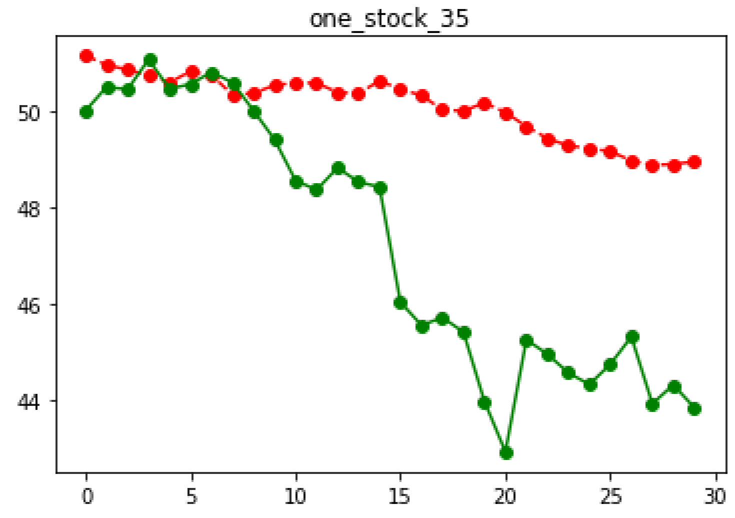
training. However, for RLSTM, only the target stock data is required for both training and prediction. The training data for RLSTM are definite and objective which need not be selected from a dataset artificially by experimenters. Another important factor differentiating these two similar architectures is that the space and time complexity of RLSTM is smaller as described in [1].
During the training process, we have followed the same one described in [1]. Mean square loss is used as a loss function, while Adam optimizer is used to optimize the model. Early stopping is used to prevent overfitting. The learning rate is set to 0.00001, the number of hidden units in the prediction module is 32 and the batch size is 32. For the number of neurons in the fully connected layers, we also used the proposed from [1] for the S&P500 index. Details can be seen in the respective code.

Experiments – Setup, Configuration

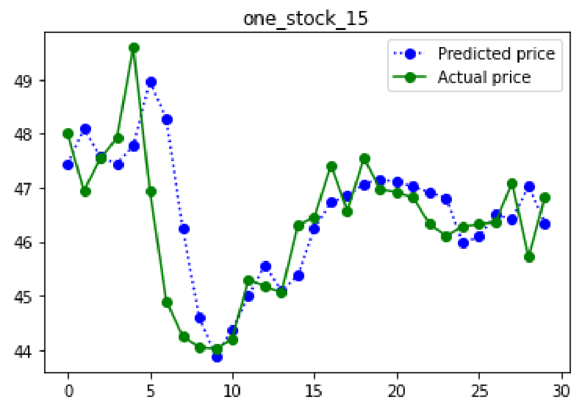
Simple NN

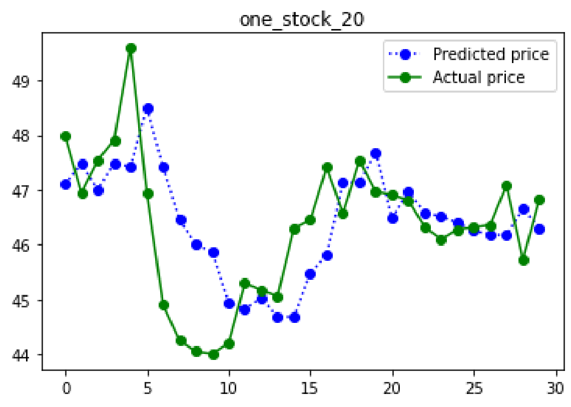
Trained and test on the same stock

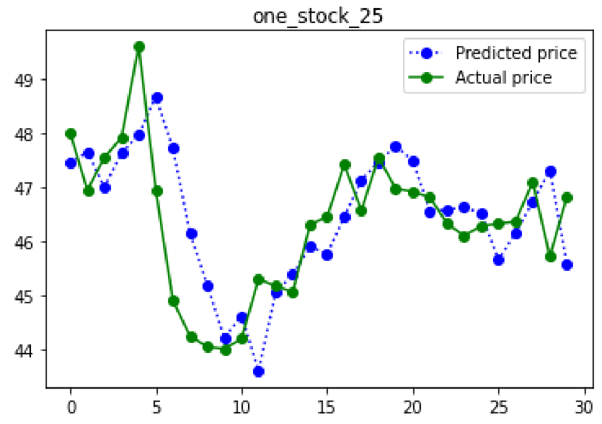
At this experiment, we will try to predict the stock price for the next 30 days given as input the previous 35 days (window size). At first, we feed the network with the known data of the previous 30 days in order to get the prediction for the 31st day. Then, we use the last 29 days, plus the predicted value, to predict the 32nd day. This way of using the model performs poorly due to the fact that the model loss will be propagated through the next results. Every unsuccessfully predicted value will affect negatively the future predictions of the stock.

