Capstone Project. Predicting Future Stock Prices

Dias Irishev

March 2020

1 Definition

1.1 Project Overview

This project takes place in the area of Finance and Investment, and we will use some of the standard tools along with the newest machine learning tools in order to try and predict the behaviour of the stock price for the indices of our interest.

Since the stock market was invented people have tried to come up with creative solutions to "beat it". With the invention of computers people have tried to use it's computational power to try and make a leverage on the stock market. Different areas of mathematics have given the rise to the quantitative analysis, where certain smart individuals have created tools and theories to predict the behaviour of the stock markets for their benefit. Today, according to some [1], up to 70% of trading is performed through the "trade-bots" and the algorithmic trading. Unlike basic algorithms (basic linear regressions and such) today algorithmic trading is way more sophisticated and wide range of machine learning algorithms can be used for different sets of data.

Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task [2]. For this project we will explore how deep neural networks, an area of unsupervised machine learning, can be applied to try and predict future behaviour of the stock market. Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated. To be specific we will explore LSTM neural networks and how effective they can be in analysing the data and making sense of it to predict the future price. Specifics of LSTM and their difference from regular deep neural networks will be discussed later. This neural

network will be created to trained along with regular linear regression model, to see how well it performs.

The historical data on the values of the stocks will be obtained from the reliable source - Alpha Vantage, using API key. Alpha Vantage is a powerful tool to get real time stock quotes, historical data, cryptocurrencies, technical indicators, FX values and more. Alpha Vantage has most famous indicies included in S&P 500, NASDAQ and Dow Jones [3]. The data that will be used is Daily Time Series for Google index. This dataset will include such information on the stock: it's open, high, low, and close price withing the day, also volume of this stock traded.

1.2 Problem Statement

The goal of this project is to Predict stock prices using deep neural networks. Specifically, we will try to predict next day opening price of the Google stock prices based on the historical data leading to the date, by creating and training different LSTM deep neural networks and regular linear regression model. Downloaded dataset will be split into training and test sets. First dataset to train our models. Second dataset to see the performance of our models, and obtain predictions.

1.3 Metrics

To evaluate the performance of the models Mean Squared Error and R^2 will be used. Real Mean Squared Error basically measures square root of average squared error of our predictions. For each point, it calculates square difference between the predictions and the target and then average those values. The higher this value, the worse the model is. It is never negative, since we're squaring the individual prediction-wise errors before summing them, but would be zero for a perfect model[4]. Mean Squared Error can be a good evaluation metrics to see how the models perform against each other. The formula for RMSE is: $RMSE = \sqrt{(\frac{1}{n})\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$.

R^2 is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model[5]. R^2 is a scale-free indicator — it does not matter if the output values are very large or very small, the R^2 is always going to be between $-\infty$ and 1. Other way to understand, R^2 is the ration between how good our model is versus how good is the naive mean model is (which is a model that will be simply predicting y-mean value every time). R^2 is a good indicator to see how well our model performs.

Figure 1: Extract from the GOOG_daily.csv

	date	1. open	2. high	3. low	4. close	5. volume
0	2020-03-06	1277.06	1306.22	1261.05	1298.41	2660628.0
1	2020-03-05	1350.20	1358.91	1305.10	1319.04	2561288.0
2	2020-03-04	1359.23	1388.09	1343.11	1386.52	1913315.0
3	2020-03-03	1399.42	1410.15	1332.00	1341.39	2402326.0
4	2020-03-02	1351.61	1390.87	1326.82	1389.11	2431468.0

2 Analysis

2.1 Data Exploration

Historical data on the Google stock prices is available to us through AlphaVantage API service. AlphaVantage offers daily price history of Google index for the past 20 years. This includes open, high, low, close and volume of trades for each day, from today all the way back to 1999. After getting a free-api key from the website, we have created a function that does extract data in the '.csv' format. The following GOOG_daily.csv file was saved in the directory for work.

In the next step data is uploaded into pandas.DataFrame format, and we can look more closely into it. The dataset consists of 6 different columns: date, open, high, low, close, and volume values (Figure 1). These columns are self-explanitory and contain most relevant information about values of the stock on the particular day (open price, highest price, lowest price, closing price, and volume of trades for each day).

