MA4J5: Structures of Complex Systems

Deep Neural Networks for Stock Price Prediction

Name: Jack Atfield

Module Leader
Markus Kirkilionis

Institute of Mathematics

January 7, 2024

Contents

1 Introduction

On a surface level stock markets are incredibly complex objects. There are thousands of assets to trade each displaying seemingly completely random movements in price with no way to predict the next. To accurately predict the pricing movement of any stock is seemingly an impossible feat, at least with today's theory and technology. However, with Mathematics having advanced at incredible speeds and increasing amounts of research being put towards such a task the end goal is ever more in reach. In this project we shall investigate one such technology, namely deep neural networks (DNNs) and the effectiveness of them in predicting future prices of stocks.

In order to do this we shall make a comparison of a DNN based model to a more traditional and well established approach to predicting the prices of stock. To this extent we shall consider four of the more popularly traded companies namely Apple (AAPL), Google (GOOG), Microsoft (MSFT) and Tesla (TSLA) as these give quite a good spectrum of differently behaving stocks. With these we shall attempt to predict 14 days worth of pricing data.

2 A Brief Introduction to the Stock Market

Similarly to many other traditional markets, and very broadly speaking, the stock market is driven by supply and demand. The more a stock is purchased the higher its price is driven, and vice versa. Of course if it was as simple as this the stock market would not be such a complex, hard to predict object; however, this is a good starting point. The natural question then is what drives supply and demand? There isn't an exact answer to this, however, factors such as politics, economic events, uncertainty in a company etc. all play a role. With this in mind it becomes clear why predicting prices accurately is such a hard task – there are a lot of factors to consider.

Within a market agents buy and sell stocks in order to profit. To do so many different strategies and techniques are applied, ranging from buy and hold strategies to high frequency trading. Each of these have their own advantages and disadvantages and the choice of which to use is solely dependant of the goal of an investor, but each have one thing in common. The price of the stock is central to them all.

There is no need for further discussion here, however if one seeks a more in depth introduction I recommend resources such as [2].

3 A More Traditional Approach

When it comes to the prediction of stock prices there are many different methods suggested in literature and across the web. Not much is there to say that there is a single best method to use and in practice often a combination of methods is used. I cannot compare the performance of DNNs to each of these as that would simply take too long. Instead we shall look into one particular method that I feel is widely enough discussed and used to make a comparison.

Here we shall focus on the use of Geometric Brownian Motion (GBM) in order to make predictions as this is one of the most widely noted tools in literature and used in practice. However, before we can make use of this, it is important for us to understand what GBM is and how we can apply it to the question of predicting markets.

Recalling the definition of a stochastic process from lectures we first define a general Brownian Motion.

Definition 3.1. A stochastic process X_t is a *general Brownian Motion* if

1. $X(0) = 0$
2. The process has independent increments.
3. $X(t) - X(s) \sim \mathcal{N}(\mu(t-s), \sigma^2(t-s))$ for all $0 \leq s < t$ with s, $t \in T$.

We write $X(t) \sim BM(\mu, \sigma^2)$

Note that setting $\mu = 0$ and $\sigma = 1$ in the above definition we recover a 'standard' Brownian Motion, shown in figure 1, however the data we will be studying and simulating will almost certainly not have such values for μ and σ so such process is of little interest here as a model in itself. Note it does have use elsewhere though which is discussed later on.

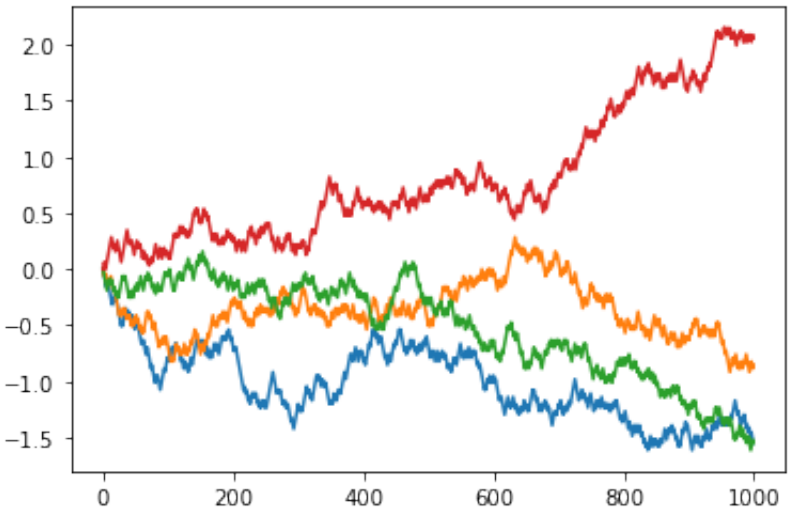


Figure 1: Simulated trajectories of a standard Brownian Motion.