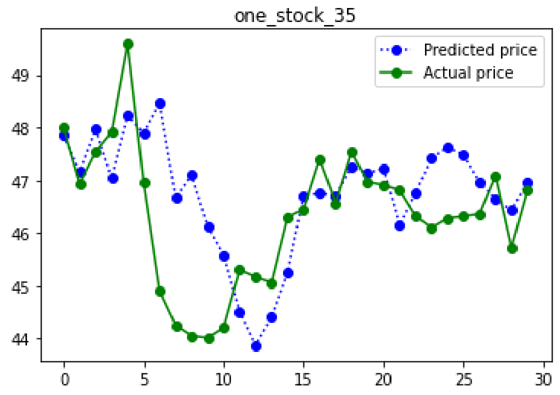


The rest of the experiments are done by using N values of the closing price of the stock as input and not the predicted one.
The first experiment is done by training the model on AAL stock and testing on unseen data of the same stock for different window values.

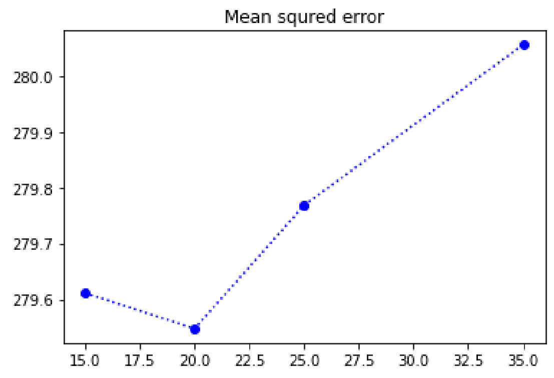




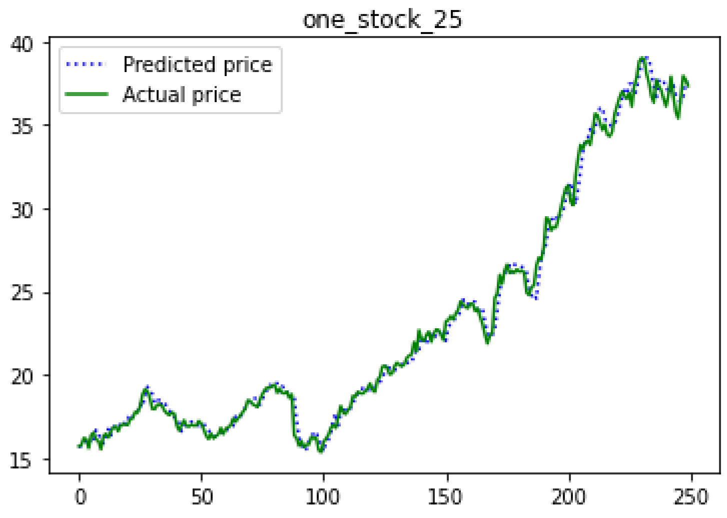




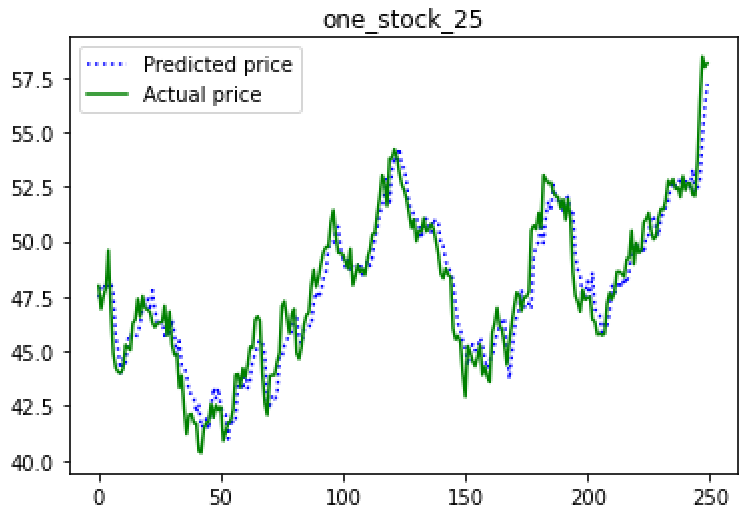
We can see from the graph the window size 20 performs best. However, the changes in the loss function are very small.



Next, we will test for window size 25 over different parts of our data.
First, we will see how the model performs on different parts of the stock that are used for training (AAL).

