Recurrent Neural Networks in Forecasting S&P 500 index.

Samuel Asuquo Edet (samueledet@aims.ac.za)
African Institute for Mathematical Sciences (AIMS)

Supervised by: Prof. Ronnie Becker
Co-supervised by: Dr. Bubacarr Bah
African Institute for Mathematical Sciences, South Africa

18 May 2017

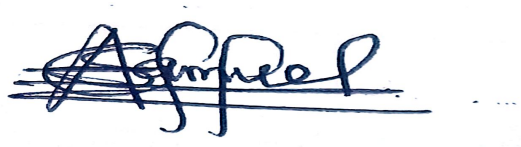
Submitted in partial fulfillment of a structured masters degree at AIMS South Africa

Abstract

The objective of this essay is to predict the movements of the S&P 500 index using variations of the recurrent neural network. The variations considered are the simple recurrent neural network, the long short term memory and the gated recurrent unit. In addition to these networks, we discuss the error correction neural network which takes into account shocks typical of the financial market. In predicting the S&P 500 index, we consider 14 economic variables, 4 levels of hidden neurons of the networks and 5 levels of epoch. From these features, relevant features are selected using experimental design. The selection of an experiment with the right features is chosen based on its accuracy score and its GPU time. The chosen experiments (for each neural network) are used to predict the upward and downward movements of the S&P 500 index. Using the prediction of the S&P 500 index and a proposed strategy, we trade the S&P 500 index for selected periods. The profit generated is compared with the *buy and hold strategy*.

Declaration

I, the undersigned, hereby declare that the work contained in this research project is my original work, and that any work done by others or by myself previously has been acknowledged and referenced accordingly.



Samuel Asuquo Edet, 18 May 2017

Contents

Abstract i

1 Introduction 1
- 1.1 Stock market . . . 1
- 1.2 Neural network . . . 2

2 Recurrent Neural Network 6
- 2.1 Cost function . . . 8
- 2.2 Back-propagation through time . . . 9
- 2.3 Exploding and vanishing gradient problem . . . 11
- 2.4 Solution to exploding and vanishing gradient problem . . . 11

3 LSTM, GRU and ECNN 13
- 3.1 Long short term memory . . . 13
- 3.2 Gated recurrent unit . . . 14
- 3.3 Error correction neural network . . . 15
- 3.4 Approximation theorem . . . 16

4 Data, Methodology and Results 19
- 4.1 Data compilation . . . 19
- 4.2 Implementation platform and procedures . . . 19
- 4.3 Design of experiment . . . 20
- 4.4 Experiments and results . . . 23
- 4.5 Trading model . . . 25

5 Conclusion 29

References 31

1. Introduction

Prediction of the financial market has long been a subject of research. This is largely due to the profit the market offers. Associated with the possibility of making profit is the possibility of making a loss. This makes the financial market a risky business to generate wealth. Some of the factors responsible for the risky nature of the financial market include volatility of interest or currencies responding to government economic policies or change in monetary policies. These result in non-stationarity and non-linearity of the financial time series. By non-linearity, we mean that often time there appears not to be a linear relationship between the factors that influence returns in the market. While by non-stationarity, we mean that financial data may have means and variances that change over time. The problem of non-linearity and non-stationarity makes predicting financial instruments a difficult task.

There are different techniques that can be used in forecasting financial instruments (e.g. stock market indices). Broadly speaking, we can classify these techniques into statistical and neural networks. The use of statistical techniques has proven less effective when compared to neural networks because neural networks are able to capture non-linearities in the market and most importantly, unlike statistical techniques, neural networks do not depend on the underlying assumptions of the independent variables used in the prediction. However, the disadvantage of the neural network techniques is that there is no formal way of designing a network. The choice of features and the tuning of different parameters in the design of the network is experimental.

We will give a brief overview of the global stock markets and neural networks.

1.1 Stock market

The stock market has proven to be a good platform for companies to raise capital for their businesses. This is done by publicly trading some percentage of the company's value through selling of shares. Thereby giving the public a measure of decision making on the company's activities. The stock market is sometimes considered an indicator of the economic growth of a country. In different countries, there are stock indexes that are reflections of the country's economic growth. Stock indexes are collections of stocks of different companies. For example, the Standard $\&$ Poor's 500 (S$\&$P 500) index is a portfolio of stocks from 500 companies considered to be a reflection of the US economy. The index accounts for 80$\%$ of the market capitalization of all the stocks listed on the New York Stock Exchange (NYSE). The stock index monitors change in the value of the underlying assets in its portfolio. Each of the underlying assets is given a weight that is proportional to the performance of the asset. The value of the S$\&$P 500 is computed by weight average market capitalization. This is simply the sum of the market capitalization of each of the underlying stock in the S$\&$P 500.

$$\text{Market Capitalization} = \text{Oustanding shares} \times \text{Company's current share price.}$$

In Figure 1.3, Apple computer is one of the underlying assets of S$\&$P 500 and Apple computer has 830 million outstanding shares with a current market price of $\$53.55$. The market capitalization of Apple computer is $\$44.45$billion (830 million $\times$ \$53.55). This value changes with change in the market value of the underlying asset in the index.

$$\text{Market weight of the stock} = \frac{\text{Market capitalization of the stock}}{\text{Total market capitalization}} \times 100\ .$$

The larger the market weight of the stock, the more impact each 1% change will have on the index.

	Market Capitalization
Exxon Mobil	$367,050,000,000
Bank of America	$168,750,000,000
Apple Computers	$44,450,000,000
New York Times	$4,070,000,000
:	:
S&P 500	$10,640,793,000,000

Figure 1.1: Market capitalization of assets in S&P 500 index.

Figure 1.2: S&P 500 non-linear market trend (Source: Yahoo finance).

1.2 Neural network

Neural networks can be explained using the concept of polynomial curve fitting. Polynomial curve fitting involves fitting a function to a set of data points such that the error function is minimized.

$$y(x) = a_0 + a_1 x + \ldots + a_k x^k. \tag{1.2.1}$$

Equation 1.2.1 can be regarded as a non-linear mapping with input x and output y. The precise form of $y(x)$ is determined by the parameters $a_0, a_1, ..., a_k$. These parameters are similar to the weights in a neural network. Suppose we have N data points in vectors x and y. We denote the data points that make up the input vector x as x^n and that of y as y^n. The polynomial can then be written as a functional mapping in the form $y = y(x^n; \mathbf{a})$, where $\mathbf{a}$ is the vector of the parameters. The curve fitting procedure involves minimizing the square of the error summed over the data points.

$$E = \frac{1}{N} \sum_{n=1}^{N} \left[y(x^n; \mathbf{a}) - y^n \right]^2. \tag{1.2.2}$$

Equation 1.2.1 represents the case of a uni-variate non-linear function. We can extend this to higher dimensions. By higher dimensions, we mean that the number of input variable is d, where $d > 1$. Thus we can consider a higher-order polynomial up to order 3.

$$y = a_0 + \sum_{i_1=1}^{d} a_{i_1} x_{i_1} + \sum_{i_1=1}^{d} \sum_{i_2=1}^{d} a_{i_1 i_2} x_{i_1} x_{i_2} + \sum_{i_1=1}^{d} \sum_{i_2=1}^{d} \sum_{i_3=1}^{d} a_{i_1} a_{i_2} a_{i_3} x_{i_1} x_{i_2} x_{i_3}. \tag{1.2.3}$$

An increase in the dimension of the input space will increase the number of parameters. Hence, there will be a need for a huge dataset in order to determine these parameters.

1.2.1 Neural network topology.

We can define a neural network topology in the context of graph theory as a graph (N, C), where N is the set of neurons and C is the set of connections. The relationship between the neurons by means of their connections gives rise to a network. Therefore, we can discuss the topology of the network based on the framework of the neurons and their connectivity. Most neural networks are layered i.e. they have the input layer, the hidden layer and the output layer. The neural network is referred to as a deep neural network if it has more than one hidden layer. Considering the connectivity of the neurons in the layers, there are different types of interconnection of neurons;

Interlayer connection: The connection of neurons in adjacent layers whose indices differ by one.

Intra-layer connection: The connection of neurons in the same layer. A neuron can also be connected to itself. In this case, the intra-layer connection is said to be a self connection.

Supra-layer connection: The connection of neurons in distinct layers, where the index of the distinct layers differ by at least 2.

The connections discussed above can either be full or partial. A fully connected network is a network that has all neurons connected, while a partially connected neural network has some of its neurons connected.

The connections help with the flow of information. The flow of information in the network is referred to as the *forward propagation* (the flow of information forward). This flow can be symmetric or asymmetric. When the flow of information is such that it is strictly fed forward i.e. direct, the neural network is a feed forward neural network. If the flow of information is such that it is fed forward and fed back (we do not mean back propagated, rather we mean information being loop), the network is a recurrent neural network.

1.2.2 Learning.

When information is fed into the network from the input vector, the network is responsible for learning the structure of the input data and extracting information from the system that is helpful to the learning task. There are three types of learning tasks:

Supervised learning: In supervised learning, the desired output is known. The learning is such that the training data set is learnt and used in predicting what the output should be. This prediction could either be classification e.g. predicting if the closing price of the stock for the next day goes up or down, or regression e.g. predicting the closing price of the stock for the next day. A classification task can be binary, as described in the example of the movement of stock or it can be multi-class classification e.g. handwritten digit recognition.

Regression task involves predicting continuous outcomes. In regression analysis, we use a number of predictors to estimate what the outcome should be. In the case of a single predictor, we have a linear regression and in the case of a multiple predictor, we have a multiple linear regression. Figure 1.3 below represents a linear regression of the variables x and y, where x is the predictor variable and y is the response variable or the output. We have that $y = f(x)$, where f is the line that best fits the data, in this case it is a straight line. Given the line, we can use the intercept of the line and the slope learned from the data to predict the response variable of a new set of data.

