Forecasting Stock Prices

Brady Metherall

Friday December 8, 2017

1 Introduction

This project seeks to forecast the stock market using a recurrent neural network[1]. The inherent stochasticity of the stock market makes predictions quite difficult. Recently neural networks have been applied to the stock market and other commodities in an attempt to learn patterns that may not be obvious [1, 3, 4, 6]. Neural networks have also been applied to other random processes such as wind speeds [4]. These networks however, are typically convolutional neural networks (CNNs). Instead of using the raw data as the input, the CNNs are fed the graphs of the data since they are much more suited for image data. This project instead uses a recurrent neural network trained with a stock simulated using the Black-Scholes model. This network is then applied to the Google and Apple stocks.

2 Generating Training Data

Historical stock data (at least that which is freely available) only provides the open, close, maximum, and minimum price (and volume traded[2]) on a daily basis. Moreover, real stock data was acquired from Yahoo Finance which only provides daily data for the last five to six

[1]Full repository of this project can be found at `https://github.com/bmetherall/Machine_Learning_Stocks`

[2]The daily volume of stocks traded was not considered since a good model could not be found, and the values vary wildly day-to-day.

years. This only provides roughly 2000 samples which is insufficient for training, and so the training data had to be created.

2.1 Black-Scholes Model

To simulate a stock for training the Black-Scholes model was used. The Black-Scholes model was developed in 1973 by Fischer Black and Myron Scholes for modelling the financial market with geometric Brownian motion; later Robert Merton extended its applicability. In 1997 they were awarded the Nobel Prize in Economics for their work.

The price of a stock is modelled by the stochastic differential equation

$$dS = \mu S dt + \sigma S dW_t, \tag{1}$$

where S is the stock price, μ is the drift / trend, σ is the standard deviation / volatility, and W_t is a Brownian motion [2, 7]. By expanding this to a first order series and making the substitution $dW_t = \mathcal{N}(0,1)\sqrt{\Delta t}$, (1) can be discretized as

$$S_{t+1} = S_t \left(1 + \mu \Delta t + \sigma \mathcal{N}(0,1)\sqrt{\Delta t}\right)$$

which is known as the Euler scheme. This algorithm can easily be implemented to simulate a stock. However, this approximates the continuous price of a stock and not the features which are provided by Yahoo Finance.

In order to transform this continuous data, suppose $\Delta t = 1$ minute, then $1440\Delta t = 1$ day. Let N be the number of simulated days, in this case 20000, and simulate for $1440N$ time steps. The array S can be reshaped to $N \times 1440$ so that each row represents a particular day, and each column represents a particular time. Since historic stock data only contains the open, close, high, and low prices per day these must be extracted. In this scheme the 570th