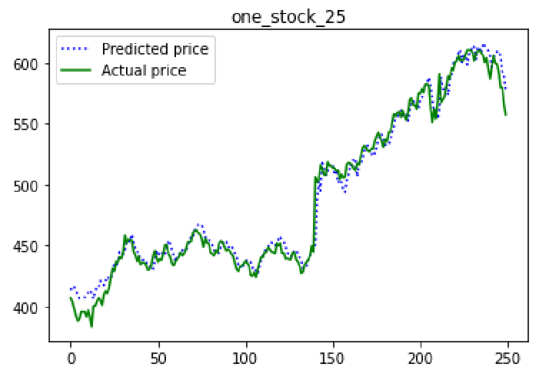


Finally, we will use unseen data from the same stock. The MSE is



Test on unknown stock

Let’s examine how the model performs to unseen data of different stocks. We will try to predict GOOGL price.



Results

	MSE	MAE	MAPE
Seen data	563.79144	22.613684	97.4674
Unseen data	2210.255	46.882046	98.04332
Other stock	166421	407.6193	99.98

The predicted prices follow the trends of the actual prices of the stock. However, the predicted values are always slightly different. The model has troubles with trend changes (from moving up for some days to switching back to moving down.

The mean squared is quite high when using stocks with higher prices (as expected). The model performs best to the data that has been trained (as expected). It performs better to the unseen data of the same stock rather than unseen data from other stock.

LSTM

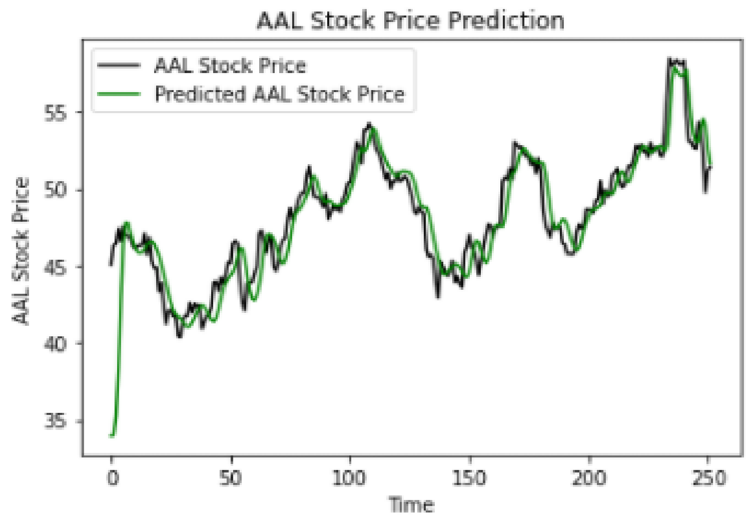
Trained and test on the same stock

In the first case we choose the AAL stock from our dataset. There is the need for our model not to over-learn from training data and to be generalizing. For this reason, we split our dataset into training, validation and testing subsets. In the first experiment, we use only one stock (AAL), therefore these subsets are from the same dataset. We chose to make the separation as follow:

- 70% of the dataset is the training subset
- 10% of the dataset is the validation subset
- 20% of the dataset is the testing subset

The training and the validation subsets were used during the fit process of the model, while the testing subsets were during the prediction process.
We used the MeanSquaredError, MeanAbsoluteError, MeanAbsolutePercentageError from TensorFlow and we have the results as follow:

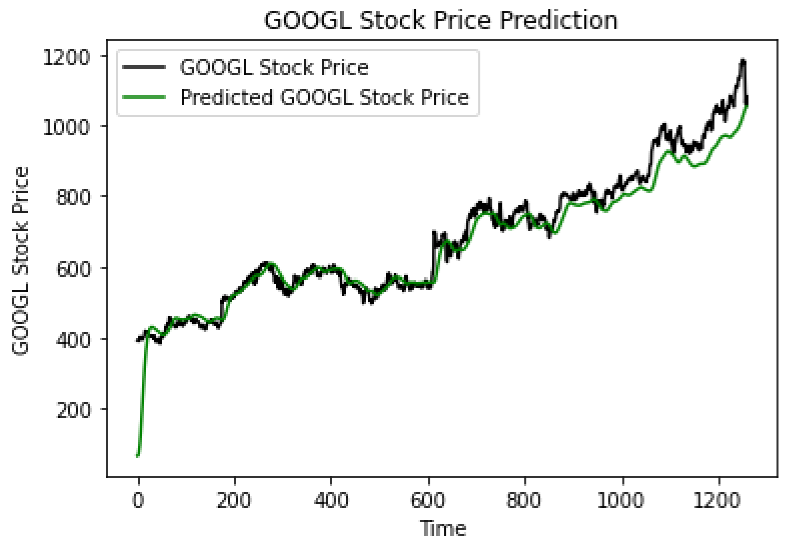
- AAL MSE 3.6569757
- AAL MAE 1.1791536
- AAL MAPE 2.481861



Test on unknown stock

In the second case, we use the AAL stock from our dataset in order to train our model and the GOOGL stock in order to make our predictions. We used again the MeanSquaredError, MeanAbsoluteError, MeanAbsolutePercentageError from TensorFlow and we have the results as follow:

- GOOGL MSE 418.25244
- GOOGL MAE 14.210677
- GOOGL MAPE 2.38179



Results

For both stocks AAL and GOOGL it is obvious that the predicted prices follow the trends of the actual prices of the stock. It is clear that there are small differences between the predicted and actual prices and the model takes a while to recognise the changes.