We now look to how such a structure can be used for our purposes i.e. modelling stock prices. If $X(t) \sim BM(\mu, \sigma^2)$ then from [7] $X(t)$ has

$$dX(t) = \mu t + \sigma dB(t).$$

Here $B(t)$ is the standard Brownian Motion as noted above. Now with $X(t) = \log(S(t))$ simple calculation yields

$$dS(t) = \mu S(t)dt + \sigma S(t)dB(t). \tag{1}$$

This is the form we shall be working with. Note the notion of 'geometric' here comes from the introduction of the logarithm.

We now follow [5] in order to obtain a solution $S(t)$ to the above stochastic differential equation to allow us to model stock prices. We first use the original transformation $X(t) = \log(S(t))$ to make a substitution for S and use Itô's formula to obtain the following:

$$\begin{aligned} dX(t) &= \frac{1}{S} dS(t) + \frac{1}{2}\left(-\frac{1}{S^2}\right)(dS(t))^2 \\ &= \frac{1}{S(t)}(\mu S(t)dt + \sigma S(t)dB(t)) + \frac{1}{2}\left(-\frac{1}{S(t)^2}\right)\sigma^2 S(t)^2 dt \\ &= (\mu dt + \sigma dB(t)) - \frac{1}{2}\sigma^2 dt \end{aligned}$$

With this we hence have $dX(t) = (\mu - \frac{1}{2}\sigma^2)dt + \sigma dB(t)$ with $X(0) = \log(S(0))$. Integrating both sides of such equation and substituting $S(t)$ back thus yields a closed form for the price at time t.

$$S(t) = S(0)\exp\left(\left(\mu - \frac{1}{2}\sigma^2\right)t + \sigma B(t)\right) \tag{2}$$

We shall discuss further exactly what $B(t)$ is here shortly.

3.1 Using GBM to Model Prices

The question that remains is how shall we use this model to finally predict stock prices. Before we may do so we first must make some reasonable assumptions on the data that we will be modelling. For simplicity we make the assumption that returns $r_t = \log\left(\frac{S(t)}{S(t-1)}\right)$ follow a $\mathcal{N}(0, \sigma^2)$ distribution, where σ^2 is the variance of the historical returns (over a period of time of fixed size). Thus in equation 2 $B(t)$ is taken to be a standard Brownian Motion i.e. its increments will be distributed according to $\mathcal{N}(0, 1)$. We may do this as $\sigma\mathcal{N}(0, 1) = \mathcal{N}(0, \sigma^2)$ hence we are ensuring we model the returns correctly under our assumption. I would like to note, however, that it can be shown that such assumption is not perhaps as reasonable as one may think. In fact using various statistical tests [5] does exactly this, yet still works under the same assumption as we are here so we shall continue under it. Secondly we assume that the mean and variance of returns $\left(\mu \text{ and } \sigma^2\right)$ remain fixed over the time period we are predicting prices for. In reality this is not the case and these values do change with time, however since we are predicting prices for short periods of time I feel this be reasonable.

Finally, before we can run our model we need the data which we will use to generate our parameters for it. In the case of our version of GBM this is a very simple affair. For each stock we wish to model the price of we shall use data from Yahoo! Finance taken over the period 2020-10-01 to 2020-11-30 to calculate such parameters and then predict the prices over the period 2020-12-01 to 2020-12-18 (a 14 day trading period). Such two month 'training' period is used to mimic the way our Neural Network based approach will make its predictions (as well as allow for easier comparison), this will be discussed more extensively in section 5.

Now with such assumptions and data, using GBM to model prices is simple. We calculate the mean, μ, and standard deviation, σ, of the historical daily returns r_t. We then simply plug the calculated parameters into our equation for $S(t)$ to generate predicted prices for each $t \in \mathbb{N}, 0 < t \leq T$ with $S(0)$ being the final value of the historical price data used to calculate the historical returns. Predicted prices can then be plotted and compared to true prices to evaluate the effectiveness and accuracy of such a model as described here. We perform such analysis in 6 by way of the Mean Squared Error (discussed in section 5.3).