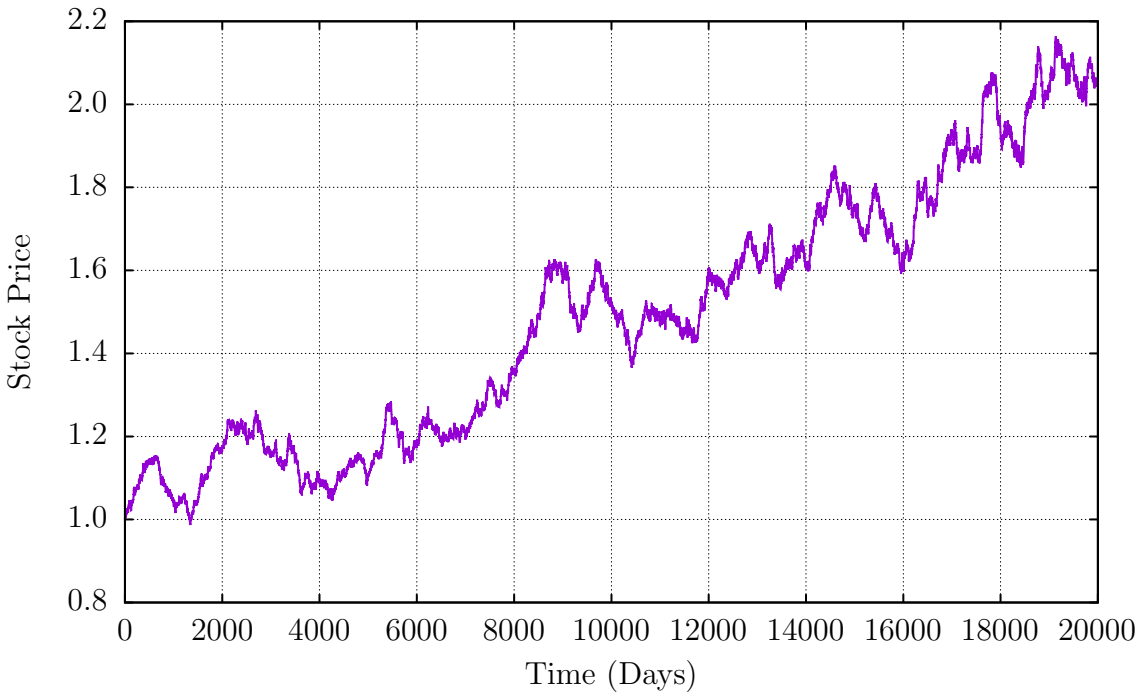


Figure 1: Daily open price of simulated stock used for training the neural network.

time step would represent the stock price at 9:30am, and the 960th time step would be the price at 4:00pm. These times coincide with the open and close times of the stock market. Thus, the 570th column of S is the open price, whereas the 960th column is the close price. To find the high and low price for the day, the maximum and minimum are taken within this range. Extracting these numbers gives the same form of data as the available historical stock data. The realization of this discretization used for training can be seen in Figure 1.

3 Neural Network

For this project Keras with the TensorFlow back-end was used. For an application such as forecasting, clearly a recurrent neural network (RNN) would be best since we wish the network to have a memory of prior events in order to better its predictions. More specifically, long short-term memory (LSTM) is used to counteract exploding / vanishing gradients.

Initially, the neural network was trained for classification. The classifications were effectively 'up', 'down', and 'neutral'; a particular day was classified as 'up' if the price in the following day was at least 4% higher, likewise a decrease of 4% would be classified as 'down', otherwise the classification was 'neutral'. This proved ineffective because of the stochasticity of the stock market the network was unable to learn any patterns. After training, regardless of the input, the network returned $(0.395, 0.388, 0.216)$ as the predicted probability distribution. These values coincide with the frequency of labels; the network simply learned the frequency of each label and always predicted this.

To alleviate this problem we change the task to one of regression. The network had the most success with a fairly simple architecture; two layers with 64 and 5 neurons respectively, and hyperbolic tangent as both activation functions, more details can be found in Table 1. The input features of the network are the open, close, high, and low values for each of the last 15 days (the window parameter in Table 1b), the output features are the predicted open price of the next 5 days (the number of neurons in the fully connected layer). Furthermore, the mean absolute percentage error (MAPE) was used for calculating the loss. The following section details why the hyperbolic tangent function and MAPE had the greatest success.

Parameter	First Layer	Second Layer
Type	LSTM	Fully Connected
Number of Neurons	64	5
Activation	Tanh	Tanh
Dropout	10%	N/A

(a) Architecture of network.

Parameter	Value
Loss	MAPE
Optimizer	Adam
α	0.8
Window	15
Batch Size	64
Validation	20%
Epochs	50

(b) Parameters used for training.

Table 1: Architecture and parameters used for training.

