NEURAL NETWORKS

Market movement estimation using neural network

Written technical report

Introduction

Everyday in the global markets people and companies suffer terrible losses and gain incredible amounts of money. Enormous investments are made practically every second so we can safely say the stock markets are the „places where it all happens“. Whether to buy or sell certain stocks used to be a choice based on various reasons, often intuition instead of mathematical models. But not even the best of mathematical models and projections can predict the influence of all the variables in the real world. What we wanted to test in this project is the efficiency of a neural network in predicting stock movement.

Overview of the existing work

This problem is essentially a sequence prediction problem – which is usually best solved using LSTM (Long Short-Term Memory) neural network. What makes this type of neural networks very effective in solving this kind of problems is their ability to store past information. In this case that is very important because knowing the previous price is crucial in predicting the next price of a stock.

To understand the implemented solution, an overview of LSTMs is in order. Given the recurrent nature of this type of neural networks, it is understandable that they are used to model variation over time or even sequence dependent behaviour, e.g. predicting the next word in a sentence or the number of visitors to a certain attraction.

What makes the LSTM network stand out is the fact that they don't have the vanishing gradient problem. Small gradients and weights are multiplied many times throughout the network and throughout time steps, because of that, the gradients are gradually reduced to practically zero. This results in weight changes too small for the network to learn long-term dependencies.

These neural networks are comprised of cells. These cells have several components called the input gate, the forget gate and the output gate.

First, our new sequence value u(t) is concatenated to the previous output from the cell $h(t-1)$. At the input gate sigmoids remove any elements of the input vector that aren’t required. These nodes can be trained to output values close to zero to remove certain input values or output values close to 1 to let other values through.

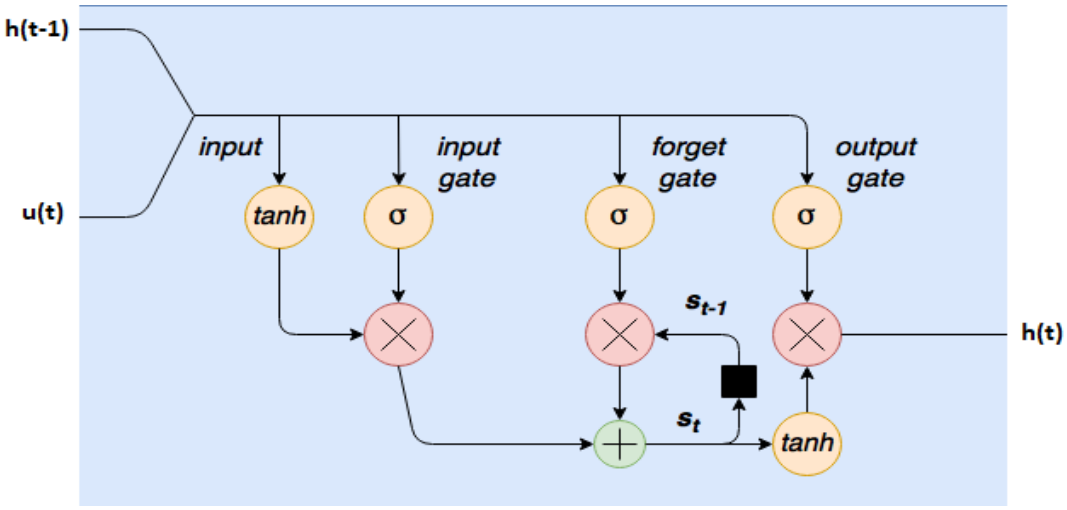

Figure 1: Graphical representation of an LSTM cell

Next comes the internal state / forget gate loop. LSTM cells have an internal state variable s(t). The last state, s(t−1), is added to the input data to create an effective layer of recurrence. This addition operation, instead of multiplication reduces the risk of vanishing gradients. This recurrence loop is controlled by a forget gate that helps the network learn which state variables to remembered or forget.

At the output there's a *tanh* squashing function. Its output is controlled by an output gate. This gate determines which values are allowed as an output from the cell.