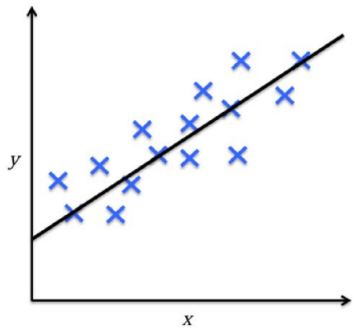


Figure 1.3: Linear regression.

Reinforcement learning: Reinforcement learning can be considered as a special case of supervised learning. In reinforcement learning, the network is designed in a way that incorporates rewards and

penalties (rewards for correct prediction, and penalties for wrong prediction). This forces the machine to learn and predict well. A classic example of reinforcement learning is the Matchbox Educable Naughts and Crosses Engine (MENACE) designed by Donald Michie. The game is built using 300 matchboxes, each containing nine beads of different colours representing the nine moves on a tic-tac board. If the matchbox wins the game, the matchbox is rewarded with three beads, if it loses, the saved beads in the boxes are removed and if the game ends in a draw, the matchboxes retain its original state.

Unsupervised learning: In unsupervised learning, we deal with data of unknown structure. The technique in unsupervised learning is to explore the structure of the dataset and extract information from the system without the knowledge of the output data.One technique that is used in exploring the data in order to give structure to the data is clustering.

Over time, different neural networks have been used in forecasting the global stock markets. S. Shen and H. Jiang proposed a prediction algorithm that exploits the temporal correlation among global stock markets. The algorithm alongside different regression algorithms was used to trace the actual increment in the markets. The results from their model indicated a prediction accuracy of 74.4 %, 76% and 77.6% for NASDAQ, S&P 500 and DJIA indexes respectively. A trading strategy was built based on their proposed model. Results of their simulations show that the trading strategy guaranteed higher profit when compared with the *buy and hold back* strategy (the stock is bought on the first day and all the stocks are sold on the last day). M. Dixon, D. Klabjan and J.H. Bang described the application of the deep neural network (DNN) in predicting financial market movement. The DNN was applied to back-testing a simple trading strategy over 43 different commodities and FX future mid-prices at five minute intervals. The problem with the prediction model of M. Dixon, D. Klabjan and J.H. Bang is that there was no formal argument for the selection of the features used in the model. Feature selection is important because if certain features that are highly correlated with the response variable (the stock to be predicted) are highly correlated with one another, it does not make sense to choose all (we select only one of them), since the neural network will extract the same information from them. Beyond the features having a strong correlation with the response variable, S. Shen and H. Jiang recommend that the closing time of the market be put into consideration. For example, in predicting S&P 500 index, if the closing price of the Hong Kong stock index (HSI) has a strong correlation with S&P 500 index, it can be chosen as a feature since the market closes before the opening of S&P 500 index. Niaki and Hoseinzade (2013) explored the use of design of experiment in feature selection. After checking the correlations of the features and eliminating them appropriately, Niaki and Hoseinzade (2013) placed the features into groups, each group having 2-factor levels of 1 and 0. Factor level 1 indicates the group is relevant to the response variable and factor level 0 indicates the group is not relevant to the response variable.

In this essay, we want to forecast the upward and downward movements of the closing price of S&P 500 index using variants of fully connected recurrent neural networks. These networks are: the simple recurrent neural network, the long short term memory, the gated recurrent unit and the error correction neural network. In selecting the features for training, we will apply the feature selection technique of Niaki and Hoseinzade (2013). After predicting the movements of the index, the proposed model in each of the networks will be used in building a trading strategy for some trading periods. The profit margin from the best neural network will be compared with the *buy and hold* trading strategy within the same trading period.

This work is organised as follows: Chapter 2 discusses the simple recurrent neural network with detail explanation of the backpropagation through time. Chapter 3 introduces variations of the recurrent neural networks: the long short term memory, the gated recurrent unit and the error correction neural network. In addition to that, we prove that recurrent neural networks can always approximate the structure of time-

variant dynamic system like the stock market. Analysis of the feature selection procedure, experiments, results and a simple trading strategy are discussed in Chapter 4. Finally, Chapter 5 summarizes the essay; highlighting the limitations of the recurrent neural network and ongoing works being done to improve the recurrent neural network.

2. Recurrent Neural Network

Recurrent neural networks have been an important focus of research in neural networks. This is because unlike the traditional artificial neural networks, recurrent neural network is most ideal for predicting sequential problems. Since there is a need for a neural network that takes into account feedback, the recurrent neural network appears most appropriate for a sequential prediction task. For example, in neuro-linguistic programming, recurrent neural network is used in learning the distributive representation of words and in the prediction of words given sequence of previous words. Many other applications involve dynamical systems with time sequence of events e.g. robotic engineering. The learning technique employed by recurrent neural networks is that it keeps track of the sequence of events. It does this using a complex activation unit in its layers. In this chapter, we will be concerned with the architecture of the simple recurrent neural network that has a simple activation unit in its hidden layers.

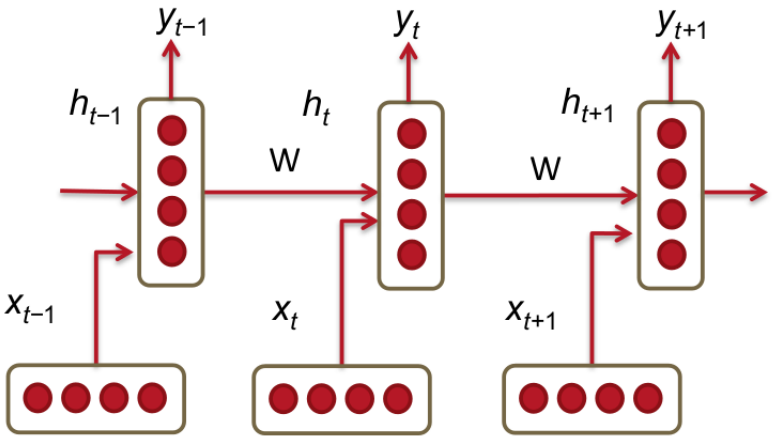


Figure 2.1: 3 - time step Recurrent Neural Network (RNN).

Suppose x_t is a sequence of input data and y_t is a sequence of output data at time step t. From Figure 2.1, the rectangular box represents the hidden layer at a time step t, holding a number of neurons. Each of these neurons performs a linear matrix operation with the input data x_t. The result of this operation is fed into an activation function (e.g. $\tanh$, sigmoid, softmax etc.) depending on the type of prediction in consideration. At each time step, the output of the hidden state h_{t-1} from the previous time step $t-1$ and the input x_t in the current time step is fed into the next hidden state h_t of the current time step t. The resulting linear matrix operation of these parameters is then fed into an activation function to produce a predicted output $\hat{y}_t$.

$$h_t = \sigma(W^{hh}h_{t-1} + W^{hx}x_t), \tag{2.0.1}$$

$$\hat{y}_t = \text{softmax}\ (W^{yh}h_t). \tag{2.0.2}$$

Equation 2.0.1 is used to compute the hidden layer output at each time step and (2.0.2) is used to compute the output. W^{hh} is the weight matrix connecting the hidden layers, W^{hx} is the weight matrix connecting the input layer to the hidden layer and W^{yh} is the weight matrix connecting the hidden layer to the output layer. Note that as in the case of a feed forward neural network, the weight matrices are shared parameters i.e. the same weights are used at different time steps. Suppose the dimensions of

the input layer, output layer and hidden layer are D_x, D_y and D_h respectively. Then, the dimensions of W^{hh}, W^{hx} and W^{yh} are $D_h \times D_h$, $D_h \times D_x$ and $D_y \times D_h$ respectively.

2.0.1 Example.

Consider a 2-time step recurrent neural network problem where we have to learn $y = f(x) = x^2$ such that $x \in \{-1, 1\}$. Given the weight matrices of the network below, use a $\tanh$ activation for the hidden state and a softmax activation to obtain the output.

$$W^{hx} = \begin{pmatrix} 0.15 & 0.20 \\ 0.40 & 0.25 \end{pmatrix}, \quad W^{hh} = \begin{pmatrix} 0.25 & 0.30 \\ 0.30 & 0.15 \end{pmatrix}, \quad W^{yh} = \begin{pmatrix} 0.25 & 0.15 \\ 0.30 & 0.30 \end{pmatrix}.$$

Solution:

Given the input x_t and target output y_t as;

$$x_t = \begin{pmatrix} -1 \\ 1 \end{pmatrix}, \quad y_t = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \forall t \in \mathbb{N}.$$

For time step $t = 1$, the hidden state h_1 (previous hidden state at the initial time step is zero);

$$h_1 = \begin{pmatrix} 0.15 & 0.20 \\ 0.40 & 0.25 \end{pmatrix} \begin{pmatrix} -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.05 \\ -0.15 \end{pmatrix}$$

Let the output from the h_1 layer be $out\ h_1$. Using the $\tanh$ activation function, we have;

$$out\ h_1 = \begin{pmatrix} \tanh 0.05 \\ -\tanh 0.15 \end{pmatrix} = \begin{pmatrix} 0.0499 \\ -0.1489 \end{pmatrix}$$

The input into the output layer is given as;

$$\hat{y}_1 = \begin{pmatrix} 0.25 & 0.15 \\ 030 & 0.30 \end{pmatrix} \begin{pmatrix} 0.0499 \\ -0.1489 \end{pmatrix} = \begin{pmatrix} -0.00986 \\ -0.0297 \end{pmatrix}$$

Next, we calculate the predicted output $\hat{y}_1$ from the output layer. Given vector $\hat{y}_j$, the softmax function softmax $(\hat{y}_j)$ is given as:

$$\text{softmax } (\hat{y}_j) = \frac{\exp^{\hat{y}_{ji}}}{\sum_{i=1}^{n} \exp^{\hat{y}_{ji}}}. \tag{2.0.3}$$

Using (2.0.3), we have that:

$$\hat{y}_1 = \begin{pmatrix} 0.5049 \\ 0.4950 \end{pmatrix}$$

For time step $t = 2$, the hidden state is h_2 (The previous hidden state is h_1);

The input from the input layer into the hidden layer h_2 gives;

$$\begin{aligned} h_2 &= W^{hh} h_1 + W^{hx} x_2 \\ &= \begin{pmatrix} 0.0178 \\ -0.1574 \end{pmatrix} \end{aligned}$$

Using the $\tanh$ activation, the output from the hidden layer h_2;

$$out\ h_2 = \begin{pmatrix} 0.0178 \\ -0.1560 \end{pmatrix}$$

The input from the hidden layer to the output layer is;

$$\hat{y}_2 = \begin{pmatrix} -0.01895 \\ -0.04146 \end{pmatrix}$$

Using the softmax function (2.0.3), the predicted output from the output layer;

$$\hat{y}_2 = \begin{pmatrix} 0.5056 \\ 0.4943 \end{pmatrix}$$

Note: Implementing the neural network in a programming language, the previous hidden layer is initialized to a zero vector and the weight matrices are initialized randomly.