3.1 Stock Normalization

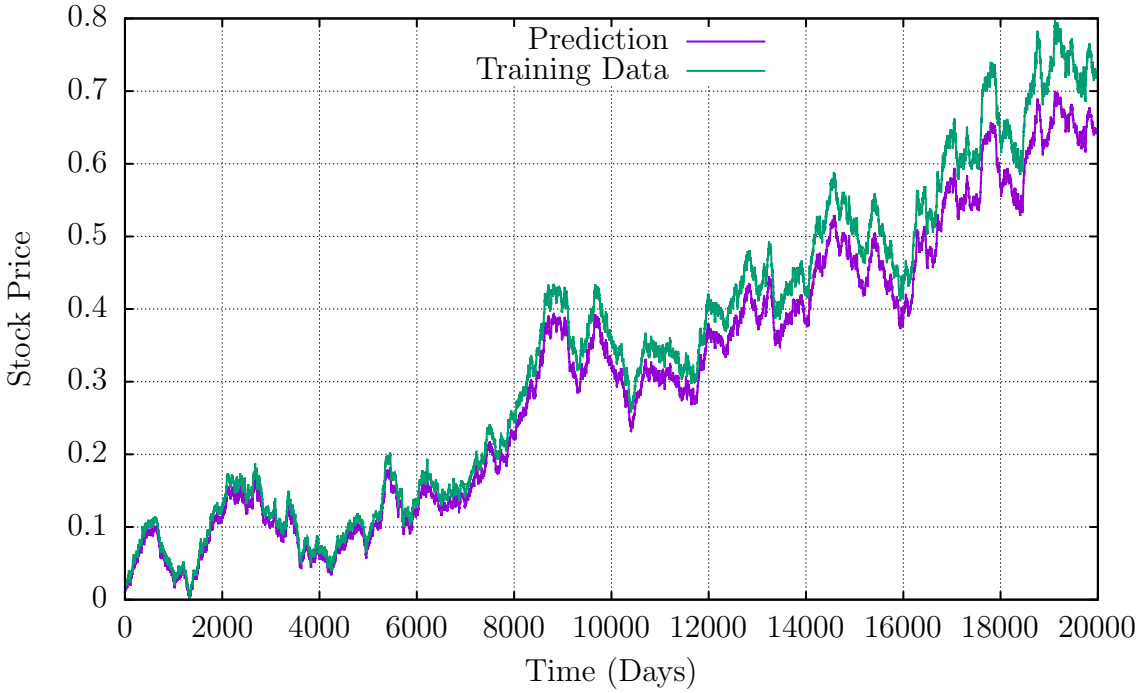
This neural network predicts the open prices of a stock in the following days, yet uses hyperbolic tangent as the activation functions. The ranges of these are wildly different, and so the stock data must first be normalized. For example, the current value of a Google share is about \$1000, whereas hyperbolic tangent's range is $(-1, 1)$. Thus, a normalization function is needed to map any stock to this range.

The most natural choice is to simply subtract the minimum and divide by the amplitude. Algebraically, this can be accomplished by

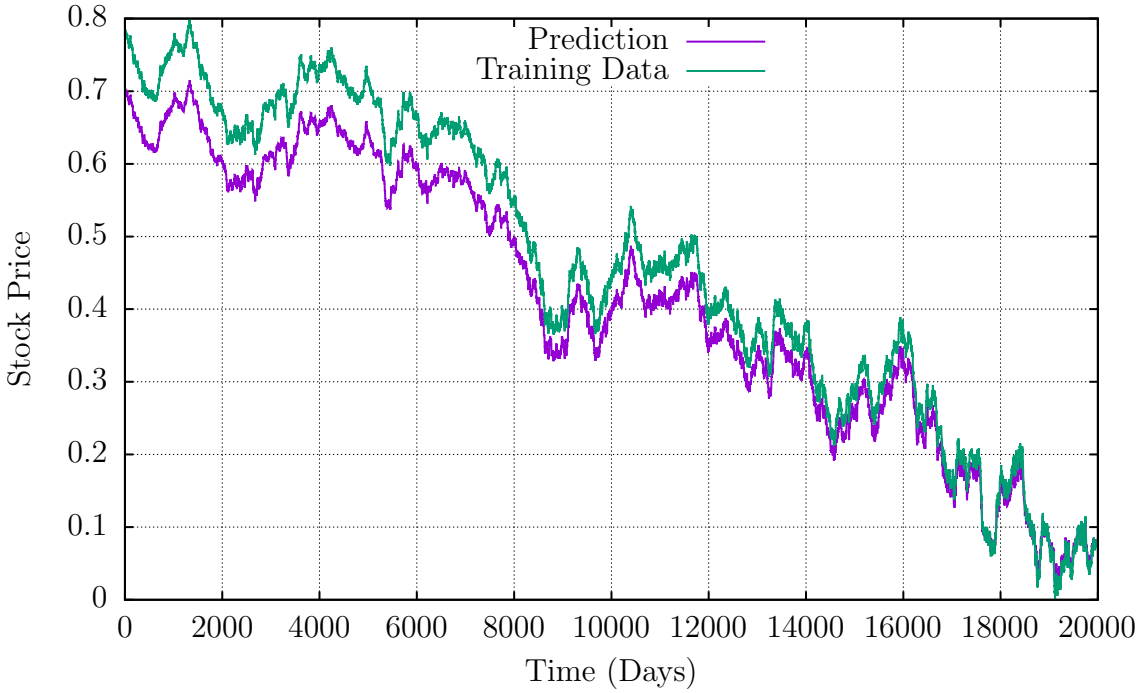
$$|S| = \alpha \frac{S - \min(S)}{\max(S) - \min(S)},$$

where α is a scaling parameter which defines the maximum value after normalization. This parameter is important because of the non-linearity of the hyperbolic tangent function; to achieve the maximum value of unity the input must be infinite which is impractical for a neural network, and so α reduces the scale of the weights. In all cases $\alpha = 0.8$ was used.