In order to get better understanding of our data we will look closer into its statistics (Figure 2). We look into the means of open, low, high, close and see that they seem to fall within a very close range between each other, nothing that raises suspicion from the first inspection. Furthermore, we look into the standard deviation of the data, and here as well, all standard deviations of open, low, high, close seem to be very close to each other. Stats for volume are a bit differently scaled due to the nature of this information (volume measures the total number of shares or contracts transacted for a specified security during a specified time period [6]), hence its statistical description looks different from the rest. But then again, we look into its mean, standard deviation, min and max, and it seems like no anomalies should not be present here too. This makes sense as the stock value data seems to be very well recorded, stored, and used on the regular basis. Hence we can be sure about accuracy and quality of the given dataset.

2.2 Exploratory Visualization

In this section we visualise the data, to get a better intuition of it (Figure 3 and 4). As we can see from Figure 3, these time series data has certain characteristics that we expect, like upward sloping trend, variation around the mean, and close

Figure 2: Statistics on GOOG_daily.csv data

	1. open	2. high	3. low	4. close	5. volume
count	1496.000000	1496.000000	1496.000000	1496.000000	1.496000e+03
mean	882.382001	889.900442	874.737309	882.604157	1.724056e+06
std	262.169260	264.724737	260.205391	262.647002	8.669839e+05
min	494.650000	495.976000	487.560000	492.550000	7.900000e+03
25%	642.885000	649.062500	635.243500	643.432500	1.212993e+06
50%	829.095000	833.565000	826.490000	830.695000	1.506194e+06
75%	1106.837500	1118.120000	1099.020000	1106.480000	1.963721e+06
max	1525.070000	1532.110000	1521.400000	1526.690000	1.116494e+07

relationship between each other. Figure 4, has it's time series data vary around mean. We expect our deep neural network to capture these features and take them into account when the future prices will be calculated.

2.3 Algorithms and Techniques

As stated in the Problem Statement section, we will use LSTM neural network.

Long Short Term Memory (LSTM) cells are like mini neural networks designed to allow for memory in a larger neural network. This is achieved through the use of a recurrent node inside the LSTM cell. This node has an edge looping back on itself with a weight of one, meaning at every feedfoward iteration the cell can hold onto information from the previous step, as well as all previous steps. Since the looping connection's weight is one, old memories wont fade over time like they would in traditional RNNs (Recurring Neural Network) [7].

LTSMs and recurrent neural networks are as a result good at working with time series data, thanks to their ability to remember the past. By storing some of the old state in these recurrent nodes, RNNs and LSTMs can reason about current information as well as information the network had seen one, ten or a thousand steps ago. LSTM cell are a default layer in Tensorflow's Keras.

To predict next day opening price, we will split the data into 'X' and 'y', where 'X' will be our regressor, and 'y' will be the dependent variable (a variable to be predicted). 'X' will consist of n-amount (in our case 50) historical data points, with information on open, high, low, close, volume values (ohlcv values from now on) leading up to the date on which we try to predcit opening value. Hence, next day opening value will be the 'y' for our model. To summarise, our LSTM neural network take X as an input and will train itself to predict y.

2.4 Benchmark model

For this problem linear regression with OLS (ordinary least square) estimation will be used as a Benchmark model. Linear regressions is pretty simple and straight forward mathematical approximation of the patterns inside the data,

Figure 3: High, Low, Open, and Close values of Google stock. (1496 days)