A.M. Saxe, J.L. McClelland and S. Ganguli, proposed choosing a scaled initial condition on the weight matrices, such that the norm of the error vector is preserved as it is backpropagated through time. In the context of A.M. Saxe, J.L. McClelland and S. Ganguli work, they considered each weight to be independently identically distributed from a zero mean Gaussian with standard deviation $\frac{1}{\sqrt{N}}$, where N is the dimension. The choice of orthogonal weight matrices works well since the eigenvalues of an orthogonal matrix have a magnitude of 1. Hence computing matrix multiplications for a large number of time steps will most likely not result in a vanishing gradient or an explosion problem peculiar to recurrent neural networks. We will discuss this explicitly in subsequent sections.

2.1 Cost function

There are quite a number of cost functions that are used in neural networks. Examples include: mean square error and cost entropy error. Given total time step T, the cost function in a recurrent neural network is given as:

$$E = \frac{1}{T}\sum_{t=1}^{T} E_t\ , \tag{2.1.1}$$

where E_t is the error at each time step and can be calculated in different ways. Given that y_t is the output and $\hat{y}_t$ is the predicted output. The mean square error E_t is given as:

$$E_t = \frac{1}{2}(\hat{y}_t - y_t)^2. \tag{2.1.2}$$

While the cross entropy error E_t is given as:

$$E_t = -y_t \log \hat{y}_t\ . \tag{2.1.3}$$

H. Zimmermann, R. Neuer and R. Grothmann pointed out that the choice of a cost function in financial time series prediction should bring into consideration the problem of outliers and heteroscedasticity (a situation where the variance of the target variable changes over time). These outliers arise from

information shock that can be caused by government dividends paid by companies, change in political policies or a shift in the policies of world financial institutions. These shocks appear as discontinuities in the trajectory of an affected asset. To be robust against such shocks, H. Zimmermann, R. Neuer and R. Grothmann propose a smoother version of the mean square error function;

$$E_t = \frac{1}{a}\ln\cosh(a(\hat{y}_t - y_t)), \tag{2.1.4}$$

where the parameter $a > 1$ (the paper recommends $a \in [3, 4]$). The motivation behind this technique for training recurrent neural network is that $\tanh ax$ is a smooth approximation of (2.1.4) with the integral $\int \tanh(az)dz = \frac{1}{a}\ln\cosh(az)$.

2.1.1 Example.

Use the mean square error to compute the cost of prediction in Example 2.0.1.

Solution:

The cost of prediction is the sum of the cost for time steps $t = 1$ and $t = 2$;

$$E = E_1 + E_2 = \frac{1}{2}\left[\begin{pmatrix}0.5049\\0.4950\end{pmatrix} - \begin{pmatrix}1\\1\end{pmatrix}\right]^2 + \frac{1}{2}\left[\begin{pmatrix}0.5056\\0.4943\end{pmatrix} - \begin{pmatrix}1\\1\end{pmatrix}\right]^2.$$

Suppose the norm on the vector space is a $l^2 - norm$ i.e.

$$\|x - y\| = \left(\sum_{i=1}^{n} |x_i - y_i|^2\right)^{\frac{1}{2}}, \tag{2.1.5}$$

therefore

$$E = \frac{1}{2}(0.7072)^2 + \frac{1}{2}(0.7072)^2 = 0.500\ .$$

The property of the cost function is that it must be differentiable. This is necessary so that the gradient of the error that is being back propagated can be computed. The mean square error is considered a good estimator for regression problems. In the case of a classification problem, the cross-entropy function is considered a good estimator. The mean square error is a measure of the error around the regression line. Its underlying assumption is that the variance is homogeneous. However, in practice variance changes over time. Hence, this can sometimes result in faulty prediction.

2.2 Back-propagation through time

The back-propagation through time algorithm is an extension of the ordinary back-propagation used in the feed-forward neural network. The back-propagation algorithm is a propagated gradient descent algorithm, where the gradients of the cost function are propagated backward and used in updating the weight matrix. In updating the weight, we use any gradient descent algorithm for learning (e.g. stochastic gradient descent). In stochastic gradient descent, we have that the change in weight is proportional to the negative gradient of the cost function with respect to the weight.

$$\triangle W = -\eta\frac{\partial E}{\partial W}\ , \tag{2.2.1}$$

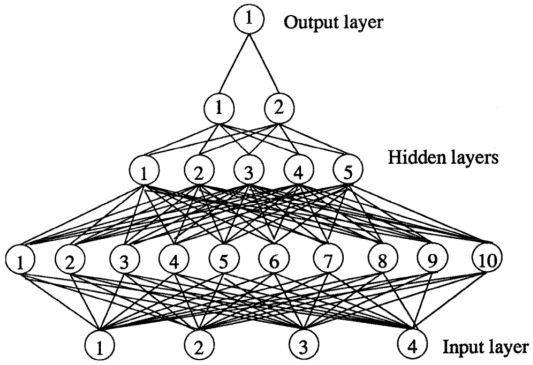


Figure 2.2: Architecture of the back-propagation.

where η is the learning rate. In implementing the RNN architecture, the learning rate is initially fixed and as the neural network learns, the learning rate is updated. To compute the gradient of the error $\frac{\partial E}{\partial W}$, we sum the gradient of the error at each time step t.

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W} \,. \tag{2.2.2}$$

To compute $\dfrac{\partial E_t}{\partial W}$ at each time step, we apply the chain rule;

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W} \,. \tag{2.2.3}$$

In (2.2.3), $\dfrac{\partial h_t}{\partial h_k}$ is the partial derivative of h_t with respect to all previous time steps $k = 1, ..., t-1$.

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \,. \tag{2.2.4}$$

Since the dimension of the hidden layer is D_h, each $\frac{\partial h_i}{\partial h_{i-1}}$ is the Jacobian matrix for the hidden layer.

$$\frac{\partial h_i}{\partial h_{i-1}} = [\frac{\partial h_i}{\partial h_{i-1},1} \cdots \frac{\partial h_i}{\partial h_{i-1},D_h}] = \begin{bmatrix} \frac{\partial h_i,1}{\partial h_{i-1},1} & \cdots & \frac{\partial h_i,1}{\partial h_{i-1},D_h} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_i,D_h}{\partial h_{i-1},1} & \cdots & \frac{\partial h_i,D_h}{\partial h_{i-1},D_h} \end{bmatrix} \,. \tag{2.2.5}$$

Substituting (2.2.4) in (2.2.3) and then in (2.2.2), we have

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \left(\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial W} \,. \tag{2.2.6}$$

2.3 Exploding and vanishing gradient problem

The backward pass of the gradient of the cost is susceptible to the explosion or vanishing problem. This is influenced by the choice of activation function and the initialization of the weight matrix. For example, in using a linear activation, the backward pass of the gradient cannot be squashed. Hence there is nothing to prevent the gradient from blowing up. Recall from (2.2.4), we have that;

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \,. \tag{2.3.1}$$

Defining a norm on the vector space, we have;

$$\|\frac{\partial h_t}{\partial h_k}\| = \| \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}\| = \| \prod_{i=k+1}^{t} M^T \times \text{diag}[f'(h_{j-1})]\| \leq (\alpha_W \alpha_h)^{t-k}, \tag{2.3.2}$$

where α_W and α_h represents the upper bound value of the matrices M^T and diag$[f'(h_{j-1})]$ respectively. If the time step is large, then $t-k$ will be large. Depending on the factor $\alpha_W \alpha_h$, if $\alpha_W \alpha_h < 1$, then we have that $(\alpha_W \alpha_h)^{t-k} \to 0$, this results in a vanishing gradient. If $\alpha_W \alpha_h > 1$, then $(\alpha_W \alpha_h)^{t-k} \to \infty$, this results in an exploding gradient. The explosion of the gradient results in an explosion of the curvature and higher order derivatives of the error function. Thereby, causing the same pattern in the error surface.

2.4 Solution to exploding and vanishing gradient problem

There are different techniques for handling the explosion and vanishing gradient problems. In this section, we will discuss the Thomas Mikolov solution that involves clipping the gradient to a small number whenever it explodes and the use of orthogonal initialization of the weight matrix to prevent the gradient from vanishing.

2.4.1 Gradient clipping.

The gradient clipping algorithm can be used as a technique in addressing the exploding problem. R. Pascanu, T. Mikolov and Y. Bengio proposed the need for a threshold to be set, such that when the gradient exceeds this threshold, the gradient is rescaled. The motivation behind this algorithm is based on the assumption that when the gradient explodes, the curvature and higher order derivatives explodes as well. This results in a specific pattern in the error surface (i.e. a valley with a single steep wall). The pseudo-code in Figure 2.3 explains how this is done.

From Figure 2.4, the solid line represents the training progress on each of the gradient descent step. When the gradient descent step approaches the wall, it bounces back far from the gradient landscape. However, the dotted lines represent the effect of the clipping gradient. When the gradient descent approaches the wall it is rescaled back such that it is not far away from the gradient landscape.