4 Deep and Recurrent Neural Networks

Neural networks, in some simple form, have been present in applied Mathematics for much longer than many people are aware. Think to the method of least squares regression for example. We may use a single layer of output nodes in which outputs are fed back as inputs via a series of weights and the sum of products of the inputs and weights is calculated at each node. Given some target, one can then minimise the Mean Square Error (MSE) via adjustment to the weights. Neural networks are far beyond this simple task now however. From classification to data generation to prediction, NNs are an extremely useful tool to have to hand. We will make use of such structure for prediction purposes, but in more complex forms than the simple one layer example given above.

A neural network can take many forms, typically with complexity proportional to that of the task at hand. As we wish to use NNs to forecast stock prices – a complex task – we shall be making use of 'deep' and 'recurrent' neural networks. Simply speaking a Deep Neural Network (DNN) is a neural network that utilises more than one layer of nodes. A canonical, albeit irrelevant, example/application of such a network was presented very early on in the development of such a technology and concerned the complex task of identifying handwritten digits (of which a brilliant discussion can be found in [4]). A Recurrent Neural Network is a bidirectional NN so to speak. That is to say the output from nodes can affect subsequent input to the same nodes i.e. such networks can process sequential data. Figure 2 displays this excellently. Such property means that when processing an element of a sequence, the network will take into account information drawn from previous elements of the sequence [6].

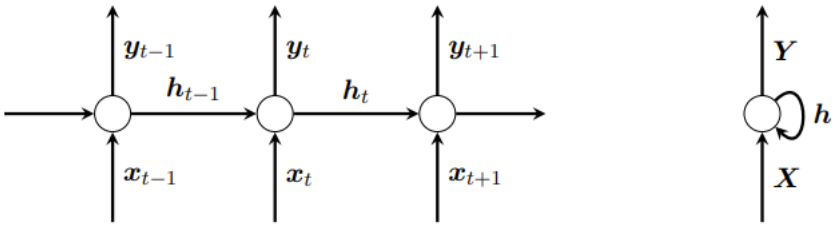


Figure 2: A single node from a RNN.

4.1 Deep Neural Networks

The theory associated with DNNs is not dissimilar to that of single layer networks, the only difference being that to obtain output is a recursive process. If we consider the first layer to be an input layer and the last an output layer, with $l-2$ layers in between then the output is generated via the following recursive formula:

$$\begin{aligned} F^{(1)}(x) &= \sigma(W^{[1]}x + b^{[1]}) \\ F^{(i+1)}(x) &= \sigma(W^{[i+1]}F^{(i)}(x) + b^{[i+1]}), \quad 1 \leq i < l. \end{aligned}$$

Here x is the input, σ is the activation function layer i and $W^{[i]} \in \mathbb{R}^{d_i \times d_{i-1}}$ and $b^{[i]} \in \mathbb{R}^{d_i}$ are the weights and biases associated with layer i. Training such networks is done via forward passes combined with backpropagation, full discussion of which can be found in [6].

4.2 Recurrent Neural Networks

Recurrent neural networks are slightly more intricate due the fact they have the memory property and work with sequential data. We shall give a brief overview here following [6]. Let $\{x_t\}$ be an input sequence and $\{y_t\}$ be an output sequence. Recurrent neural networks work by encoding previous information from the input sequence in a state variable h_t that depends only on the output from the previous element of the sequence y_t and the previous state variable h_{t-1} i.e. $h_t = f(y_t, h_{t-1})$ where f is some function. The next element of the output sequence is then generated by applying an activation function to some linear combination of the input, state variables and biases. In this way we can see that the state variable h_t not only encodes information from just the previous input element, but the entirety of the sequence. Concisely we may write this process in the following way.

$$\begin{aligned} z_t &= Wx_t + Vh_{t-1} + b \\ y_t &= \sigma(z_t) \\ h_t &= f(y_t, h_{t-1}) \end{aligned}$$

Looking back to figure 2 we can see this aligns with the algorithm we have written above, accounting for the propagation of information through the sequence to generate the output.