It is also important to note that each sigmoid, tanh or hidden state layer in the cell is actually a set of nodes, whose number is equal to the hidden layer size. Therefore, each of the "nodes" in the LSTM cell is actually a cluster of normal neural network nodes, as in each layer of a densely connected neural network.

Many have seen the benefits of using neural networks to predict future sequence values, stock values especially. The fact that many papers on the subject aren't available for free says enough about the efficiency of these networks. Practically all of the available papers are student made.

Description of the implemented solution

Firstly, we acquired the necessary historical data. For this we used iexfinance. We decided to select the Close and Volume columns for use in our modelling. There is no significant difference between selecting open or close prices, and volume (amount of traded stocks) is a very important data for stock movement so it should definitely be an input.

For optimal performance it's standard procedure to scale the data. In our case, we used Scikit- Learn's MinMaxScaler and scaled our dataset to the interval between zero and one.

LSTMs require data in a specific format, usually a 3D array. First we formed the data into groups of 60 – representing two months of values, and converted it into an array using NumPy. Finally, we converted the data into a 3D array with X_train samples, 60 time steps, and two features at each step.

The LSTM layers and other layers used in the project were imported from the Keras library. Overfitting is a serious problem in more powerful neural networks, such as those using LSTM layers. To prevent overfitting a few Dropout layers were added. The argument, in our case 0.2, is the share of units that will randomly be „dropped" from the neural network during training. This prevents units from co-adapting too much.

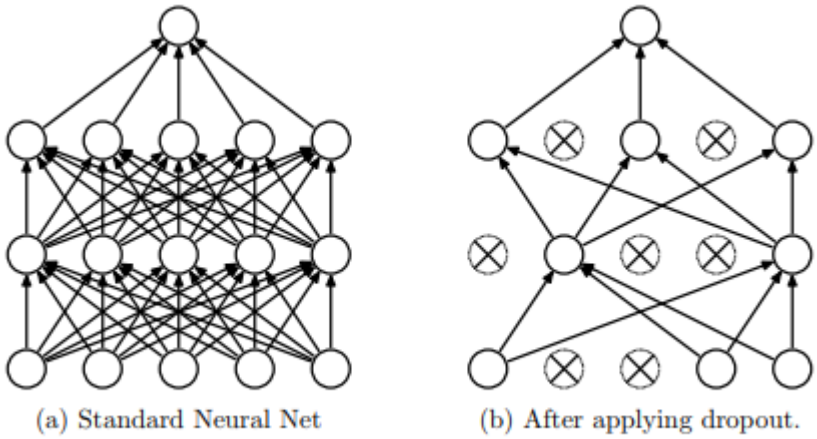

Figure 2: Neural network before(left) and after (right) applying dropout

LSTM layers used here had the following arguments: units, return sequences, input shape. Number of units is an integer argument defining the output space dimensionality. This is actually the number of nodes in the hidden layers within the LSTM cell, e.g. the number of cells

in the forget gate layer, the tanh squashing input layer and so on. We decided for units=60 because it proved to give the best results.

We set return_sequences=True so the full sequence of outputs is returned (hidden state output for each input time step). This way we're able to compare the LSTM cell output at each time step with the very next value in the sequence – for every stock price value we get a source to correct errors in the model (via back-propagation).

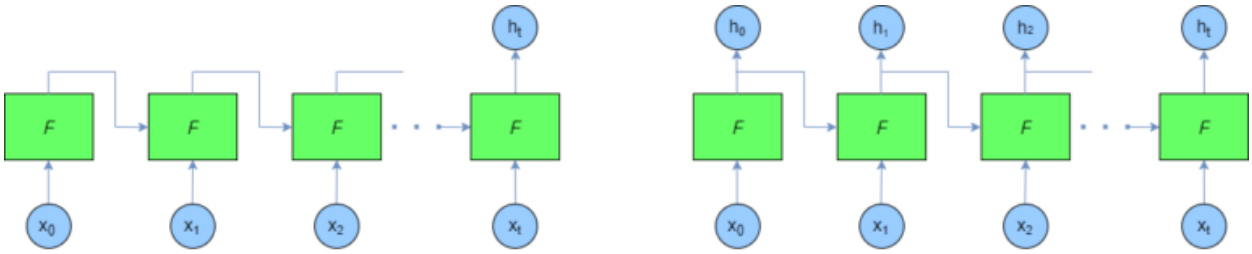

Figure 3: Outputs when using Return_sequences=False (left) i Return_sequences=True (right)

When stacking LSTM layers of the same size this has to be done because the next LSTM layer has a three-dimensional sequence input - of the same shape as the previous layer. To clarify, at each time step the model is trying to predict the very next value in the sequence. However, it does this at every time step – hence the output layer has the same number of time steps as the input layer.