The proposed clipping is simple to implement and computationally efficient, but it does however introduce an additional hyper-parameter, namely the threshold. R. Pascanu, T. Mikolov and Y. Bengio proposed that a good heuristic for setting this threshold, is to look at the statistics on the average norm over a sufficiently large number of updates. In the experiment done in R. Pascanu, T. Mikolov and Y. Bengio, it was observed that for a given task and model size, the training is not very sensitive to this hyper-parameter and the algorithm behave well even for small thresholds.

$\hat{g} \leftarrow \frac{\partial E}{\partial W}$;
if $\|\hat{g}\| \geq threshold$ **then**
  $\hat{g} \leftarrow \frac{threshold}{\|\hat{g}\|}\hat{g}$;
end

Figure 2.3: Pseudo code for the gradient clipping technique (source: R. Pascanu, T. Mikolov and Y. Bengio).

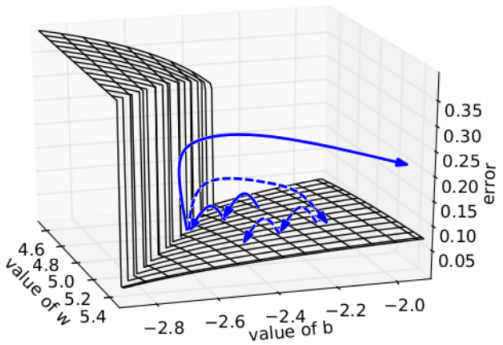


Figure 2.4: Error surface visualization of the gradient clipping technique (source: R. Pascanu, T. Mikolov and Y. Bengio).

2.4.2 Orthogonal initialization.

Hochcreiter and Schmidhuber (1997) discussed the different gradient based algorithm. However, these algorithms are prone to the vanishing gradient problem. Hochcreiter and Schmidhuber (1997) propose four solutions that can be applied in the learning process in order to avoid the vanishing gradient problem. One such solution is the use of a method that does not require the gradient information. Methods such as simulated annealing, muti-grid random search, random weight guessing etc. M. Henaff, A. Szlam and Y. Lecun considered a random weight guessing that involves orthogonal and identity transition matrices. The choice of an orthogonal matrix for the weight matrix is because orthogonal matrices are norm preserving i.e. given an orthogonal matrix O and a vector $\vec{x} \in V(+, *)$, we have that $\|O\vec{x}\| = \|\vec{x}\|$. The norm preserving property of the orthogonal matrix results from the absolute value of its eigenvalue being 1. Therefore, considering the matrix $O = DAD^{-1}$, where D is the matrix whose columns are the eigenvectors of O and A is the matrix whose diagonal entries are the eigenvalues of O. We have that $O^n = DA^nD^{-1}$. Since the absolute value of the eigenvalues of matrix O is 1, the matrix O^n is constant and the norm is preserved.

Other techniques that have been explored in addressing the vanishing and exploding gradient problem include; the use of the $L1/L2$ regularization term, the Hessian free optimization proposed by Martens and Sutskever (2011) and the echo-state network proposed by Lukosevicius and Jaeger (2009).

3. LSTM, GRU and ECNN

In Chapter 2, we discussed the simple recurrent neural network. We saw that this network can be trained using a gradient descent algorithm. However, the network is prone to the vanishing and explosion problem when we have a long term dependency (the time step is large). One way to address this problem is to use the clipping gradient algorithm technique to handle explosion or use an orthogonal initialization technique for the vanishing problem as discussed in the previous chapter. In addition to this, we can improve the dynamics of the network by using complex activation units rather than simple non-linear activations like the sigmoid function. There are a number of complex activation units that can be used, and we will consider a few of them.

3.1 Long short term memory

In theory, recurrent neural network can capture long-term dependencies. However, training them to do this is very hard and comes with computational cost. The motivation for using a complex activation unit like the long short term memory (LSTM), is to ensure that the network has a persistent memory i.e. it only keeps information relevant to the learning process. This will ensure that it is able to capture long term dependencies. Hochcreiter and Schmidhuber (1997) showed that LSTM can learn to bridge time intervals at an excess of 1000 steps even in the case of noisy, incompressible input sequences, without loss of short time lag capabilities. In comparison with real-time recurrent learning, back propagation through time, recurrent cascade correlation, Elman nets, and neural sequence chunking, LSTM leads to many more successful runs, and learns much faster. LSTM also solves complex, artificial long-time-lag tasks that have never been solved by previous recurrent network algorithms.

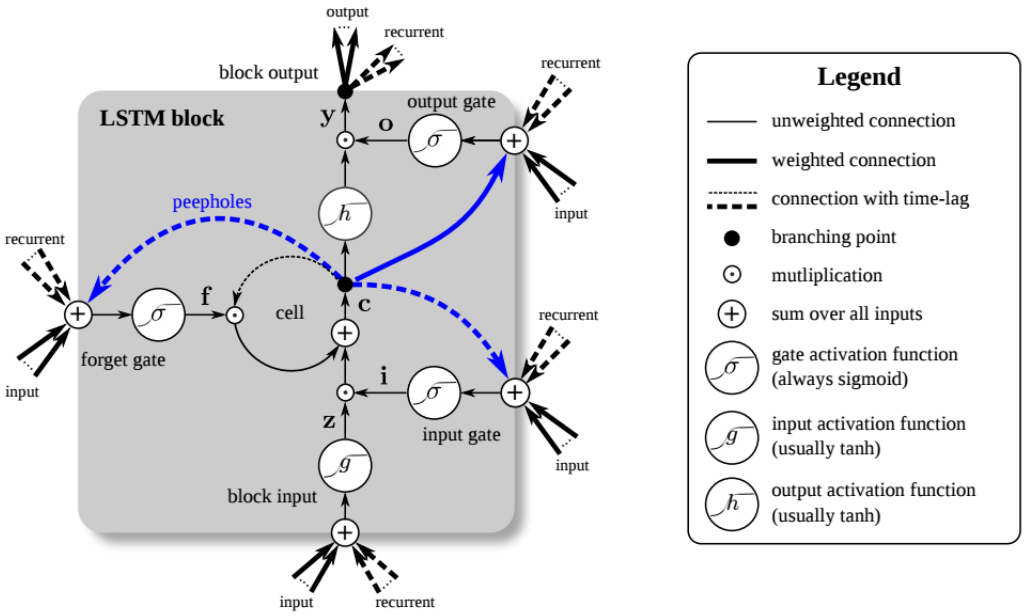


Figure 3.1: Long Short Term Memory (LSTM).

The architecture shown in Figure 3.1 has five basic operations:

Input gate: The function of the input gate in an LSTM architecture is to check if the new input is important, before generating memory to store it. It does this by taking into consideration the new input and the previous hidden state. Suppose i_t is the signal generated after accessing the relevance of the input data and σ be any activation function. The mathematical expression for this operation is;

$$i_t = \sigma\left(W_i^{hx} x_t + W_i^{hh} h_{t-1}\right). \tag{3.1.1}$$

Forget gate: The function of the forget gate is similar to that of the input gate. It takes into consideration the input data and the previous hidden state. It decides if previous memory is useful for the computation of the current memory.

$$f_t = \sigma\left(W_f^{hx} x_t + W_f^{hh} h_{t-1}\right). \tag{3.1.2}$$

New memory: The function of the new memory is to harmonize the newly observed input with the previous hidden state.

$$\tilde{h}_t = \tanh\left(W_n^{hx} x_t + W_n^{hh} h_{t-1}\right). \tag{3.1.3}$$

Final memory: The final memory checks the decisions made by the forget gate and input gate. From this, it acts accordingly to forget the past memory and to generate a new memory respectively.

$$\hat{h}_t = f_t \circ \hat{h}_{t-1} + i_t \circ \tilde{h}_t. \tag{3.1.4}$$

where $\circ$ is the dot product.

Output gate: The output gate is a very crucial gate in the architecture because it gauges what information from the final memory should be transferred to the hidden state. This enables a reduction in computational cost.

$$o_t = \sigma\left(W_o^{hx} x_t + W_o^{hh} h_{t-1}\right). \tag{3.1.5}$$

This output is used to gate the point-wise $\tanh$ of the memory.

$$h_t = o_t \circ \tanh \hat{h}_t. \tag{3.1.6}$$

According to Hochcreiter and Schmidhuber (1997), only the derivatives need to be stored and updated. Hence the LSTM algorithm is very efficient. The computational complexity of the LSTM algorithm is shown to be linear with respect to the weight matrix i.e. $O(W)$, where W is the number of weight matrices. This architecture has proven to learn long-term dependencies better than the simple recurrent network and has been applied to machine translation problem, image conversion to text, speech recognition and forecasting of exchange rates.