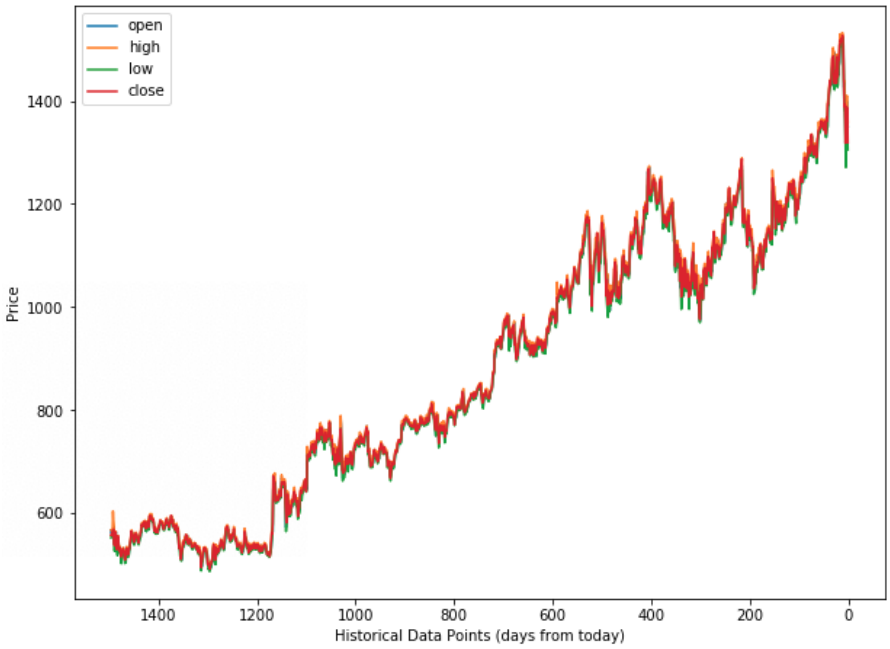


Figure 4: Volume traded values of Google stock. (1496 days)

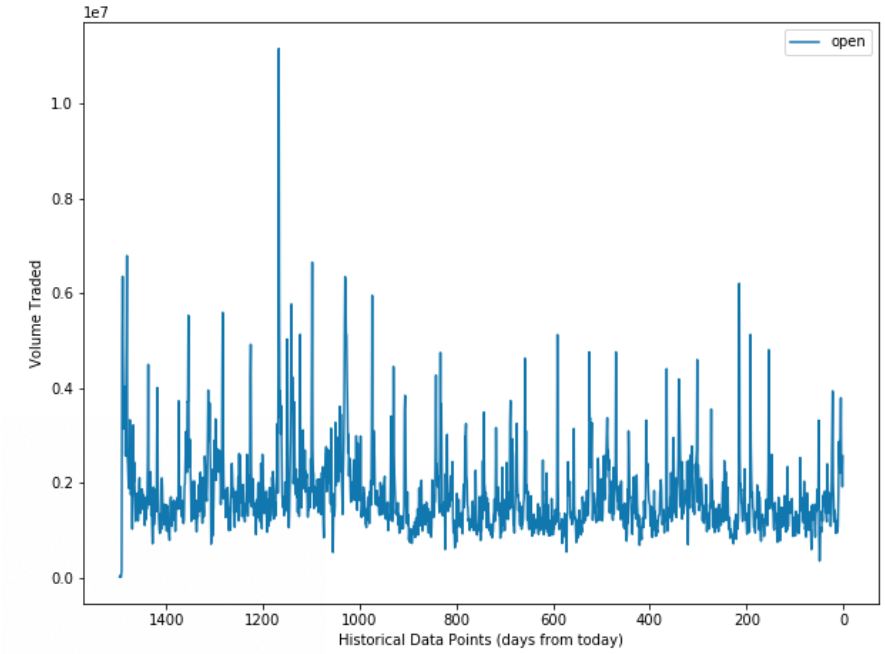
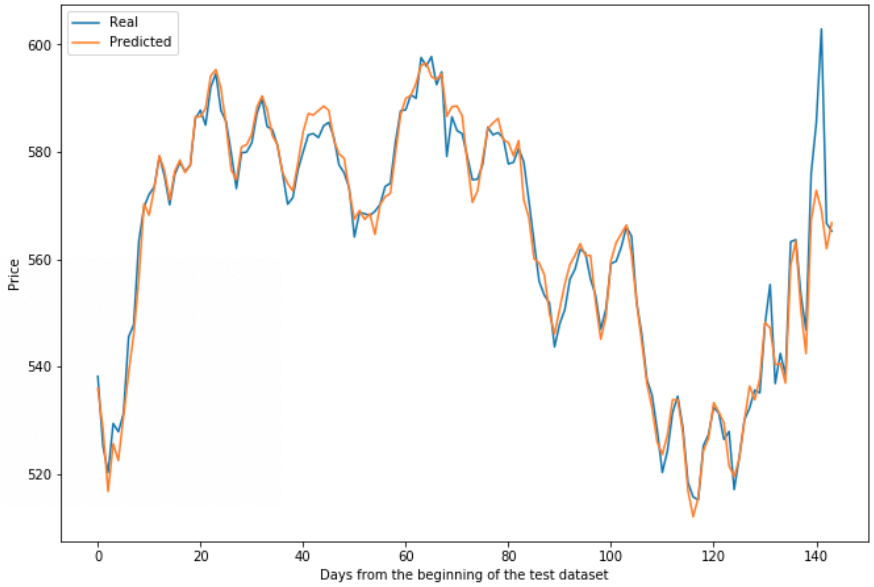


Figure 5: Linear regression results. $\hat{y}$ vs y. (Predicted open values vs given open values)



that gives reasonable approximations and predictions. Due to personal previous experience with time-series data and linear regressions, we would like to see how the LSTM Neural Network will perform against the Linear Regression Model.