The last LSTM layer is defined without return_sequences=True because its output is the input to the Dense layer – a densely connected neural network layer, so there's no need for a three-dimensional output with the outputs for all the time steps.

The third argument is the shape of the input data. As previously mentioned it's

[batch size x number of time steps x hidden size (number of features)].

Description of experimental results

Various numbers of epochs, batch sizes and other parameters were tested in training. The best results were with the following:

- time steps: 60
- units in an LSTM layer: 60
- optimizer: Adam
- epochs: 20
- batch size: 30

Figure 4: Real and predicted Apple stock prices, prediction made using our implemented solution

As mentioned once before, a serious problem is overfitting. This plot shows our results when using too many epochs, thus overfitting the network.

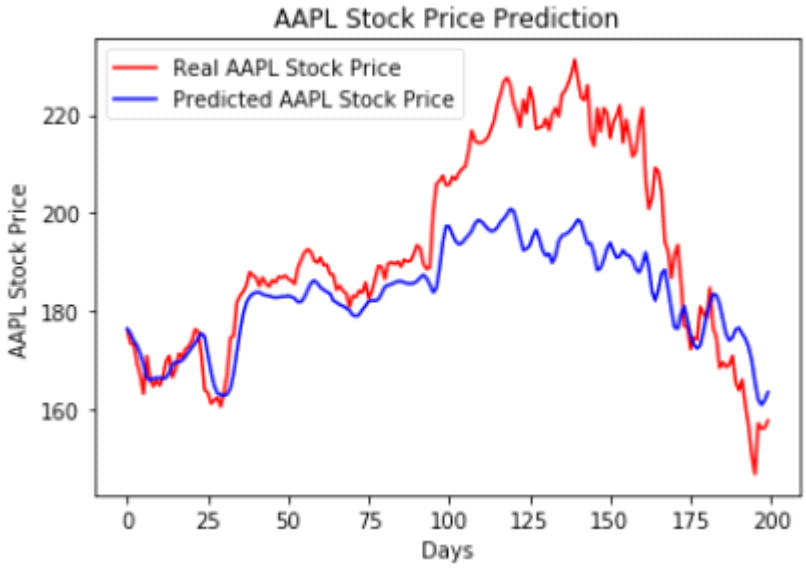

Figure 5: Results when overfitting

During training we used Adam optimizer and calculated the mean square error.

In order to predict future stock prices we had to merge the training set and the test set on the 0 axis. If we do this, our training data must be directly followed by the test data. The test data was scaled based on the same parameters as the training data. The last training sample of 60 stock price values was used as the first test sample. After testing, the outputs of the neural network, predicted values, were scaled back in order to be plotted alongside the real values.

Discussion comparing our results with previous results from the literature

Our implemented solution has shown that it is functional. The predicted values are somewhat different than the actual values. In comparison to available student projects on the same subject, our network seems to be less precise.