Figure 2a shows the prediction after the network is fully trained. Clearly, as time goes on, the prediction becomes less accurate. At first this was assumed to be caused simply by the errors compounding. In fact, this is a consequence of the loss function. Since the loss is calculated by the absolute percentage difference, the network is forced to have a higher accuracy for smaller numbers. This is best seen in an example: suppose the normalized price of a stock is 0.05 and the predicted price is 0.075 – this estimate is fairly close, although it equates to a 50% error. Conversely, if the normalized stock price is 0.5 and the prediction is 0.6, the absolute difference is four times as large as the previous example, but only an error of 20%. For this reason the model is most accurate for small values.



(a) Linear normalization of sample stock.



(b) Linear normalization with reflection of sample stock.

Figure 2: Two normalization schemes for training. The loss was calculated by the mean absolute percentage difference – a better accuracy is achieved for small values. The reflected scheme provides the better present day prediction.

Initially, this seems undesirable since the best prediction occurs at the beginning of the stock's history. However, this effect can be exploited if the stock is reflected about $\frac{1}{2}\alpha$. This can be done by normalizing instead with

$$|S| = \alpha \frac{\max(S) - S}{\max(S) - \min(S)}. \tag{2}$$

Conveniently, this process can be undone to obtain the model's prediction in terms of the actual price of the stock. The inversion of (2) is

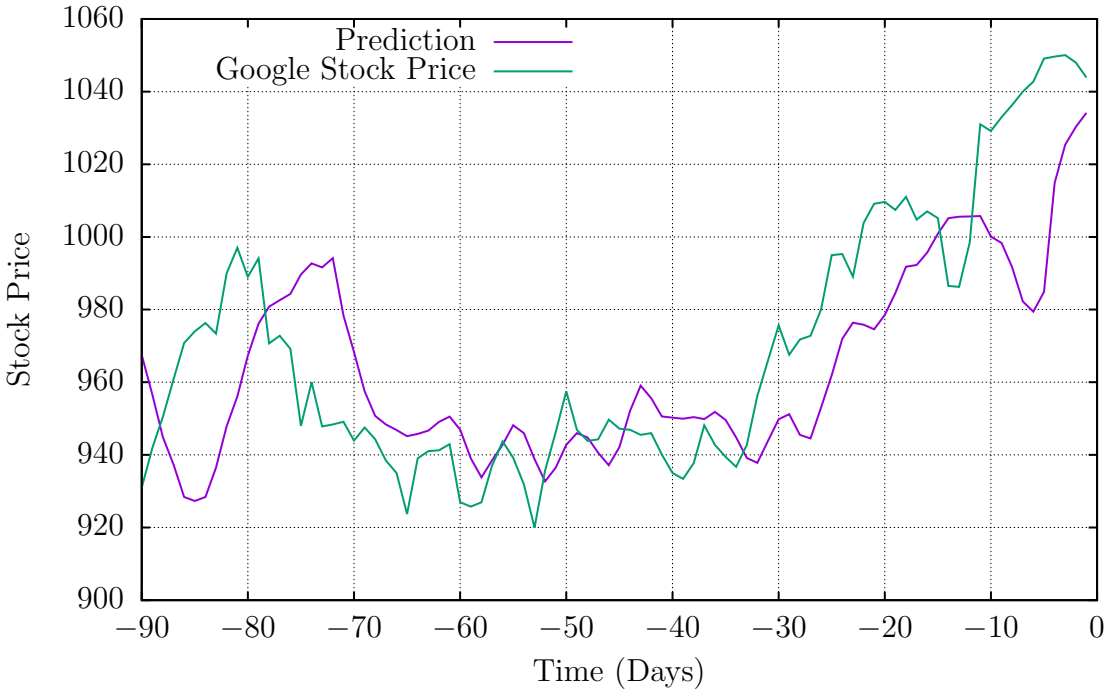
$$S = \max(S) - |S| \frac{\max(S) - \min(S)}{\alpha}.$$

Figure 2b shows the prediction after training on this new normalization. As expected, the most accurate prediction still occurs for small normalized stock prices. However, this region is now at the present time instead of the stock's inception; much more useful for forecasting.