We must note here that RNNs in this form are not without issue. Training such a network require a form of backpropagation called backpropagation through time (BPTT). Such method is very similar to standard backpropagation, the only change being such method 'unfolds' the RNN and then treats each time t as a layer of a DNN. The first problem comes with time to train as we now have to account for T (assuming sequences are length T) data points per input when training, leading to an increased time complexity. Secondly, and more importantly there is the issue that arises of 'vanishing gradients' when training. When applying the BPTT algorithm the gradients calculated can become very small exponentially quickly which in turn leads to long term dependencies within sequences being 'forgotten' which can lead to inaccurate predictions – a big issue especially for our use case. We shall address such issue by consider a special type of RNNs call Long Short Term Memory Networks and discussion of such architectures can be found in the next section.

5 Our Approach

Before continuing it is important to understand why such network structures discussed in section 4 are necessary for our purposes. Note that that type of data that we will be working with is rather dissimilar to that of an image or any that may be typical for a NN that performs classification (as is a very common use for NNs now). Stock price data comes in the form of a time series, a (typically non-stationary) stochastic process that is sequential in nature. To this extent the type of neural network that we use must be able to account for sequential data in which relationships may be present throughout, hence we take advantage of RNNs. To ensure that all relationships within our data are accounted for when training and to mitigate the vanishing gradient problem we shall specifically make use of a LSTM network. A brief discussion of such architecture can be found below. Since relationships between data points are typically very complex in such data, it is very unlikely that a single layer network will be able to extract and replicate present relationships/patterns – this isn't a regression task after all! As such multiple layers are necessary and hence a deep network architecture will be used.

5.1 The Long Short Term Memory Architecture

As mentioned in section 4.2 RNNs have quite a big shortcoming in the fact that they have potential to forget previously observed patterns in long sequences of data – exactly the kind we wish to feed our network. In order to overcome this we make use of Long Short Term Memory (LSTM) networks which allow our RNN to exhibit short term memory over large sequences. This is done by adjusting the structure of a node to allow it to learn what information may be needed later on, maintain that and discard what is not needed. The structure of such node can be seen in figure 3 and is quite complicated, so we shall briefly explain what each part represents here. Equations 3 through 8 specify an LSTM

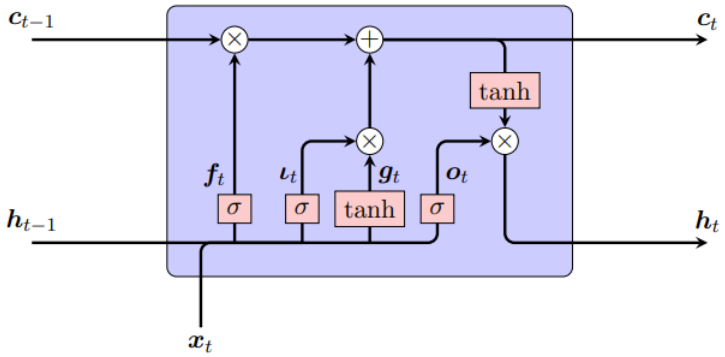


Figure 3: A LSTM Node.

node. The 'relevant' memory of the node at time t is c_t and is constructed from f_t, c_{t-1}, l_t and g_t. Here f_t decides what information to keep from previous memory c_{t-1}, l_t decides what information to update and g_t provides such information. Essentially c_t forgets irrelevant information and only keeps what is deemed important. The remaining components o_t and h_t then dictate output, where o_t decides which parts of cell state are to be used as output (alongside the memory) and h_t is the output itself. More simply put an LSTM node sequentially updates its memory and output based on the previous memory, output and new data as input.

$$f_t = \sigma(W_f x_t + V_f h_{t-1} + b_f) \tag{3}$$
$$l_t = \sigma(W_l x_t + V_l h_{t-1} + b_l) \tag{4}$$
$$g_t = \tanh(W_g x_t + V_g h_{t-1} + b_g) \tag{5}$$
$$o_t = \sigma(W_o x_t + V_o h_{t-1} + b_o) \tag{6}$$
$$c_t = f_t \odot c_{t-1} + l_t \odot g_t \tag{7}$$
$$h_t = o_t \odot \tanh(c_t) \tag{8}$$

From a brief overview of how LSTM nodes work we can immediately see why they are extremely useful to us. Stock prices follow trends over time but also are also extremely noisy. In order to make meaningful predictions we wish for our network to remember the trends over time while not over accounting for the noise present – an LSTM network allows us to do exactly this.

5.2 Network Architecture

With a basic understanding of the tools we will be using we may now look to the exact architecture being implemented. There is no exact science for designing a NN for any given task and often to find the best architecture comes down to trial and error. Many of the authors referenced in this paper have suggested architectures they found best for their use cases and data and although similar we are not exactly replicating the situations being modelled in these papers. As such we have designed our own architecture for our use case and data.