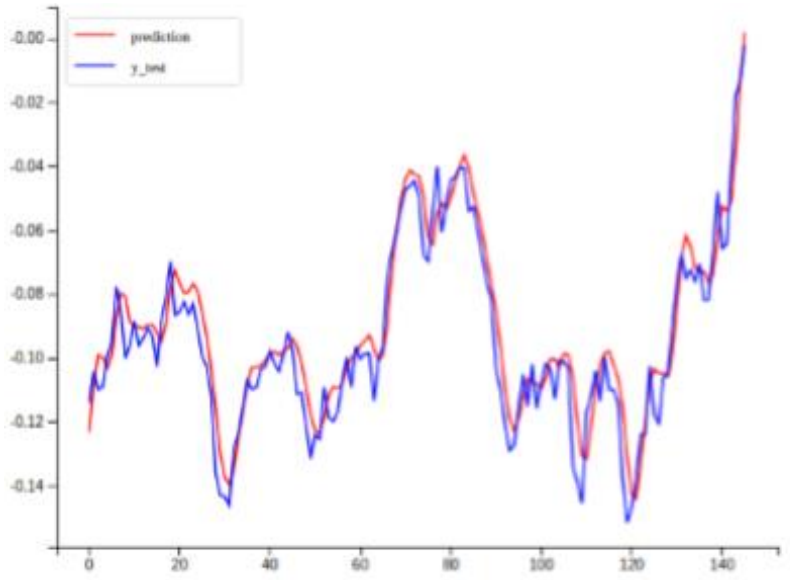

Figure 6: Real market movement and prediction in similar student project (Roondiwala M., Patel H., Varma S., Predicting Stock Prices Using LSTM)

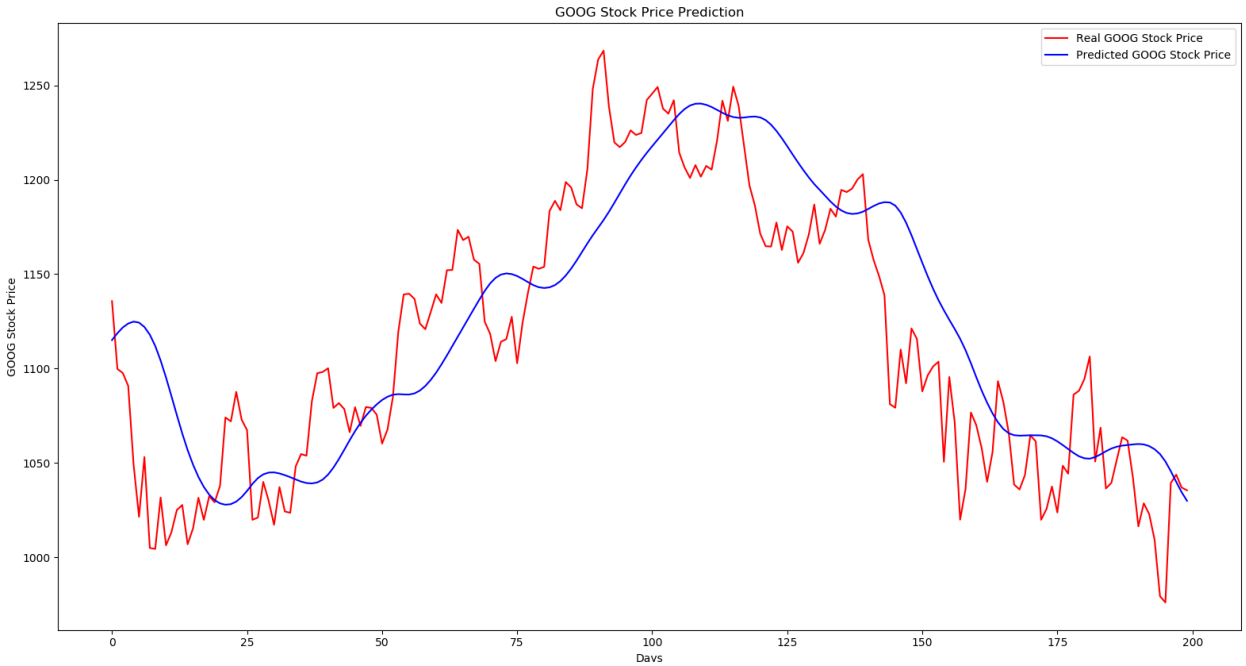

Figure 7: Real and predicted Google stock price movement, prediction made using our implemented solution

Financial Instruments

We will use two common financial instruments for making profit, they are long and short.

A long is the buying of a security such as a stock, commodity or currency with the expectation that the asset will rise in value. The profit is the difference between the value of position when you bought the security and when you sold the security.