The mean squared error is higher when using stocks with higher prices (as expected). As we can see in the MAPE values, the GOOGL stock has a lower value even if it is an unknown stock for our model than the AAL stock.

RLSTM

Trained and test on the same stock

As in the two above models, we train our model using AAL stock. The same breakdown of the dataset is used for train, validation and test set, though here we also have the input of the prevention module from which we initially create the random data with the size of the whole dataset and similarly break them down with the same percentages as the prediction module sets (70% train set, 10% validation and 20% test set).

We trained and tested our model for different window sizes (10, 20, 30 and 50) in order to observe how this may affect the results. For the evaluation of the model with used as above three different error functions the MeanSquaredError, the MeanAbsolutePercentageError and the MeanAverageError, which describe different aspects of the model (the first its stability, the second the forecast accuracy and the third the actual situation of the predicted value error).

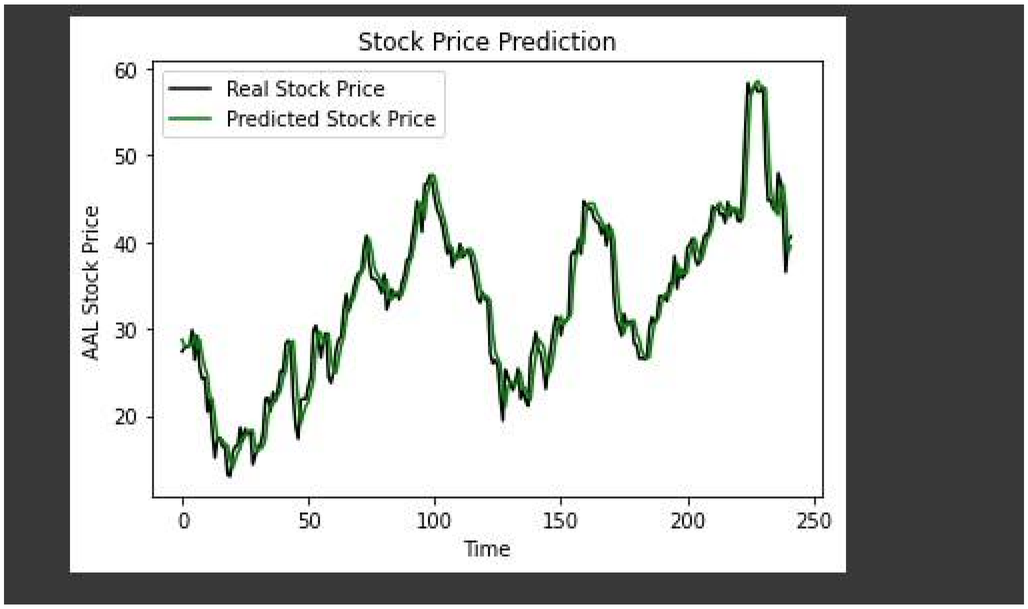


Figure: Prediction for AAL stock price for window size 10

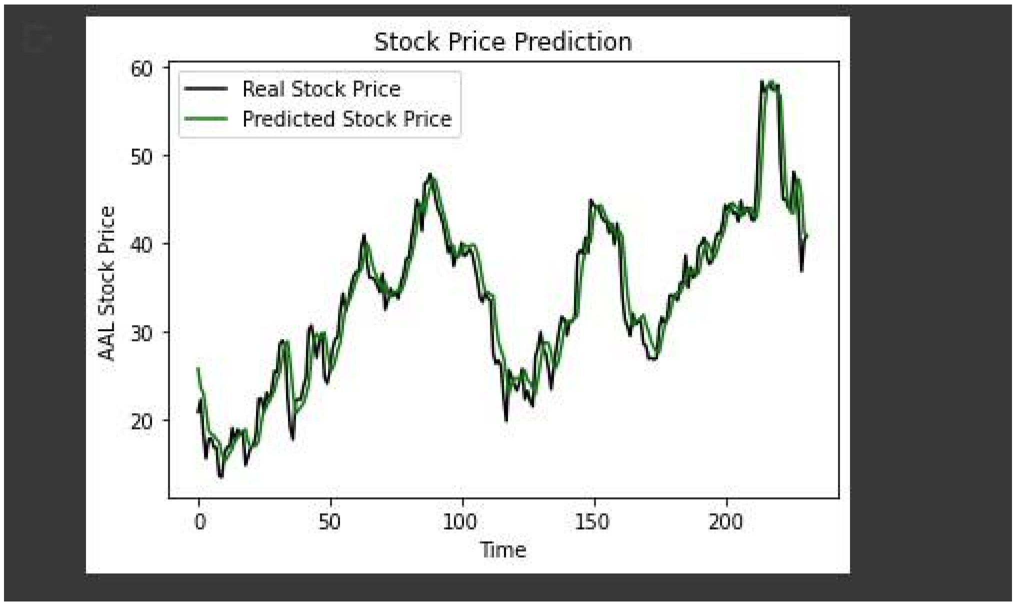


Figure: Prediction for AAL stock price for window size 20

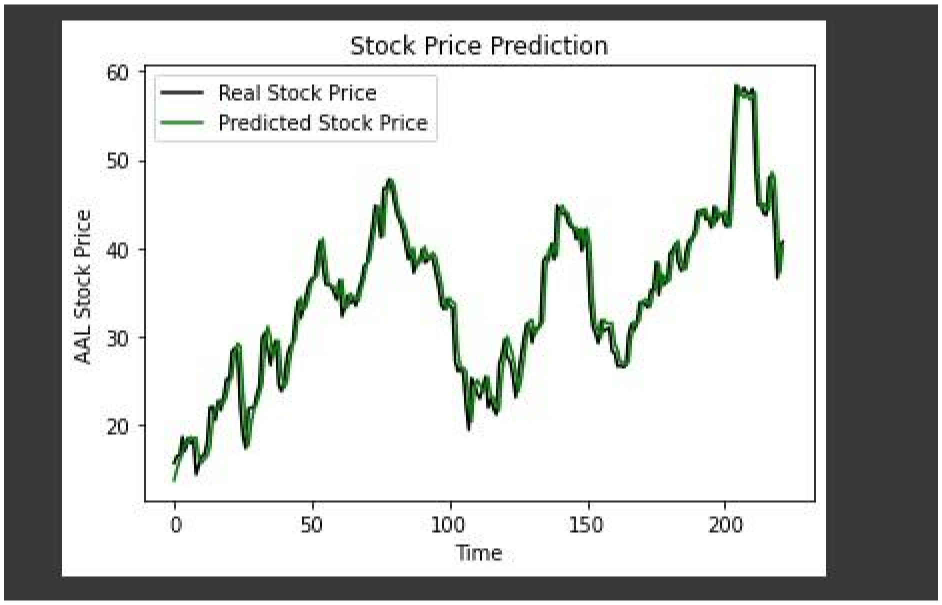


Figure: Prediction for AAL stock price for window size 30

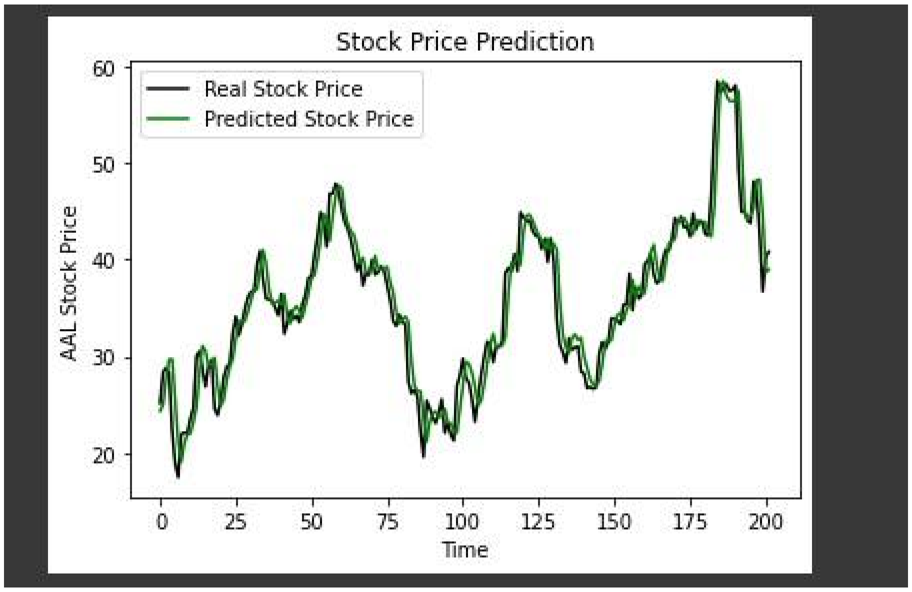


Figure: Prediction for AAL stock price for window size 50

Although it looks that the predicted values are very close to the real ones, if we “zoom in” in a smaller time period (let’s say 30 days), it becomes clear that this statement is not so accurate and in fact, there is always a slight delay for the model to follow the actual trends. This behaviour is common for all window sizes so we only display an example for the best behaving window size of 30 (days 170-199).