Similarly to [3] we will be using Open, Close, High, Low, Volume, Average (of the previous four prices), and Returns (given by the same r_t as in 3.1) as input for our NN, we also use fully connected layers. That, however, is where similarity ends since we won't be predicting just a single price (or sequence of prices produced recurrently) here. Here we use a 7-14-21-14 architecture, shown in figure 4. That is our input layer consists of 7 nodes (which are the quantities described above), we have two hidden LSTM layers each with 14 and 21 nodes respectively and then our output layer consists of 14 nodes (14 days of prices). Such chosen architecture is the one which was found in testing to provide the lowest MSE in predictions across the board for each of the stocks chosen for our analysis. Many different architectures were considered, however we shan't discuss them here as optimising network design is not the purpose of this project.

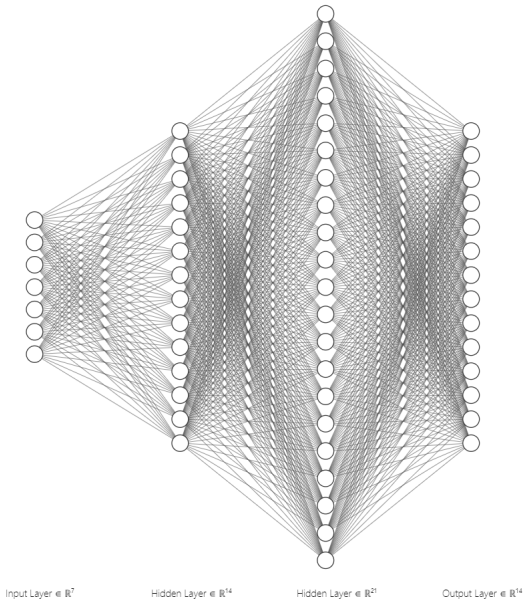


Figure 4: Our Network Arcitecture.

5.3 Training and Testing

In light of our architecture, we shall train the model using previous pricing data for the stock price of a given company. To do so we will be using 5 years worth of data (on a given company) taken from Yahoo! Finance, which contains Open, High, Low, Close, Adjusted Close prices and Volume traded each day. Before feeding such data into our network it must first be pre-processed to match the input-output format of our network. Hence, we will train the network on the set input-output pairs $\{x_t, y_t\}$ where x_t is

a sequence consisting of the previous 42 trading days (two months) of data, including average price and day's returns and excluding adjusted closing price, and y_t consists of the subsequent 14 days of closing price data. In this way, once trained, we are able to provide our NN with 42 days worth of Yahoo! Finance data for a given stock and it will provide us with a prediction of closing prices for the next 14 days. The specific period we will be predicting will be 2020-12-01 to 2020-12-18 (i.e. using 2020-10-01 to 2020-11-30 prices as input).

With the network trained, testing our predictions then becomes a simple task. We will use the Mean Squared Error which is given by

$$\text{MSE} = \frac{1}{T}\sum_{i=1}^{T}(y_i - \hat{y}_i)^2, \tag{9}$$

where $\{y_i\}_{t=1}^{T}$ is the sequence of previously observed prices over the period (our test data, the next 14 days after train data ends) we are predicting and $\{\hat{y}_i\}_{t=1}^{T}$ is our sequence of predicted prices. Using such metric we can make inference on the fitness and usability of our model. Such is discussed below in section 6.

6 Results

In order to assess our models we focus on four stocks, namely AAPL, GOOG, MSFT and TSLA. We present the results of the models discussed above, presenting plots depicting predicted versus actual prices with discussion of what these show. Alongside such plots we shall assess the performance of each model. Finally, we make a direct comparison of the RNN model with the GBM model in order to assess the effectiveness of DNN based models when it comes to predicting market movements.

6.1 GBM Model

Executing the GBM model was a simple task and only requires a slight extension of what is discussed in section 3.1. We first calculated μ and σ from our historical returns data and then used such values to generate 10,000 possible pricing paths for each of our chosen stocks. From these paths we find the maximum path by choosing the maximum simulated price for each t and the minimum path in a similar manner (choosing the minimum simulated price at each t. We then take our predicted prices to be the average of these two paths (averages taken over each time t) as we work under the assumption that each price is equally likely (no arbitrage, see [1]). Figure 5 shows the predicted paths (orange) and the actual paths (blue) for each of the four stocks and table 1 shows the MSE for each prediction.