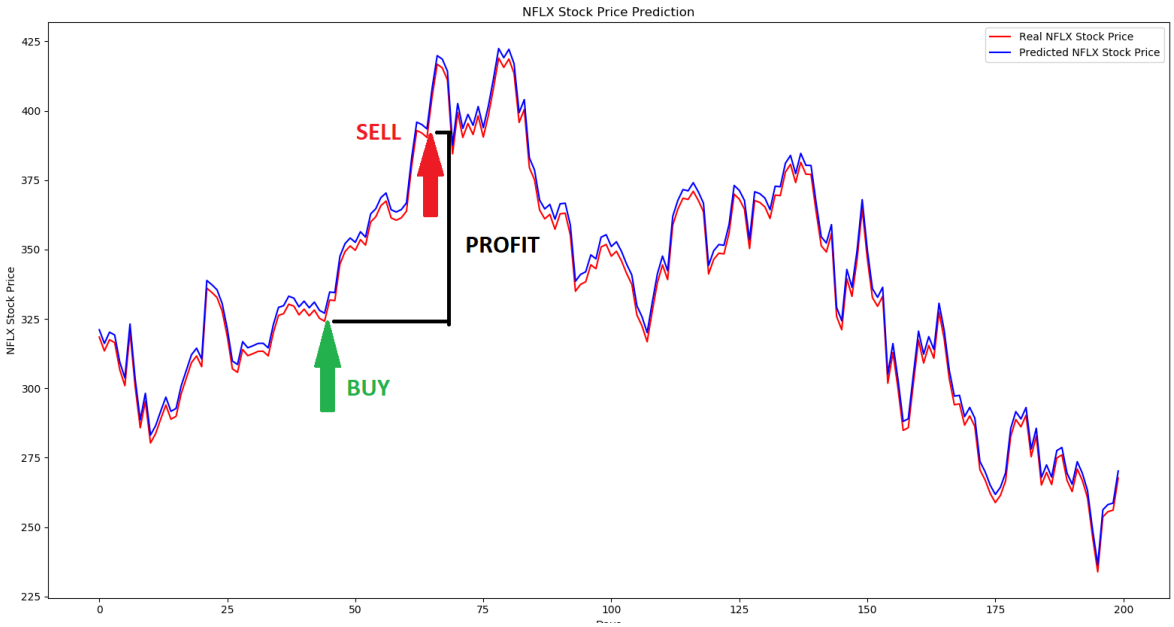

Figure 8: Example of long buying

A short, or short position, is selling first and then buying later. The trader's expectation is that the price will drop, the price they sell at is higher than the price they buy it at later. That implies that if the value of the position is lower at the closing than the value of opening, the profit will be made by that difference.