To obtain the predicted results, we have used the same procedure to pre-process the data as we used for the LSTM model. This was done in order to avoid any unnecessary errors, or wrong implementations, and be able to compare two models more closely. The linear regression model had one major difference from the LSTM model, instead of having ohlcv values as the regressors, we decided to only use open value, hence our regressor was consisting of less dimensions of data then the LSTM neural network. Nevertheless, linear regression model was trained on the same dataset with the same lag, and we have obtain results.

After the model was trained, we have created $\hat{y}$ values (predicted open values). RMSE of the regression was 4.12, and R^2 of the regression was 0.966. The obtained values can be seen in Figure 5. This benchmark model seems to be predicting open values very well, which can be seen from the predicted data having very similar different pattern to original open values. Furthermore very high R^2 score indicates that our model does a very good job in capturing how the model explains the dependent variable data. This is a very strong model that will be very "tough to beat".

3 Methodology

3.1 Data Preprocessing

The first step is to prepare the pollution dataset for the LSTM. This involves framing the dataset as a supervised learning problem and normalizing the input variables. We will frame the supervised learning problem as predicting the open value at the current day (t) given the stock value measurement and volume traded at the prior time steps. We can transform the dataset using the series_to_supervised() function developed in the by Jason Brownlee [8]:

```
# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = pd.DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = pd.concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

Firstly 'GOOG_daily.csv' is loaded. Next, all features are normalized, then the dataset is transformed into a supervised learning problem. The close, high, low, and volume value variables for the day to be predicted (t) are then removed.

```
# load dataset
stock_data = pd.read_csv('GOOG_daily.csv', header=0, index_col=0)
values = stock_data.values

# ensure all data is float
values = values.astype('float32')

# normalize features
scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
```

```
scaled = scaler.fit_transform(values)

# frame as supervised learning
reframed = series_to_supervised(scaled, history_points, 1)

# drop columns we don't want to predict
reframed.drop(reframed.columns[[-4,-3,-2,-1]], axis=1, inplace=True)
print(reframed.head())
```

3.2 Implementation

First, we must split the prepared dataset into train and test sets. We choose a split of 0.9 of train data to 0.1 of test data (where 1 is full dataset). Now we can define and fit our LSTM model.

We will define the LSTM with 50 neurons in the first hidden layer and 1 neuron in the output layer for predicting open value. The input shape will be (50, 5) for historical_data window of 50, and 5 variables to be included. We will use the Mean Absolute Error (MAE) loss function and the efficient Adam version of stochastic gradient descent. The model will be fit for 50 training epochs with a batch size of 72. The output should be scaled open value for the time (t).

```
# design network
lstm_model = Sequential()
lstm_model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
lstm_model.add(Dense(1))
lstm_model.compile(loss='mae', optimizer='adam')
# fit network
history = lstm_model.fit(train_X, train_y, epochs=50, batch_size=72,
... validation_data=(test_X, test_y), verbose=2, shuffle=False)
```

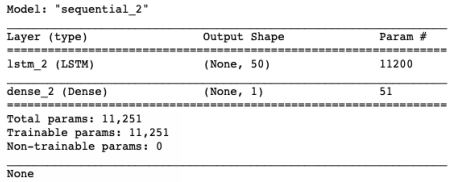
Finally, we fit the test data (test_X) into the lstm model to obtain the prediction for the open value (yhat). Then to compare the results we use inverse-transformation of the obtained 'yhat' and 'y_test' variables (because we have scaled and normalized data for the better neural network performance). We perform inverse transformation with the same scaler object that we have normalised data with, in Data Preprovessing step.