3.2 Gated recurrent unit

The gated recurrent unit (GRU) was proposed in the work of J. Chung, C. Gulcehre, K. Cho and Y. Bengio. The architectural structure of the GRU is similar to that of an LSTM. It has an input gate, forget gate, new memory, final memory whose functions are same as the LSTM. However, it has no

output gate. Recall that the output gate in an LSTM determines what information in the final memory should be stored in the hidden layer. In the case of the GRU, all the information is exposed to the hidden layer. J. Chung, C. Gulcehre, K. Cho and Y. Bengio describe GRU as follows: The hidden state is a linear combination of the previous hidden state and the new memory is given as:

$$h_t = (1 - z_t) \circ \tilde{h}_t + z_t \circ h_{t-1}, \tag{3.2.1}$$

where z_t is the update gate and $\circ$ is the element-wise multiplication. The update gate is computed as;

$$z_t = \sigma(W_z^{hx} x_t + W_z^{hh} h_{t-1}). \tag{3.2.2}$$

The operation of the update gate determines how much of the previous hidden state h_{t-1} should be forwarded to the hidden state h_t. If $z_t \approx 1$, the previous hidden state is copied out to the next hidden state. If $z_t \approx 0$, then only the new memory $\tilde{h}_t$ is forwarded to the hidden state. The computation of the new memory is similar to that of the LSTM.

$$\tilde{h}_t = \tanh(W^{hh}(r_t \circ h_{t-1}) + W^{hx} x_t), \tag{3.2.3}$$

where r_t is the reset gate. This gate determines how important the previous hidden state is to the new memory $\tilde{h}_t$.

$$r_t = \sigma(W_r^{hx} x_t + W_r^{hh} h_{t-1}). \tag{3.2.4}$$

3.3 Error correction neural network

H. Zimmermann, R. Neuer and R. Grothmann proposed a variation of the recurrent neural network called the error correction neural network (ECNN). The motivation behind this network is simply due to significant autonomous part in a system that limits our knowledge of the dynamics of the system. For example the presence of noise or shocks in the financial market limits the extraction of adequate information from the system. Under such conditions, it is necessary to use the information from the external shock to guide the dynamics of the network.Therefore, in addition to the external input x_t being fed into the network, the error in the previous time step is also fed in as additional input. The mathematical formulation of the dynamics of the system is given as:

$$h_t = \tanh(W^{hh} h_{t-1} + W^{hx} x_t + W \tanh(W^{yh} h_{t-1} - y_{t-1})) \tag{3.3.1}$$

$$\hat{y}_t = W^{yh}(h_t). \tag{3.3.2}$$

A new parameter W is introduced into the system. W is a weight matrix whose dimension corresponds to the dimension of the error $\hat{y}_i - y_i$. The system optimizes the weight matrices $W, W^{yh}, W^{hx}, W^{hh}$ such that the errors computed in the previous time steps are as minimal as possible.

$$\frac{1}{T}\sum_{t=1}^{T}(\hat{y}_t - y_t)^2 \rightarrow \min(W, W^{yh}, W^{hx}, W^{hh}) \tag{3.3.3}$$

Figure 3.2 represents the architecture of ECNN. We observe the new variable $z_{t-\tau}$, which is the measure of error in the previous hidden state i.e. $z_{t-\tau} = y_{t-\tau} - \hat{y}_{t-\tau}$. The aim of this network is to take into consideration the error as an input and optimize the system such that $z_{t-\tau} \rightarrow 0$. The network has two inputs, the first is the external input x_t that directly influences the hidden state h_t and the second is

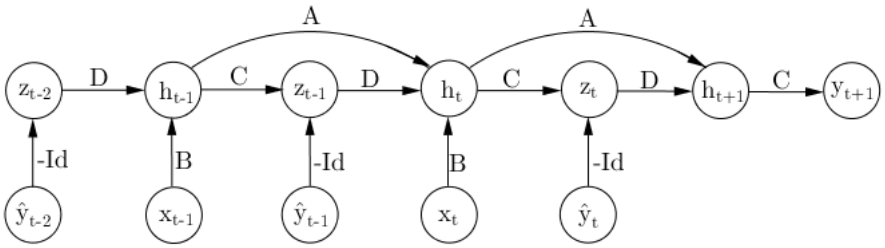


Figure 3.2: Error Correction Neural Network (ECNN).

the internal error z_{t-1} that affects state h_t. Another variable that can be seen is $-Id$, which is the negative identity matrix. This weight does not change in the learning process.

According to H. Zimmermann, R. Neuer and R. Grothmann, in time series forecasting, we often are faced with the difficulty of trend following behaviour of models. Trend following models are models that underestimate upward trends. This model failure can be identified by a sequel of non-alternating model errors $z_\tau = (y_\tau - \hat{y}_\tau)$. Therefore, to reduce the trend following tendencies, we enforce the alternating errors z_τ. This can be achieved by adding a penalty term to the error function of the error correction neural network.

$$\frac{1}{T}\sum_{t=1}^{T}(y_t - \hat{y}_t)^2 + \lambda(z_t - z_{t-1})^2 \to \min(A, B, C, D) \tag{3.3.4}$$

The additional penalty term $\lambda(z_t - z_{t-1})^2$ is used to minimize the residual auto-covariance in order to avoid trend following behaviour. H. Zimmermann, R. Neuer and R. Grothmann observe that predictions are not sensitive to the choice of λ.

3.4 Approximation theorem

At this point, we have discussed the non-linearity of the financial market and have seen that the closing price changes disproportionately with change in time (i.e. closing price is heteroscedastic). Also, we have discussed varieties of the recurrent neural network, that include: the simple recurrent network, LSTM, GRU and ECNN. The question, we ought to ask is: what is the mathematical proof that guarantees that a recurrent neural network can approximate trends in the financial market?

Dynamical System: Let $\mathbf{x} \in \mathbb{R}^n$ be a point in a Euclidean space. A system that describes the time dependence of $\mathbf{x}$ is known as dynamical system i.e. $\mathbf{y}(t) = \dot{\mathbf{x}}(t) = \mathbf{F}(\mathbf{x}(t))$. The evolution of the closing price in the stock market is an example of a dynamical system. A dynamical system can be variant or invariant in time. By time invariant, we mean that the measure of change in the time scale is same as the measure of change in the output i.e. $\mathbf{y}(t-\tau) = \mathbf{F}(\mathbf{x}(t-\tau))$. While a time variant dynamical system simply means that the system does not depend on time. The financial market is an example of a continuous time variant dynamical system.

Therefore, prior to the application of the recurrent networks discussed so far to forecasting the movement of S&P 500 closing price, we will prove that the recurrent neural network can approximate the dynamical

structure of this time variant dynamical system. Before the proof, we define some terms and assume the reader is familiar with the concept of a metric space.

3.4.1 Definition.

A recurrent neural network can be described as;

$$\dot{\mathbf{z}}(t) = \sigma(W_1, \mathbf{z}(t), W_2, \mathbf{u}(t)), \tag{3.4.1}$$

where $\mathbf{z} = (\mathbf{y}, \mathbf{h}) \in \mathbb{R}^{n+N}$ is the overall state of the neural network, $\mathbf{y} \in \mathbb{R}^n$ is the state of the network output, $\mathbf{h} \in \mathbb{R}^N$ is the hidden neural state and $\mathbf{u} \in \mathbb{R}^m$ is the input vector.

3.4.2 Definition.

Let (X, d) be a metric space. A subset $A \subseteq X$ is open if $\forall x \in A$, there exist an open ball $B(x, \epsilon)$ such that $B(x, \epsilon) \subseteq S$.

3.4.3 Definition.

Let $A \subseteq X$ be the subset of a metric space and $G = \{G_i : i \in \mathbb{I}\}$ be the collection of open subsets of X. Then G is said to cover A if $A \subseteq \bigcup_{i\in\mathbb{I}} G_i$.

3.4.4 Remark.

- If G is a collection of open sets, then G is open.
- If $H \subseteq G$ covers A, then H is a subcover of A.

3.4.5 Definition. Let (X, d) be a metric space. A subset $A \subseteq X$ is said to be compact, if every open cover of A has a finite subcover.

3.4.6 Lemma.

Let $A \subset \mathbb{R}^n$ and $B \subset \mathbb{R}^m$ be open sets, $D_A \subset A$ and $D_B \subset B$ be compact sets and $\mathbf{F} : A \times B \to \mathbb{R}^n$ be a C^1-class vector valued function (i.e. class of first order continuously differentiable functions). Given an invariant dynamic system of the form;

$$\dot{\mathbf{x}}(t) = \mathbf{F}(\mathbf{x}(t), \mathbf{u}(t)), \quad \mathbf{x} \in A, \mathbf{u} \in B, t \in \mathbb{R}, \tag{3.4.2}$$

with initial state $\mathbf{x}(0) \in D_A$. Then for an arbitrary $\epsilon > 0$, there exist an integer N and a recurrent neural network in the form (3.4.1), such that for any bounded input $\mathbf{u} : \mathbb{R} \to D_B$,

$$\max_{t\in[0,T]} \|\mathbf{x} - \mathbf{y}\| < \epsilon, \quad 0 < T < \infty \text{ holds.}$$

The lemma above is for a time invariant dynamical system and the proof can be seen in Funahashi and Nakamura (1993). We will extend Lemma 3.4.6 to a time variant dynamical system.

3.4.7 Theorem.

Let $A \subset \mathbb{R}^n$ and $B \subset \mathbb{R}^m$ be open sets, $D_A \subset A$ and $D_B \subset B$ be compact sets (i.e. closed and bounded), and $\mathbf{F} : A \times B \to \mathbb{R}^n$ be a C^1 class vector valued function. Given a time-variant dynamic system of the form;

$$\dot{\mathbf{x}}(t) = \mathbf{F}(\mathbf{x}(t), \mathbf{u}(t), t), \quad \mathbf{x} \in A, \mathbf{u} \in B, t \in \mathbb{R}, \tag{3.4.3}$$

with initial state $\mathbf{x}(0) \in D_A$. *Then for an arbitrary* $\epsilon > 0$, *there exist an integer* N *and a recurrent neural network in the form* (3.4.1), *such that for any bounded input* $\mathbf{u} : \mathbb{R} \to D_B$,

$$max_{t\in[0,T]}\|\mathbf{x} - \mathbf{y}\| < \epsilon, \quad 0 < T < \infty \text{ holds.}$$

Proof.

Let $\overline{\mathbf{x}}(t) = \begin{pmatrix} \mathbf{x}(t) \\ t \end{pmatrix} \in \mathbb{R}^{n+1}$. We can write the time variant system (3.4.3) as a time invariant system (3.4.2);

$$\dot{\overline{\mathbf{x}}}(t) = \overline{\mathbf{F}}(\overline{\mathbf{x}}(t), \mathbf{u}(t)), \quad \overline{\mathbf{x}} \in A \times \mathbb{R}, \mathbf{u} \in B, \tag{3.4.4}$$

where $\overline{\mathbf{F}}(\overline{\mathbf{x}}(t), \mathbf{u}(t)) = \begin{pmatrix} \mathbf{F}(\mathbf{x}(t), \mathbf{u}(t), \overline{\mathbf{x}}_{n+1}(t)) \\ 1 \end{pmatrix}$. Since $\overline{\mathbf{F}}_{n+1} = 1$ is first order continuously differentiable. Therefore $\overline{\mathbf{F}}$ is a C^1-class vector valued function.