We see from figure 5 that GBM performs very well on AAPL and MSFT and this is reinforced by the low respective MSE's shown in table 1. The model also performs well on TSLA, but has a far higher MSE – perhaps something we could put down to the volatile nature of the TSLA stock (that is how 'jumpy' the price is). However, figure 5 shows poor performance of the model for GOOG, with our predicted prices behaving seemingly opposite to that of the trend of the actual prices albeit the apparently low MSE in table 1. It is important to note here that poor performance in one case, over one specific time period should not invalidate the efficacy of our model. We must remember the stock market is a complex object influenced by many factors and as such in a pricing model we cannot take all into account. It is very likely that in the data used to model GOOG we have an upward trend present, hence the upward trending output of our model, while in reality there was some event which caused growth to halt and decline to begin in the period we model – something our GBM model cannot account for.

	AAPL	**GOOG**	**TSLA**	**MSFT**
MSE	6.33	10.19	46.4	7.14

Table 1: MSE for each stock analysed for GBM Model.

(a) AAPL

(b) GOOG

(c) TSLA

(d) MSFT

Figure 5: Predicted vs Actual 14 day prices for stocks using GBM.

6.2 RNN Model

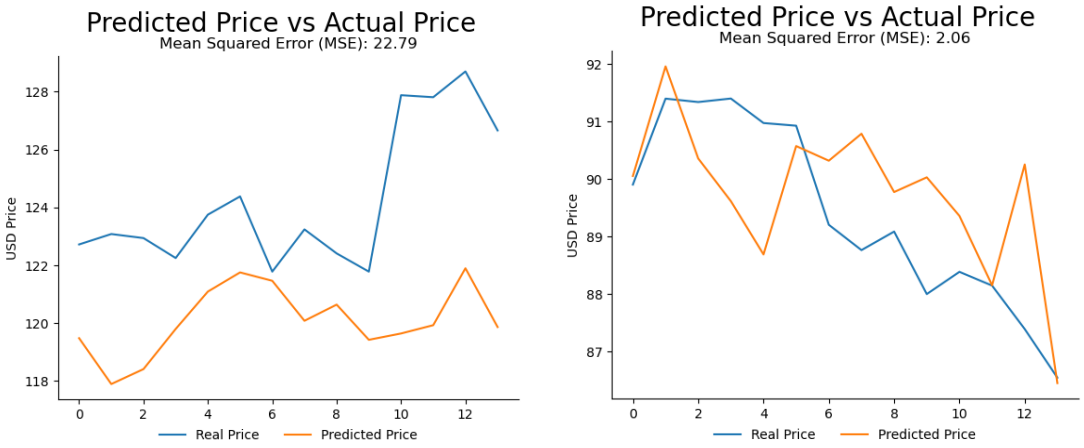
Execution of our RNN model was a simple. For each stock we took two months (42 days over the period 2020-10-01 to 2020-11-30) of trading data from Yahoo! Finance and pre-processed it to match the format of the input training data – that is Open, Close, High, Low, Volume, Average and Returns and all scaled to be between 0 and 1. We then fed this sequence into the relevant trained network and were provided with pricing predictions. Figure 6 shows the predicted paths (orange) and the actual paths (blue) for each of our four stocks and table 2 shows the MSE for each prediction.

Figure 6 shows that our model performs extremely well for three of the four stocks with regards to predicting the actual observed prices as well as trends. The RNN model appears to struggle when predicting prices for GOOG with large gaps between predicted and actual prices. However, the model yields a low MSE of 2.06 and captures the trend in prices that were actually observed well. In all cases, excluding GOOG, the models mimic where rises and falls occur in observed prices extremely well, evidenced by the fact predicted price curves show similar shapes to that of the actual price curves. Albeit the rises and falls in predicted price are neither of the same magnitude, nor the exact prices, we see our RNN gives very good indication of price behaviour – not only does the model predict trends well, but also when prices are rising and falling. Such quality is extremely desirable when modelling such seemingly random data.

NN capture subtle relationships between points in time series...