```
# make a prediction
yhat = lstm_model.predict(test_X)
print(yhat.shape)

LSTM_test_X = test_X.reshape((test_X.shape[0],
...history_points* n_features))

# invert scaling for forecast
inv_yhat = np.concatenate((yhat, LSTM_test_X[:, -4:]), axis=1)
```

Figure 6: LSTM Model Architecture



```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 50)                11200
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 51
=================================================================
Total params: 11,251
Trainable params: 11,251
Non-trainable params: 0
_________________________________________________________________
None
```

```
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]

# invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
inv_y = np.concatenate((test_y, LSTM_test_X[:, -4:]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:,0]

# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)

# calculate R^2
LSTM_r_squared = r2_score(inv_y,inv_yhat)
LSTM_r_squared
```

After data was scaled back, we can compare the obtained predicted open value results ($\hat{y}$) with the original open values (y). This model results in RMSE = 6.89, and $R^2 = 0.90$. In general these are good results, but not good enough to outperform the linear regression model.

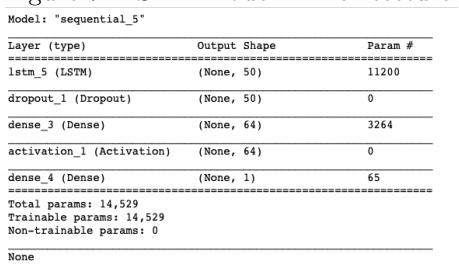
3.3 Refinement

As stated in the previous sub-section, the first intial LSTM neural network model obtained precision: RMSE = 6.89, and $R^2 = 0.90$. To beat the benchmark model, we have to try adjust some parameters, and/or reconsider the neural network architecture.

First we focus on adding extra layers. We do this by intuition and decide to add 3 extra layers: 1 dropout (with 20% dropout rate), 1 dense layer (with 64 'neurons'), and 1 Activation layer (sigmoid function) (can be found in Figure 7). After training this lstm neural network and obtaining predicitons for the test values, we compare them with the real y values and observe the resulted metrics. RMSE = 13.3 and $R^2 = 0.64$. This is clear downgrade from the previous model. So we abandon more complex architecture in favor of original simpler one.

Second we look into the hyper-parameters that are inside our original lstm model, and try to adjust those. Our first intuition is to adjust the batch_size

Figure 7: LSTM Model 2 Architecture