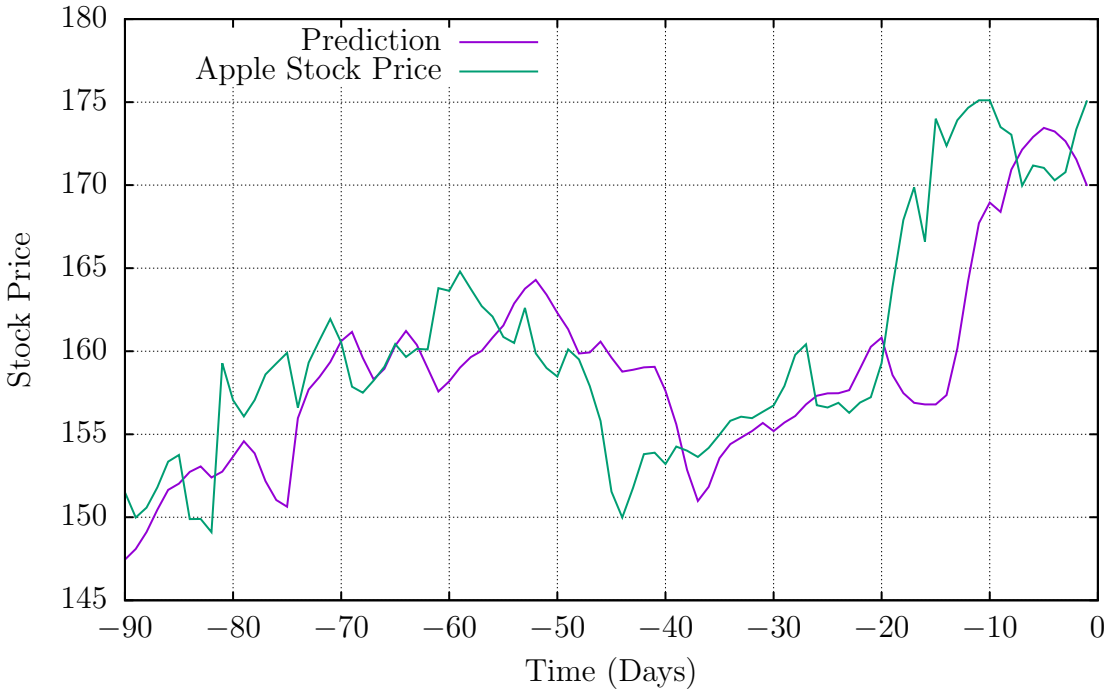
Moreover, notice that this normalized stock has a minimum value of 0, whereas the minimum of hyperbolic tangent is -1. This is desirable considering a stock could surpass its previous maximum resulting in a negative value in the normalized space. An activation such as ReLU, or sigmoid would not be able to handle such an event, whereas hyperbolic tangent is capable.

4 Application to Real Stocks

The historical stock data was pulled from Yahoo Finance using the `pandas_datareader` package in Python. This package includes the function `get_data_yahoo` which takes the stock's ticker symbol as its argument and pulls a dataframe of the stock's history. From this dataframe, the useful data can be extracted and normalized, this can then be fed into the model. The model was tested / verified using two stocks, Google and Apple. The model's predictions for the last 90 days are shown in Figure 3.



(a) Prediction of the model on the Google stock.



(b) Prediction of the model on the Apple stock.

Figure 3: Predicted value of two stocks over the last 90 days.

The model's prediction seems to estimate the price of the two stocks reasonably well – recall the model was trained using only the simulated Black-Scholes stock. Nonetheless, there is a glaring pitfall of this model, the prediction lags behind the actual stock price by about six days. Unfortunately, although the model was trained to estimate the open price of the next five days (a day per neuron in the second layer), each of those predictions are practically the same. The difference is the day five predictions are a little smoother than the day one predictions without providing any additional insight. This lagging behind is most evident when the price steadily increases or decreases such as near the beginning and the end of the 90 day period for Google.