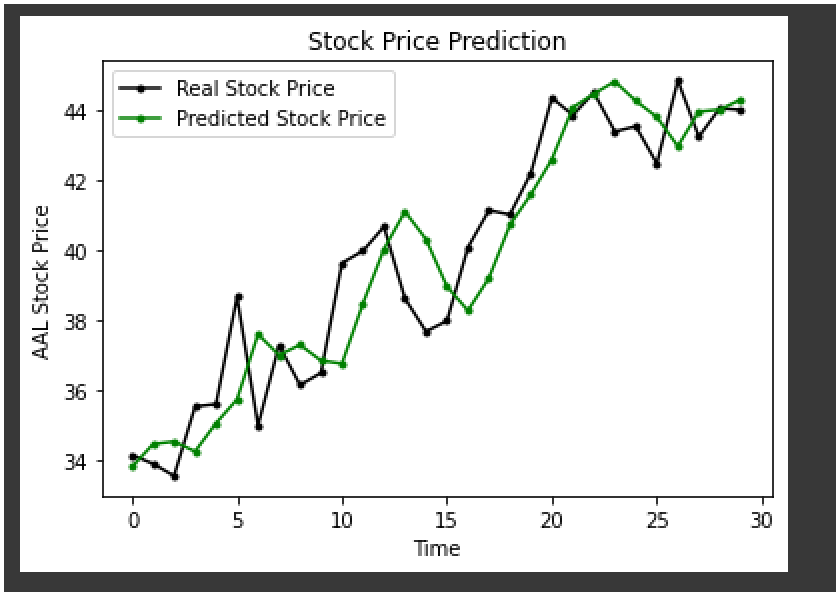


Figure: Prediction for AAL stock price for window size 30 in a “short” time period of just 30 days.

The error metrics as described above for the different window sizes can be seen in the following table. We can see that the window size 30 performs best. However, the changes in the loss function are not so significant.

Window Size	**MSE**	**MAPE**	**MAE**
10	5.988753	6.140924	1.8141321
20	6.8636127	6.520426	1.9316665
30	**5.4198327**	**5.575551**	**1.7205967**
50	6.4201994	5.7080493	1.8719776

Test on unknown stock

Again as we did for the first two models we also tested our trained model against data of an unknown stock to see how it generalizes. GOOGL stock also has a much greater value so one may see if this affects our model. Apart from this, the procedure and the tests are the same like with AAL.