```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_5 (LSTM)                (None, 50)                11200
_________________________________________________________________
dropout_1 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_3 (Dense)              (None, 64)                3264
_________________________________________________________________
activation_1 (Activation)    (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 65
=================================================================
Total params: 14,529
Trainable params: 14,529
Non-trainable params: 0
_________________________________________________________________
None
```

parameter. After setting it's value to 22 (instead of 72), we train our new lstm model, and obtain the predictions. Luckily we have obtained improved RSME and R^2 values of 3.81 and 0.971, respectfully. This indicates that our third version of lstm neural network model does a better job predicting opening stock price than benchmark model! Hence, this will be our final model that we will use to predict the opening stock prices.

4 Results

4.1 Model Evaluation and Validation

Our final model for this research is LSTM model with 1 LSTM layer with 50 neurons, and 1 Dense layer. We used Mean Absolute Error loss function and Adam version of stochastic gradient descent. The model has 50 epochs to train witha batch size of 22.

This model seems to perform well based on it's RSME, R^2, and the visualized results (Figure 8). Due to the nature of LSTM layer, and extensive documentation of usage LSTM model to predict time-series data, we conclude that our model must be a correct way to solve a problem of predicting opening stock price values. Furthermore model inputs and outputs are specified correctly, hence, this machine learning algorithm must give us results as we expect them.

We have quickly experimented with omitting certain single observations from the dataset (first, last and one in the middle), and still the RSME, R^2, and the visualization of the results was not changing drastically. Hence, we assume that the model should be robust enough to be used. Unfortunately, when we tried a little more sophisticated way to check for model's robustness, we came into difficulties using the k-Fold Cross validation, and were unable to test the model for robustness the way we initially intended (we will add it to the 'improvements' section). But our quick analysis with omitting variables seemed to gave us encouraging results.

Generally speaking, we are satisfied with the results and the performance of the model, and we suggest that the model is good enough to be used.

4.2 Justification

Benchmark model for this task, the linear regression model, had a very high RMSE score of 4.12 and $R^2 = 0.966$. In fact, it seems that linear regression remains one of the most effective ways to predict the future open price stocks.

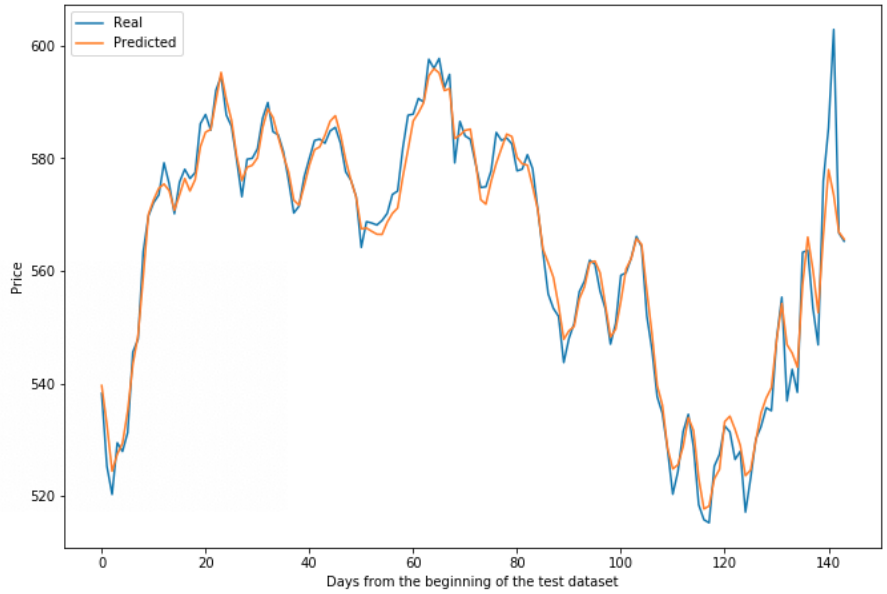
Nevertheless, after some adjustments to our LSTM neural network we have created a model that is capable to have better predictions and fit the predicted data more closely. This LSTM neural network has outperformed our benchmark model, but by a very small margin. RMSE score imporved from 4.12 to 3.81, and R^2 improved from 0.966 to 0.971. In our opinion this is not a very drastic difference, and hence there is no clear evidence if in our case LSTM should be preferred over the linear regression. Also, taking into the consideration that LSTM has higher dimensionality in this case, and is more computationally expensive (we base this assumptions solely on the time it took to train both models, linear regression was trained in seconds, and lstm neural network took about 1 to 2 minutes), we might suggest that for this specific task, the way we designed it, linear regression might be preferred.

On the other hand, lstm neural network allows us for a greater dimensionality, which means that if we would like to further improve the model, and add additional technical indicators, we might have a better scaling option with the lstm. In fact, in our case, the expensiveness of the lstm neural network does not seem to affect our lives too much. Hence, we should not be too worried about it. Furthermore, there is still an improvement made from the benchmark model, and our lstm has outperformed linear regression model, only after the 3rd adjustment! Hence, we get a feeling that further fine tuning and adjustment of the lstm model might bring even better results.

In the end, simply by comparing two graphs of $\hat{y}$ to y of linear regression, and lstm neural network (Figure 5, and Figure 8), we intuitively get a feel that lstm neural network provides us with a better fit, than the linear regression model.