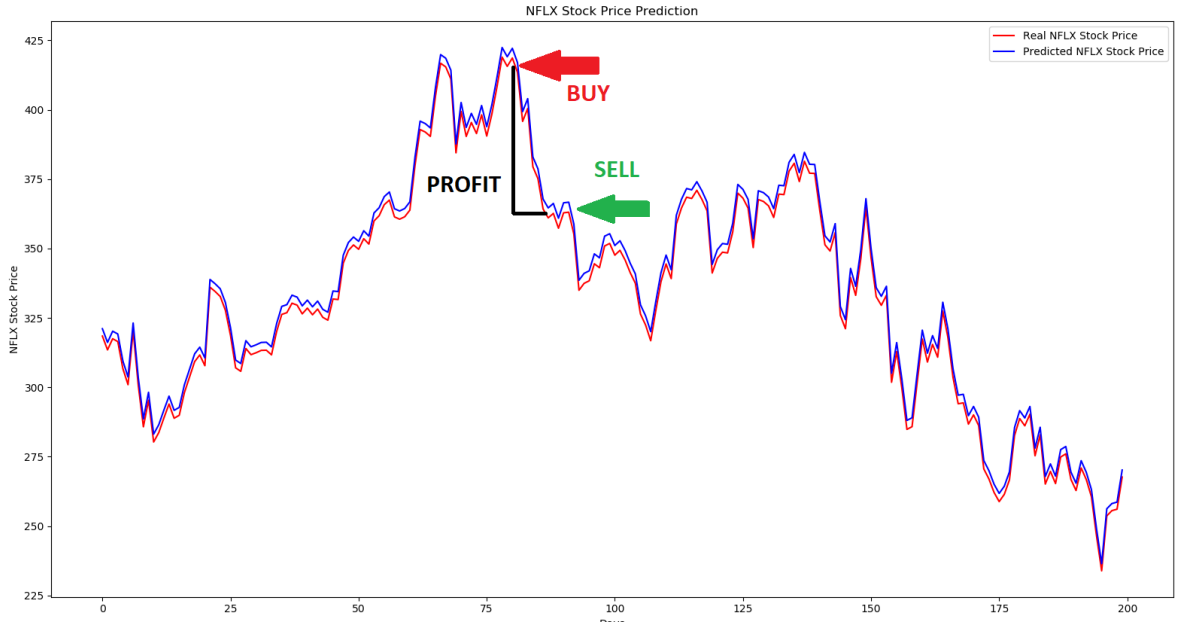

Figure 9: Example of short selling

Trading strategy

One thing that should be noted before we explain our trading strategy is the term “spread”. Spread is the difference between buy and sell on trading platforms, and it’s a commission for use of the platform. It is usually inversely proportional to the liquidity of traded security. In our case for modelling profits we assumed that spread for opening of position costs 0.35$ for one Apple stock.

Output of neural network gives us prediction of the stock price for the next day. We begin with opening of our position at the first day if the requirements are satisfied. If the neural network predicts that next day the stock will rise, and the rise will be higher than the spread we will go long on the stock. If the neural network predicts the stock will fall more than the spread, we will go short on the stock. Otherwise we will wait for a better opportunity. If we opened the position, and assuming we went short (long and short cases are symmetrical, so only one explanation will be needed) and if the neural network gives the prediction for stock price in the next step that is even lower than the current one we will stay in position for as long as that is

the case. When we come to the point when our prediction will say that the next value will be higher than the current one, then we liquidate our position and take profits. After liquidation, we repeat the process for opening of the position and go on and on.

Conclusion

In this project we made a model of an LSTM neural network that was supposed to evaluate future stock prices. LSTM has once again proven to be a good solution for solving sequence prediction problems so it is no wonder that there is so much interest in them. Overfitting has also shown to be a problem, yet the dropout method is effective in avoiding it. Parameters are best optimized by testing with various values.

List of references

Keras LSTM tutorial – How to easily build a powerful deep learning language model, February 3, 2018, https://adventuresinmachinelearning.com/keras-lstm-tutorial/?fbclid=IwAR0tjgvwRHlTuPSaQrl7FTglnxUi6mrWHVhW27xZctOWL_-AK25F5V0U6mg

Srivastava N., Hinton G., Krizhevsky A., Sutskever I., Salakhutdinov R., Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Journal of Machine Learning Research 15 (2014) 1929-1958

Roondiwala M., Patel H., Varma S., Predicting Stock Prices Using LSTM, International Journal of Science and Research (IJSR), Volume 6 Issue 4, April 2017

Jia H., Investigation Into The Effectiveness Of Long Short Term Memory Networks For Stock Price Prediction, August 2016.

Jason Brownlee, Understand the Difference Between Return Sequences and Return States for LSTMs in Keras, October 24, 2017, https://machinelearningmastery.com/return-sequences-and-return-states-for-lstms-in-keras/

Jason Brownlee, How to Diagnose Overfitting and Underfitting of LSTM Models, September 1, 2017, https://machinelearningmastery.com/diagnose-overfitting-underfitting-lstm-models/

Jason Brownlee, How to Reshape Input Data for Long Short-Term Memory Networks in Keras, August 30, 2017, https://machinelearningmastery.com/reshape-input-data-long-short-term-memory-networks-keras/

Jason Brownlee, How to Make Predictions with Long Short-Term Memory Models in Keras, August 28, 2017, https://machinelearningmastery.com/make-predictions-long-short-term-memory-models-keras/

Jason Brownlee, Get the Most out of LSTMs on Your Sequence Prediction Problem, August 11, 2017, https://machinelearningmastery.com/get-the-most-out-of-lstms/

Keras Documentation, https://keras.io/layers/recurrent/

Investopedia: Long Buying:

https://www.investopedia.com/terms/l/long.asp

Investopedia: Short selling

https://www.investopedia.com/terms/s/short.asp