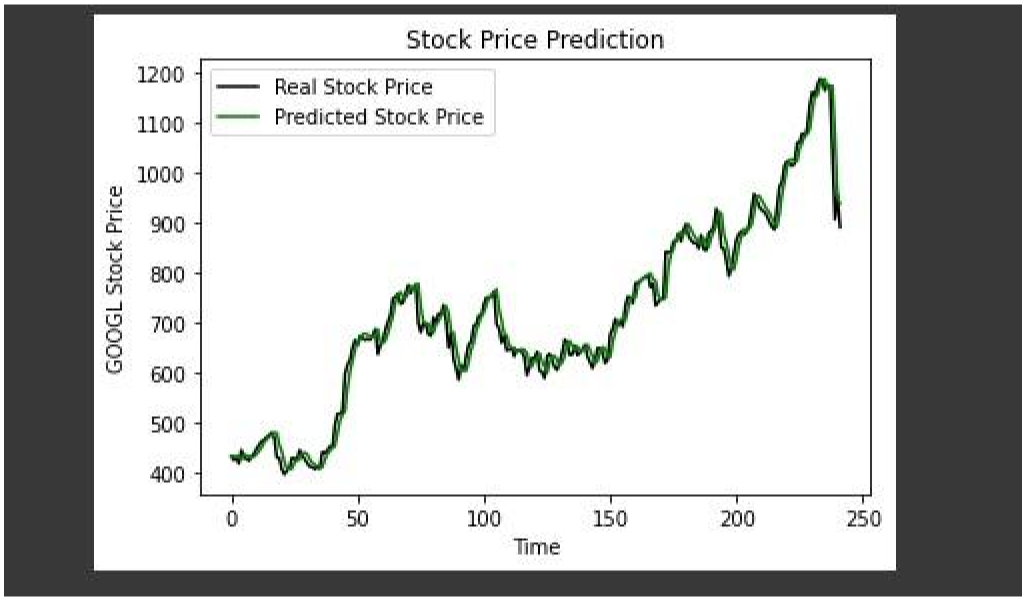


Figure: Prediction for GOOGL stock price for window size 10

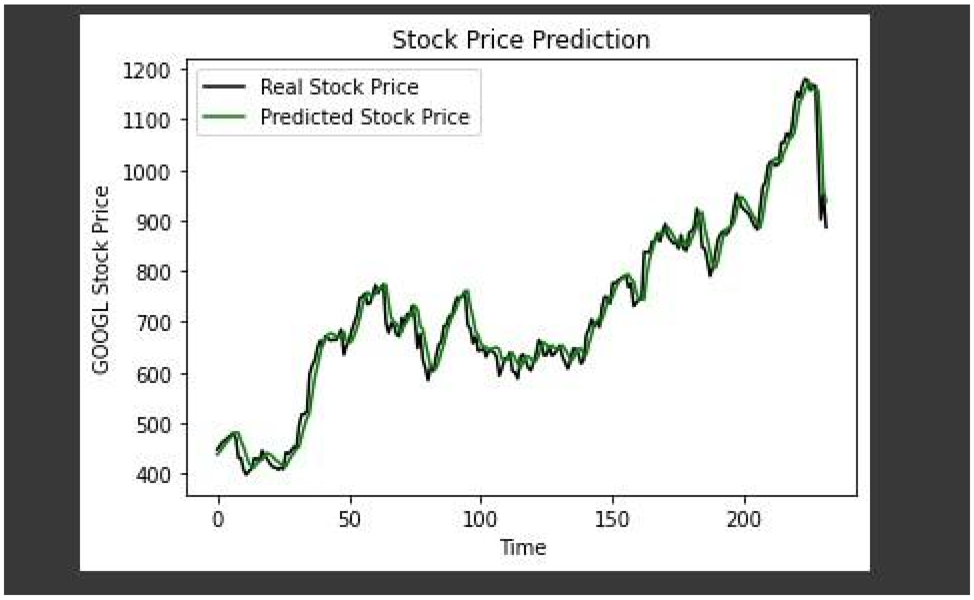


Figure: Prediction for GOOGL stock price for window size 20

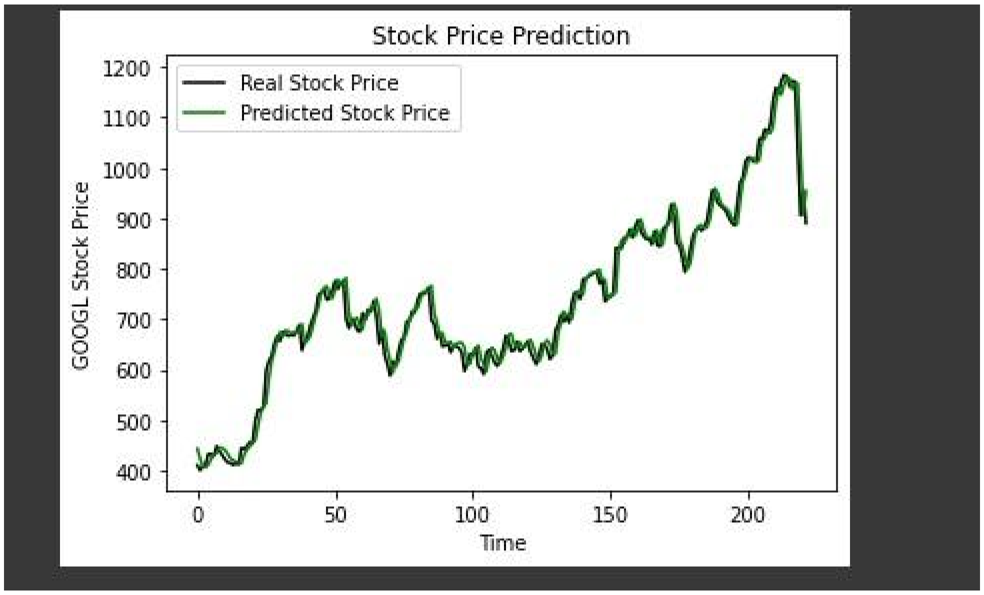


Figure: Prediction for GOOGL stock price for window size 30

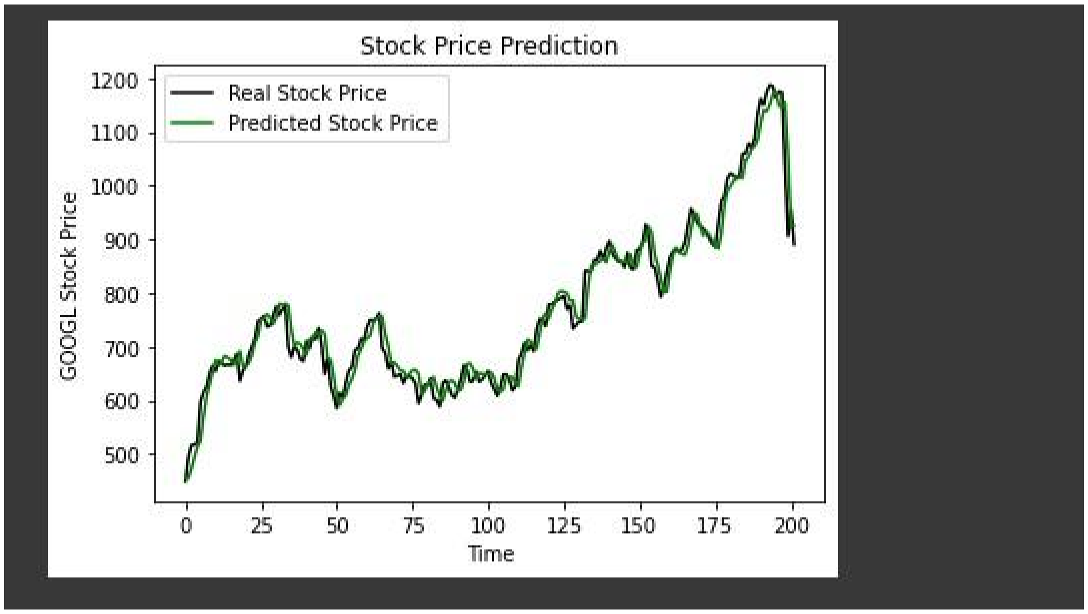


Figure: Prediction for GOOGL stock price for window size 50

What we observed for AAL with the shorter time frame also stands here, however, do not provide any graphs for the shake of space.

The error metrics as described above for the different window sizes can be seen in the following table. We can see again that the window size 30 performs best. However, again the changes in the loss function are negligible.

Window Size	**MSE**	**MAPE**	**MAE**
10	761.22406	2.7274594	18.96417
20	881.86264	3.047008	21.158346
30	**686.7477**	**2.5506752**	**18.234316**
50	846.5944	2.8268263	21.102041

Results

Generally both for the AAL and GOOGL the predicted prices follow the trends of the actual prices of the stock. However, the predicted values are always slightly different. The model has trouble identifying immediately the trend changes (from moving up for some days to switching back to moving down).

The mean squared error is higher when using stocks with higher prices (as expected). To our surprise though the MAPE error for the unknown stock is lower than the test in the unknown data of the stock it has been trained to. This is verified also from the graphs where the model looks to follow the trend of GOOGL better than AAL most of the time.

Conclusions

Regardless of the network topology, all three models managed to predict the general trends of the two stocks, some more and some less accurate. Moreover, all three models tend to slightly mispredict the exact value for the next day due to their difficulty to identify the change in the trend. Perhaps this is an indication that neural networks cannot accurately predict the price of a stock just by looking at its previous prices.