The initial condition of the dynamical system (3.4.3) is given as $\overline{\mathbf{x}}(0) = \begin{pmatrix} \mathbf{x}(0) \\ 0 \end{pmatrix}$. From the theorem, we have that $\mathbf{x}(0) \in D_A$, where $D_A \subset A$ is a compact set. Therefore $\overline{\mathbf{x}}(0) \in D_A \times [0, T]$, where $[0, T] \subset \mathbb{R}$ is compact. Since the cartesian product of compact set is compact, we have that $D_A \times [0, T]$ is a compact subset of $A \times \mathbb{R}$.

Therefore, we have that the time invariant system (3.4.4) satisfies Lemma 3.4.6. Hence,

$$\max_{t\in[0,T]}\|\overline{\mathbf{x}}(t) - \overline{\mathbf{y}}(t)\| < \epsilon. \tag{3.4.5}$$

The neural state is given as $\mathbf{z} = (\overline{\mathbf{y}}, \overline{\mathbf{h}}) \in \mathbb{R}^{n+N}$, where $\overline{\mathbf{y}} \in \mathbb{R}^{n+1}$and $\overline{\mathbf{h}} \in \mathbb{R}^{N-1}$. Since $\overline{\mathbf{y}} = (\mathbf{y}, \overline{\mathbf{y}}_{n+1}) \in \mathbb{R}^{n+1}$, then $\mathbf{y} \in \mathbb{R}^n$ is the neural output state for the first n units. The Euclidean distance between $\overline{\mathbf{x}}$ and $\overline{\mathbf{y}}$ is given as;

$$\|\overline{\mathbf{x}} - \overline{\mathbf{y}}\|^2 = \|\mathbf{x} - \mathbf{y}\|^2 + (\overline{\mathbf{x}}_{n+1} - \overline{\mathbf{y}}_{n+1})^2. \tag{3.4.6}$$

Therefore,

$$\max_{t\in[0,T]}\|\mathbf{x}(t) - \mathbf{y}(t)\| \leq \max_{t\in[0,T]}\|\overline{\mathbf{x}}(t) - \overline{\mathbf{y}}(t)\| < \epsilon. \tag{3.4.7}$$

□

The proof above shows that the finite time trajectory of a time variant dynamical system can be approximated by the output of a neural network.

4. Data, Methodology and Results

In this chapter, we will be discussing the systematic exploration of the design of experiment (DOE) used in the feature selection for the model. The results of the simple recurrent neural network, long short term memory, gated recurrent unit and error correction neural network will be compared to determine the best model for forecasting the S&P 500 index. Next, we will build a trading strategy based on each model and calculate the profit generated within 5 trading days. This profit will be compared with the profit generated using the buy and hold strategy for the same trading period.

4.1 Data compilation

The dataset for this essay contains the daily closing price of the S&P 500 index and 14 financial independent variables considered to possibly have an impact on the closing price of the S&P 500 index. These independent variables are called the features or predictors from which we will use the DOE technique to choose the features that are most relevant to the prediction. The dataset has 2110 daily closing price for the predictors and the S&P 500 index from 1st January, 2009 to 1st April, 2017.

Table 4.1: Features for forecasting S&P 500 index.

Features	**Description**
SPYt-1,SPYt-2,SPYt-3	Previous daily price of S&P 500 index
HIS, FCHI, FTSE, GDAXI, IXIC, DJI	Stock indices
XOM, PG, GE, MSFT, JNJ	SPY asset with strong weight average
25, 50, 75, 100	Number of neurons in the hidden layer
40, 80, 120, 160, 200	Number of epoch

From Table 4.1, SPYt-1,SPYt-2, SPYt-3 represents the closing price of day $t-1, t-2, t-3$ respectively. For the indices, we have that: HIS is the Hang Seng index, FCHI is the CAC 40 index in France, FTSE is the financial times stock exchange of 100 companies in the UK, GDAXI is the German DAX index, IXIC represents the NASDAQ composite index and DJI is the Dow Jones Industrial average index both in the US. The ticker symbols XOM, PG, GE, MSFT and JNJ in Table 4.1 represent the stock of Exxon Mobil Corporation, Procter & Gamber Co, General Electric company, Microsoft Corporation and Johnson & Johnson. We consider the returns of SPYt-1,SPYt-2, SPYt-3 and the returns of the previous days of the stock indices and the stock of the companies. In calculating the returns (log returns), we use the formulae:

$$\text{Return} = \ln\left(\frac{x_t}{x_{t-1}}\right). \tag{4.1.1}$$

The other features in Table 4.1 are the number of neurons in the hidden layer and the number of epochs. An epoch is simply a full training cycle on a training set (training set is a percentage of the dataset used for training).

4.2 Implementation platform and procedures

We used the Amazon web service (AWS) platform for computation. This is because the service offers efficient computational power and large database storage. The codes of the neural network were written

using Python programming language and can be accessed from https://github.com/samueledet/AIMS_essay_codes. Specifically, we used Keras, which is an open source neural network library written in Python. The reason for using Keras is because it saves time spent in building the neural network from scratch. Using the AWS, we built an Amazon machine image (AMI) that has the Keras and other neceessary library installed. We specifically used the t2.micro instance having 1 virtual central processing unit (vCPU) and 1 Gigabyte (GiB) memory for experimenting with small epochs. Then we proceeded to use the g2.2xlarge instance having 8vCPU and 15GiB memory for larger epochs.

The procedure for implementation is as follow: we import the daily closing price of the financial variables we want to use for prediction. These variables were imported from the Yahoo finance website. Next, for each of the financial variable, we insert the mean of the closing price in place of the missing data. To obtain the closing price of the three previous days, we define a function that takes S&P 500 index and gives the closing prices of the previous three days. These new features (the previous three days closing price) are then added to the existing features. Next, we find the correlation of all the features and select the features relevant for prediction based on the discussion in Section 4.3. After selecting the features that are relevant for prediction, we find the difference in the daily log returns and where the difference is positive, this is classified as 1 (upward movement) and where the difference is negative, this is classified as 0 (downward movement). This process is performed on the whole dataset before splitting it to train set (80% of the dataset) and test set(20% of the dataset). The train set is used to learn the structure of the dataset. After training, we test the neural network on the test dataset. The quality of prediction is measured based on the following metrics; misclassified sample, accuracy, confusion matrix and classification report. After prediction, we propose a trading strategy and use the propose strategy alongside the prediction made by the neural networks to simulate the profit made for 1 week, 1 month, 3 months, 6 months, 9 months and 1 year trading periods. These profits are compared with what is obtainable using the buy and hold strategy.

4.3 Design of experiment

The motivation behind experimental design stems from resource limitation. Sometimes we may be dealing with multiple independent variables and the experiments to be performed may be too large for the resources we have at hand. Beyond selecting the features for prediction, we may also be interested in knowing if there is a main effect or an interaction effect among the independent variables. There are different design alternatives; the complete factorial design, the fractional factorial design, the individual experiments etc. The complete factorial design considers all possible experimental conditions, the fractional factorial design involve a fraction of all possible experimental conditions. These experimental conditions are chosen such that the balance property (the number of times the levels of each of the factor appear in the design should be the same) is preserved. Finally, the individual experiments require that each of the factors should be used separately for the experiments. This design will not be relevant to us, since it is not possible to use only the neurons or epoch as factors without the stocks or index.

In the language of experimental design, our features are referred to as factors. These factors are experimental and quantitative. They are experimental because we can set the levels and they are quantitative because we can assign a specified level of a quantitative factor. In this essay we will assign 1 and 0 to be the levels associated with the factors. 1 indicates that the factor is relevant to the prediction and 0 indicates otherwise. We have the company stocks and stock index as factors with 2 levels, the number of neurons has 4 levels (25, 50, 75, 100) and the number of epoch has 5 levels (40, 80, 120, 160, 200). Therefore, we are dealing with $2^{14} \times 4 \times 5$ factorial design. These imply we

have 327680 experiments to do. It is impractical for us to do this number of experiments. Therefore in building a practical design, we will compute the pairwise correlation of all the factors. This is because, if the factors are strongly correlated, then the network will extract the same information from them, so it does not make sense to keep all the factors (one of them is sufficient). By strongly correlated, we mean that the absolute value of their correlation is at least 0.5.

From the table below, we first consider those factors that have a strong correlation with SPY. Those factors include; SPYt-1, GE, MSFT, PG, JNJ, DJI, IXIC. Since GE, MSFT, PG and JNJ belong to the same group and each pair of them are strongly correlated, we will select only JNJ (since JNJ is most correlated with SPY). Also since DJI and IXIC are in the same group and are strongly correlated, we will select the index that is more correlated with SPY. Hence we select IXIC.

Table 4.2: Correlation of the features.