Unfortunately, there is one set-back that we were unable to correct due to the time-constrained nature of this research. There was a problem with re-scaling data back to it's normal size. This error does not seem to affect our models predictive power (as $\hat{y}$ and y values both use the same scaler to inverse transform values into the open prices), but it does affect the results by producing generally biased numbers instead of open price that we should have expected. This error must be due to the incorrect use of the scaler, and how we chose to scale the data. In the near future, we will look closely into it, and the error will be fixed on my github page [9]. Hence, we advice against using these models in the current state for the real-life work and research.

Figure 8: Final Model. LSTM Model 3 results. $\hat{y}$ vs y. (Predicted open values vs given open values)



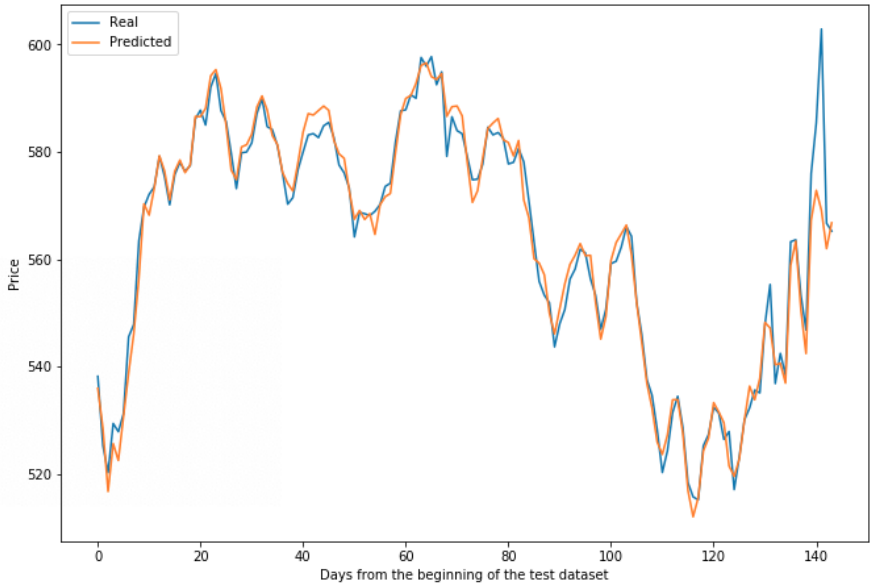
5 Conclusion

5.1 Free-Form Visualization

Once again, we will look closer into the performance of two models by the means of graphs: Figure 8 and Figure 9. As we can see it is visually observable that our LSTM model is a tighter fit to the y-test values. At some 'peaks' where the open value stops growing abruplty, the linear model keeps on giving larger values for the open price, while lstm model adjusts itself quicker, and does not 'jump over' the real open price (y-value). This and overall tigher fit of $\hat{y}$ around y value resulted in the higher R^2 value and lower RMSE. These graphs are good indicator of how well our models performed in this task.

Furthermore, from these graphs we can see the problem mentioned in the previous section. The open price of GOOG stocks varies between \$520 - \$600, while in reality, these days Google's open stock value was avereging at \$1300. As mentioned before this happened due to incorrect way of using scaler and inverse trandformation function. Once again, it should have not affected the performance of the models, as the models was trained on normalised data. In the near future this mistake will be corrected on my github page.

Figure 9: Benchmark Model. Linear regression results. $\hat{y}$ vs y. (Predicted open values vs given open values)



5.2 Reflection

For this project we have learned many different instruments, theories, models, and generally have upgraded our machine learning skills. First we have learned how to extract investment data using API, from the reliable source. Next, we have managed to clean the data, and find a way to prepare data for the supervised learning problem. This included: making sure the data is a float type, creating lags, standardizing column names, cleaning unnecessary data, and of course scaling it down to be in range of [0,1]. In the next step, we have split our data into X_train, X_test, y_train, and y_test. After data was prepared, we have created linear regression model as our bench model. It gave us necessary prediction values, and we have compared them with the real values, in order to create performance measurements. Afterwords, we have defined model architecture, and used train data to train our model. Same process was repeated with LSTM model. After obtaining necessary performance measurements, we have fine-tuned parameters, to make create the 3rd version of the LSTM model that outperformed our benchmark model. Unfortunately, we have came to difficulties trying to use kfold.split function in order to perform more sophisticated sensitivity analysis.

Unfortunately, due to the incorrect scaling this project not quite fits my own expectations about this project. But on the other hand, I personally was excited to learn how to train lstm neural networks, and even more excited about the performance of the lstm model against linear regression. Personally,

we will continue working on this project, add other variables, and try to use this knowledgge as a platform to try and create a trade-bot, for the sake of having fun with this technology.

5.3 Improvement

Further improvements for this project might be:

- Better scaling of the data (to obtain correct results).
- Use technical indicators to collect extra data (information) about the behaviour of the stock market. Use them as the input for our lstm neural network.
- Further exploration of different architectures, and further fine-tuning of the parameters.
- Find better ways for the sensitivity analysis.

We definitely would like to focus on understanding how to inverse transform data in a better way, to obtain correct results in the near future (we were unable to achieve this due to time constraints on our project).

References

[1] https://www.experfy.com/blog/the-future-of-algorithmic-trading.

[2] C.M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[3] https://www.alphavantage.co/about.

[4] https://medium.com/@george.drakos62/how-to-select-the-right-evaluation-metric-for-machine-learning-models-part-1-regrression-metrics-3606e25beae0.

[5] https://www.investopedia.com/terms/r/r-squared.asp.

[6] https://www.investopedia.com/terms/v/volumeoftrade.asp.

[7] https://colah.github.io/posts/2015-08-understanding-lstms/.

[8] https://machinelearningmastery.com/backtest-machine-learning-models-time-series-forecasting/.

[9] https://github.com/diasirish.