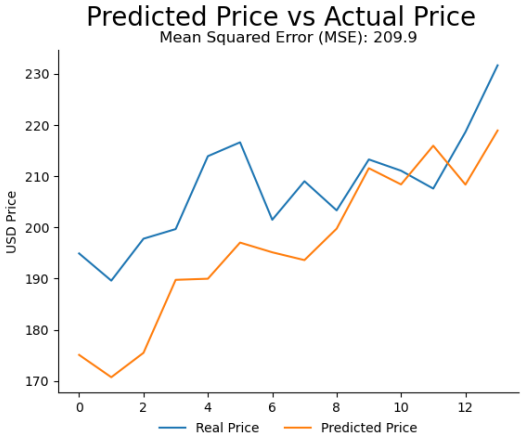
	AAPL	**GOOG**	**TSLA**	**MSFT**
MSE	22.79	2.06	209.9	6.02

Table 2: MSE for each stock analysed for RNN Model.

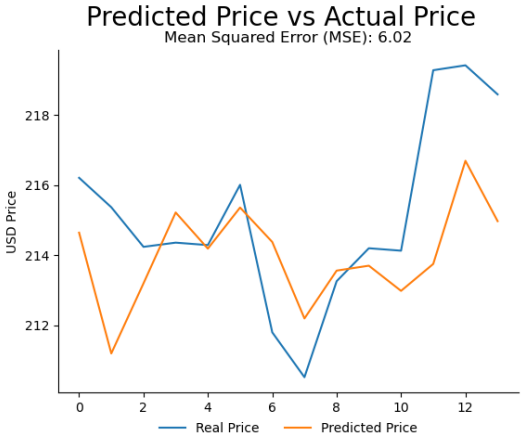


(a) AAPL

(b) GOOG



(c) TSLA



(d) MSFT

Figure 6: Predicted vs Actual 14 day prices for stocks using RNN.

7 Conclusion

Predicting stock prices is not an exact science. By nature of the stock market there is no way, that we are aware of yet, to exactly predict future prices and anyone who were to describe a method would become very rich very quickly. With that said, the methods described and demonstrated in this project are both effective means by which we can model stock prices.

Geometric Brownian Motion in the form described in this project provides a simple and easy to implement method by which to gain insight into the behaviour of a stock price. For the stocks we have chosen to analyse using this method, for the majority of cases, gives us valuable information regarding the trend in price over short periods of time based on previous trends in the historical data (via statistics from returns). Issues arise where GBM does not take more into account than recent trends, shown well here by the predictions made for GOOG. Stock prices are affected by many things. News, politics, economic events just to name a few and GBM is such a simplification that it simply cannot take these things into account. In a similar way our RNN model cannot do this, and makes its predictions based on previous trends. That said we must be aware of its ability to find more intricate patterns and relationships between input data when making its predictions thanks to the LSTM architecture we have used, hence the more effective predictions it makes.

Assessing efficacy of one model over another is a very subjective task and very situation dependant. Although table 1 shows lower MSE across the board for our GBM model than that of our RNN model, we see comparing figure 5 to figure 6 that pricing curve shapes are closer to the original in the case of our RNN network as well as more accurately showing present trends. Such properties of the RNN model thus demonstrate its feasibility as a model for future stock prices, with specific ability to find and predict movements/trends.

Overall, both models can be taken as effective predictors of future prices given the data given to them is accurate and we have good faith current trends will continue into the near future. Geometric Brownian Motion is a very well researched and widely used method (often in combination with other models) to make predictions, even in its simplest form. Our results here show that well trained and structured deep neural network based models are able to compete with such well established methods and offer accurate predictions for near future stock prices.

References

[1] Matteo Bottacini. Stochastic asset pricing in continuous time, 2021. URL https://github.com/bottama/stochastic-asset-pricing-in-continuous-time. Accessed: 06/12/23.

[2] Investopedia. What is the stock market, what does it do, and how does it work?, 2023. URL https://www.investopedia.com/terms/s/stockmarket.asp. Accessed: 2023-12-20.

[3] Mohammad Rafiqul Islam and Nguyet Nguyen. Comparison of financial models for stock price prediction. *Journal of Risk and Financial Management*, 2020.

[4] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, R. Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network, 1989. URL https://proceedings.neurips.cc/paper_files/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf.

[5] Joel Lidén. Stock price predictions using a geometric brownian motion, 2018.

[6] Martin Lotz. Mathematics of machine learning, 2020. URL https://homepages.warwick.ac.uk/staff/Martin.Lotz/files/learning/lectnotes-all.pdf.

[7] Zhijun Yang and D Aldous. Geometric brownian motion model in financial market. *Graduation Work, Princeton, USA. Yang*, 2012.