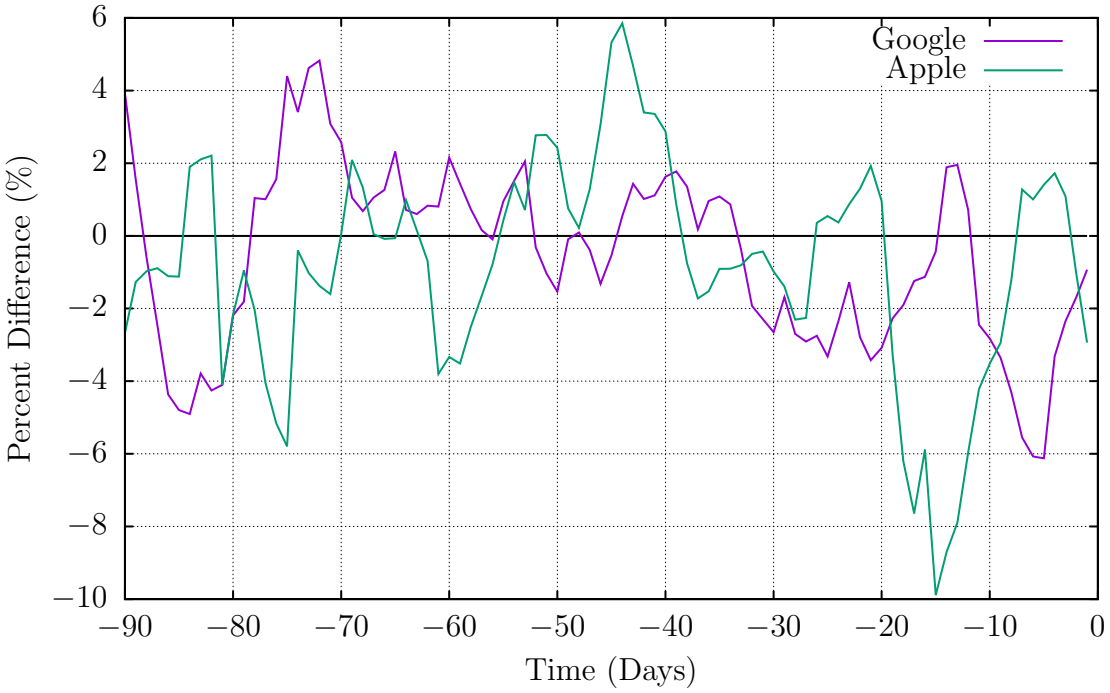


Figure 4: Percentage error of the prediction for Google and Apple stocks.

Since the model was trained using the mean absolute percentage difference, it is useful to investigate the accuracy of the predictions with a more qualitative measure. Figure 4 shows the percentage error for both stocks. The mean absolute percentage error (MAPE) over

this 90 day period is 2.06% and 2.28% for the Google and Apple stocks respectively. While this is a very good accuracy for a neural network, sadly it is not good enough. The day-to-day fluctuations of the stocks are only 0.68% and 0.92% for Google and Apple respectively. Hence, simply assuming tomorrow's open price will be the same as today's will give better results.

5 Conclusion

In this project a recurrent neural network with long short-term memory was used to attempt to forecast the open prices of the Google and Apple stocks. The network had good success at predicting the open price in the following days, however, the main drawback was that the predictions trailed the actual stock price by about six days. This, paired with the fact that the average error was around 2%, ultimately means this model is not likely suitable for forecasting the stock market in the short term.

References

[1] W. Bao, J. Yue, Y. Rao, *A deep learning framework for financial time series using stacked autoencoders and long short-term memory*, PLoS ONE 12(7), 2017.

[2] S. Dineen, *Probability Theory in Finance: A Mathematical Guide to the Black-Scholes Formula*, Graduate Studies in Mathematics (AMS) 70, 2005.

[3] S. Giebel, M. Rainer, *Forecasting Financial Asset Processes: Stochastic Dynamics via Learning Neural Networks*, BULLETIN de la Société des Sciences Médicales du Grand-Duché de Luxembourg 1, 2010.

[4] S. Giebel, M. Rainer, *Stochastic processes adapted by neural networks with application to climate, energy, and finance*, Applied Mathematics and Computation, Volume 218, Issue 3, 2011.

[5] N. Gold, Q. Wang, M. Cao, H. Huang, *Liquidity and volatility commonality in the Canadian stock market*, Mathematics-in-Industry Case Studies, 2017.

[6] A. Siripurapu, *Convolutional Networks for Stock Trading*, Stanford University Department of Computer Science, 2015.

[7] R. J. Williams, *Introduction to the Mathematics of Finance*, Graduate Studies in Mathematics (AMS) 72, 2006