	SPY	SPYt-1	SPYt-2	SPYt-3	FCHI
SPY	1.000000	-0.496888	0.006337	-0.003752	0.005153
SPYt-1	-0.496888	1.000000	-0.496892	0.006336	-0.004540
SPYt-2	0.006337	-0.496892	1.000000	-0.496901	0.010580
SPYt-3	-0.003752	0.006336	-0.496901	1.000000	-0.037684
FCHI	0.005153	-0.004540	0.010580	-0.037684	1.000000
FTSE	-0.108714	0.004658	-0.024392	-0.036033	0.378694
GDAXI	-0.112388	-0.003895	0.006566	-0.011610	0.344643
XOM	-0.415252	0.012621	-0.014121	0.008859	0.113898
GE	-0.483895	0.002849	0.002000	-0.008233	0.094181
MSFT	-0.468465	0.002053	-0.005670	0.000472	0.065550
PG	-0.482705	0.002723	-0.006275	0.006816	0.057984
JNJ	-0.485034	0.004802	-0.001488	-0.000721	0.035067
DJI	-0.492941	0.006909	-0.005361	-0.000986	0.096512
IXIC	-0.494649	0.006205	-0.005072	-0.001746	0.074634

	FTSE	GDAXI	XOM	GE	MSFT
SPY	-0.108714	-0.112388	-0.415252	-0.483895	-0.468465
SPYt-1	0.004658	-0.003895	0.012621	0.002849	0.002053
SPYt-2	-0.024392	0.006566	-0.014121	0.002000	-0.005670
SPYt-3	-0.036033	-0.011610	0.008859	-0.008233	0.000472
FCHI	0.378694	0.344643	0.113898	0.094181	0.065550
FTSE	1.000000	0.186405	0.260738	0.239626	0.225608
GDAXI	0.186405	1.000000	0.158445	0.255542	0.229085
XOM	0.260738	0.158445	1.000000	0.843109	0.728928
GE	0.239626	0.255542	0.843109	1.000000	0.926529
MSFT	0.225608	0.229085	0.728928	0.926529	1.000000
PG	0.235010	0.221170	0.843379	0.943994	0.910331
JNJ	0.221451	0.195103	0.815062	0.945740	0.953022
DJI	0.265053	0.261944	0.862835	0.970866	0.929466
IXIC	0.237530	0.260123	0.818048	0.967740	0.951511

	PG	JNJ	DJI	IXIC
SPY	-0.482705	-0.485034	-0.492941	-0.494649
SPYt-1	0.002723	0.004802	0.006909	0.006205
SPYt-2	-0.006275	-0.001488	-0.005361	-0.005072
SPYt-3	0.006816	-0.000721	-0.000986	-0.001746
FCHI	0.057984	0.035067	0.096512	0.074634
FTSE	0.235010	0.221451	0.265053	0.237530
GDAXI	0.221170	0.195103	0.261944	0.260123
XOM	0.843379	0.815062	0.862835	0.818048
GE	0.943994	0.945740	0.970866	0.967740
MSFT	0.910331	0.953022	0.929466	0.951511
PG	1.000000	0.956926	0.966192	0.951937
JNJ	0.956926	1.000000	0.952219	0.960124
DJI	0.966192	0.952219	1.000000	0.989273
IXIC	0.951937	0.960124	0.989273	1.000000

Therefore, the number of factors have been reduced to 3 factors (SPYt-1, JNJ and IXIC) each having 2 levels, 4-levels of the number of neurons and 5-levels of the number of epochs. Now, we have a $2^3 \times 4 \times 5$ design i.e.. 160 experiments. However, we cannot have SPYt-1, JNJ and IXIC all have level 0. Therefore the experiments reduces to 140 experiments. We will not be doing the complete factorial design, we will use the fractional factorial design and consider $\frac{1}{7}$ of the complete factorial design i.e.. 20 experiments. In the design, SPYt-1, JNJ and IXIC will all have the level 1, while the levels of the hidden neuron and epoch will vary. We will be performing 20 experiments, each for the simple recurrent neural network, long-short term memory and gated recurrent unit.

Table 4.3: Design of experiment.

Experiments	SPYt-1	JNJ	IXIC	Neurons	Epochs
1	1	1	1	25	40
2	1	1	1	25	80
3	1	1	1	25	120
4	1	1	1	25	160
5	1	1	1	25	200
6	1	1	1	50	40
7	1	1	1	50	80
8	1	1	1	50	120
9	1	1	1	50	160
10	1	1	1	50	200
11	1	1	1	75	40
12	1	1	1	75	80
13	1	1	1	75	120
14	1	1	1	75	160
15	1	1	1	75	200
16	1	1	1	100	40
17	1	1	1	100	80
18	1	1	1	100	120
19	1	1	1	100	160
20	1	1	1	100	200

4.4 Experiments and results

From Table 4.3, we will be performing 20 experiments for each of the simple recurrent neural network, long short term memory and gated recurrent unit using the t2.micro and g2.2xlarge instances on the Amazon Web Service. The aim is to choose the best experiment base on their accuracy score. In the dataset, we have 2110 closing price that has been converted to logarithmic returns using (4.1.1). Next, in preprocessing the dataset, we standardize the log returns such that the log returns have zero mean and unit variance. This is done using (4.4.1) below. In (4.4.1), μ and σ represent the mean and standard deviation of the log returns respectively.

$$x_{new} = \frac{x - \mu}{\sigma}. \tag{4.4.1}$$

The process of standardizing helps in providing same scale for the features to be compared. Next, we allocate 80% (1688 data points) of the dataset as training set and 20% (422 data points) as testing set.

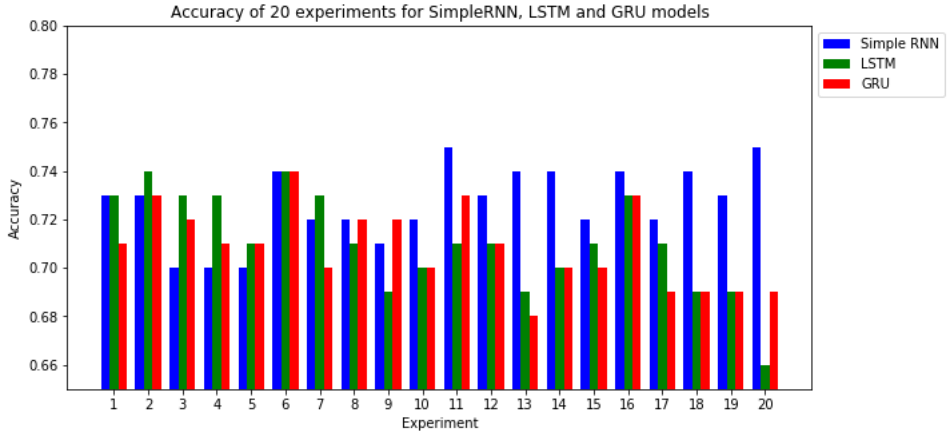


Figure 4.1: Accuarcy of 20 experiments for SimpleRNN, LSTM and GRU.

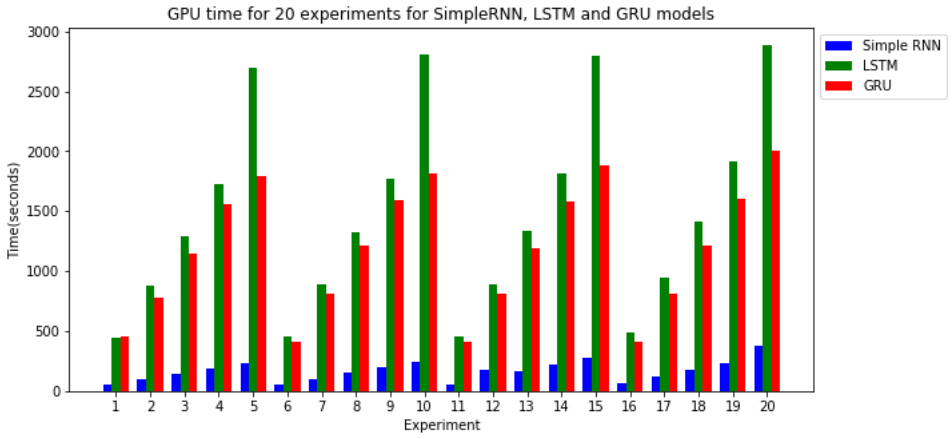


Figure 4.2: GPU time of 20 experiments for SimpleRNN, LSTM and GRU.

From Figure 4.1, we can observe that for the simple recurrent neural network, experiments 11 and 20 yield the best accuracy score, their accuracy score being 75%. However, from Figure 4.2, comparing

the GPU time for both experiments, the GPU time for experiment 11 is 58.6347 seconds and that of experiment 20 is 373.436 (approximately six times more than experiment 11). Based on this information, the neural model that will be used for prediction will be experiment 11. Hence, the simple recurrent neural network model will have the following features; SPYt-1, JNJ, IXIC, 75 hidden neurons and 40 epochs. For the long short term memory, experiments 2 and 6 had the best accuracy score of 74%. However, comparing the GPU time for both experiments, we have that the GPU time for experiments 2 and 6 are 880.36 seconds and 455.511 respectively. Based on the GPU time, we will consider experiment 2 as long short term memory neural network model. This model has the following features; SPYt-1, JNJ, IXIC, 25 neurons and 80 epochs. Similarly, for the gated recurrent neural network model, we will choose experiment 6 (accuracy score of 74% and GPU time of 409.454). The features of this model are; SPYt-1, JNJ, IXIC, 50 neurons and 40 epoch. We observe that on average, experiment 6 performs best in the three different neural networks with a mean accuracy score of 74%. Table 4.4 captures the classification report and confusion matrix of the chosen experiments. The classification report gives information about the precision, recall, accuracy and F1-score. The precision metric is the ratio of correctly predicted class to the total predicted class, recall measures the sensitivity of the class, F1-score is the weighted average of precision and recall, and support is simply the total observations we have. Let $i = 0, 1$ be the binary class, t_0, t_1, f_0 and f_1 be correctly classified 0, correctly classified 1, misclassified 0 and misclassified 1 respectively. We denote precision, recall and F1-score of a class as p_i, r_i and f_i respectively. Therefore the indicators in Table 4.4 can be calculated as follows:

$$p_i = \frac{t_i}{t_i + f_i}\,, \quad r_i = \frac{t_i}{t_i + f_{\sim i}}\,, \quad f_i = \frac{2p_i r_i}{p_i + r_i}\,.$$

$$\text{Accuracy} = \frac{t_i + t_{\sim i}}{t_i + t_{\sim i} + f_i + f_{\sim i}}\,.$$

(Note: $\sim i$ denotes not i. If $i = 0$, then $\sim i = 1$ vis-a-vis).

Table 4.4: Classification report and confusion matrix for SimpleRNN, LSTM and GRU models.