As was expected, the worst model is the simple NN. However, regarding the RLSTM and the LSTM models, we did not manage to reproduce the result of [1] where RLSTM prevailed. This possibly has to do with the fact that we used a more sophisticated LSTM architecture for the second model with multiple LSTM layers, whereas the authors may have used the simplest one. In fact, no certain info was given about it in the paper, though it seems they used just one LSTM layer. Another possible cause for this not so great behaviour of the RLSTM in our case might be that the sizes of the fully connected layers of its modules where not adapted to the specific stocks but the proposed for the general S&P500 index were used.

Future Work

One thing we could introduce to the RLSTM architecture is more LSTM layers in order to see whether we get better results. More experiments can be made with the number of neurons used in the fully connected layers of its modules in order to find the optimal ones for our training set.

The stock price prediction is a way more complex task than just predicting the next value of a time series. An interesting approach would be to provide more data about the market and include sentiment analysis on articles and tweets. Stocks are highly correlated with each other and belong to different sectors, this could be also provided as input to the model alongside some correlation between similar stocks.

Bibliography

[1] RLSTM: A New Framework of Stock Prediction by Using Random Noise for Overfitting Prevention, *Hongying Zheng, Zhiqiang Zhou and Jianyong Chen*

[2] B. Yujin and K. H. Young, “ModAugNet: a new forecasting framework for stock market index value with an overfitting prevention LSTM module and a prediction LSTM module,” Expert Systems with Applications, vol. 113, pp. 457–480, 2018.

[3]Selvin, S., Vinayakumar, R., Gopalakrishnan, E. A., Menon, V. K., & Soman, K. P. (2017, September). Stock price prediction using LSTM, RNN and CNN-sliding window model. In *2017 international conference on advances in computing, communications and informatics (icacci)* (pp. 1643-1647). IEEE.

[4]Roondiwala, M., Patel, H., & Varma, S. (2017). Predicting stock prices using LSTM. *International Journal of Science and Research (IJSR)*, *6*(4), 1754-1756.