Simple RNN model				
Classification Report				
Binary class	Precision	Recall	F1-score	Support
0	0.80	0.73	0.76	55
1	0.69	0.77	0.73	44
Misclassified Sample = 25				
Accuracy = 75%				

Simple RNN model		
Confusion matrix		
Binary class	0	1
0	40	15
1	10	34

LSTM model				
Classification Report				
Binary class	Precision	Recall	F1-score	Support
0	0.80	0.71	0.75	55
1	0.68	0.77	0.72	44
Misclassified Sample = 26				
Accuracy = 74%				

LSTM model		
Confusion matrix		
Binary class	0	1
0	39	16
1	10	34

GRU model				
Classification Report				
Binary class	Precision	Recall	F1-score	Support
0	0.81	0.69	0.75	55
1	0.67	0.79	0.73	44
Misclassified Sample = 26				
Accuracy = 74%				

GRU model		
Confusion matrix		
Binary class	0	1
0	39	16
1	10	34

In predicting the movement of the S&P 500 index for the next 99 days, the selected model for the simple recurrent neural network model predicted 74 trends correctly. Out of the 55 downward trends, only 15 were misclassified as upward trends. Also, out of the 44 upward trends, 10 were misclassified as downward trends. The long-short term memory model and the gated recurrent unit model both forecast 73 trends correctly, they misclassified 16 downward trends as upward trends and 10 upward trends as downward trends.

4.5 Trading model

In the previous section, we have discussed how the neural networks can be evaluated using the number of correctly predicted movements. However, investors are more concerned with the profit they stand to earn. Hence, we will use the prediction models in the previous section to estimate the profit an investor stand to gain investing in S&P 500 index over some trading periods (1 week, 1 month, 3 months, 6 months, 9 months and 1 year). We will compare the returns of our training model with the *buy and hold strategy*.

In the trading strategy, we will assume that:

- The number of days in the training periods 1 week, 1 month, 3 months, 6 months, 9 months and 1 year are 5 days, 20 days, 60 days, 120 days, 180 days and 240 days respectively.
- The investor has an initial investment capital of $\$100,000$.
- Trading is made using the closing price on the same day.
- Transaction cost is not considered in the computation.

The trading strategy to be implemented is as follows:

- When the output is 1 i.e. the model forecasts an increase in S&P 500, if the capital is in the form of cash, all the cash will be used in buying stocks. If the capital is in form of stocks, no trade will be performed.
- When the output is 0 i.e. the model forecasts a decrease in S&P 500, if the capital is in form of cash, no trade will be performed. If the capital is in form of stocks, all the stocks will be liquidated (sell all the stocks).
- At the end of the trading period, if the capital is in form of stocks, we will liquidate the stock using the closing price of the last trading day. Whatever we have in cash is the return after the trading period. The profit is the difference between the return and the initial investment.
- We will compare the profit from our trading strategy with the *buy and hold strategy*. In the *buy and hold strategy*, the investor simply uses the initial investment capital to buy stocks on the first trading day and liquidate all the stocks at the end of the trading period.

Using the trading strategy above, we will show a simple calculation for the 1 week trading period. Then we will show a graphical and tabular result of the simulation for all the trading periods. From Table 4.5, we will observe that the forecast of the three models for the 1 week period are the same. Therefore, we will only compute the profit from the Simple RNN trading strategy.

Table 4.5: 1 week forecast of the SimpleRNN, LSTM and GRU models.

Simple RNN	LSTM	GRU	Closing Price
1	1	1	200.4552
1	1	1	202.9807
0	0	0	201.1492
0	0	0	201.3902
1	1	1	201.1396

Simple RNN trading strategy:

Day 1: The forecast is 1 (the stock price will move up tomorrow). Therefore we will buy stocks using the closing price of today.

Initial capital $= \$100,000$

$$\text{Capital} = \frac{\$100,000}{\$200.4552}.$$

Therefore, our capital is in the form of stocks and cash. We have 498 stocks and $\$173.31$ cash at hand.

Day 2: The forecast is 1. Since the capital is in the form of stocks, we will perform no trade.

Day 3: The forecast is 0 (the stock price will move down tomorrow). Therefore, we will liquidate the stock.

$$\text{Capital} = (498 \times \$201.1492) + \$173.31 = \$100,345.61$$

Day 4: The forecast is 0. Since the capital is in the form of cash, we perform no trade.

Day 5: The forecast is 1.

$$\text{Capital} = \frac{\$100,345.61}{\$201.1396}.$$

Therefore, we have 498 stocks and $\$178.09$ at hand.

After the five trading days, we will have to liquidate the stock using the closing price of the last trading day.

$$\text{Capital} = (498 \times \$201.1396) + \$178.09 = \$100,345.61$$

$$\text{Profit} = \$100,345.61 - \$100,000 = \$345.61$$

Buy and Hold strategy:

The *buy and hold strategy* involves using all the initial investment to buy stocks on the first trading day. Then, at the end of the trading period, the stocks are liquidated and the profit is the difference between the initial investment capital and the capital at the end of the trading periods.

Day 1:

Initial capital $= \$100,000$

$$\text{Capital} = \frac{\$100,000}{\$200.4552}.$$

Therefore, we have 498 stocks and $\$173.31$ at hand.

After the five day trading period, we liquidate the stock

$$\text{Capital} = (498 \times \$201.1396) + \$173.31 = \$100,340.83$$

$$\text{Profit} = \$100,340.83 - \$100,000 = \$340.83$$

From the calculation above, we can see that using the proposed strategy based on the forecasts of the simple recurrent neural network model for 1 week yields a greater profit than the buy and hold strategy. Specifically, the profit exceeds that of the *buy and hold strategy* by $\$4.78$.

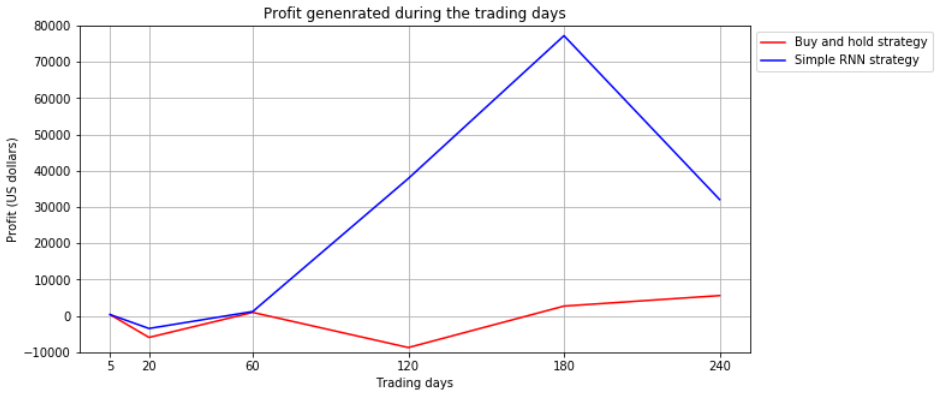


Figure 4.3: Profit of the trading strategies over some trading periods.

Table 4.6: Trading Profit of Simple RNN and Buy and hold strategies.

Trading periods	**SimpleRNN profit** ($)	**Buy and hold profit** ($)
1 week (5 days)	345.63	340.84
1 month (20 days)	-3,489.18	-5,952.63
3 months (60 days)	1,188.78	943.63
6 months (120 days)	37,873.18	-8,742.67
9 months (180 days)	77,224.58	2,685.85
1 year (240 days)	32,060.69	5,582.98

Figure 4.3 shows the simulation of the profit generated for the trading periods (1 week, 1 month, 3 months, 6 months, 9 months and 1 year), using the *buy and hold strategy* and the proposed strategy based on the Simple RNN forecast. We observe from the graph that the Simple RNN model strategy outperform the buy and hold strategy. The Simple RNN strategy only made a loss during the 1 month (20 days) trading period, while the *buy and hold strategy* made a loss for 1 month (20 days) and 6 months (120 days) trading periods. From Table 4.6, the loss made using the Simple RNN strategy was $\$3,489.18$ which is smaller than the loss of $\$5,952.63$ made using the *buy and hold strategy*. In

addition to this, we observe that the Simple RNN trading strategy loss margin is smaller compared to its profit margin. Also, while the average profit of the Simple RNN strategy over the six trading periods is $\$24,200.61$, that of the *buy and hold strategy* results in a loss of $\$857$. The peak profit of the Simple RNN strategy is $\$77,224.58$, which was generated in the 9 months trading period. This profit is approximately 77.2% of the initial investment capital. Similarly, the peak profit of the *buy and hold strategy* is $\$5,582.98$, which was generated in the 1 year trading period. This profit is approximately 5.6% of the initial investment capital.

5. Conclusion

In this essay, we discussed variants of recurrent neural network. We discussed the simple recurrent neural network whose hidden layers are made up of simple activation functions e.g. the sigmoid function. In addition to this, we considered recurrent neural networks with complex activation functions. The resulting neural networks are the long short term memory and gated recurrent unit networks. Since these networks may not take into account immediate shocks typical of financial market trend. We discussed the error correction neural network (ECNN). In applying these networks (Simple RNN, LSTM, GRU) to forecast the movement of S&P 500 index, we used the concept of experimental design to choose the features that are most appropriate for prediction. In each case of the three neural networks to be used for prediction, we performed 20 experiments to determine which of the experiments give the best accuracy score. For the three neural networks, the common features were the SPYt-1, JNJ, IXIC closing price. In particular, it was sufficient to use 75 hidden neurons and 40 epochs for the Simple RNN model, 25 neurons and 80 epochs for the LSTM model and 50 neurons and 40 epochs for the GRU model. The three selected experiments for these models were able to predict the movement of S&P 500 index with an accuracy of 75%, 74% and 74% respectively. In the experiment and result, we did not apply the error correction neural network. This is because the ECNN is not a standard recurrent network, it is a variant of the RNN used in SIEMENS AG for the financial forecast of their assets and the code for this network is a proprietary property of the company. However, writing a code for this and all other networks like the historical consistent neural network that cannot be found in machine learning libraries will constitute good research.

The Simple RNN gives us a better prediction when compared to other networks that were implemented. This is a deviation from the norm, since we expect better prediction from the LSTM network. This deviation is due to the fact that the financial dataset was not large enough. We will recall that the essence of a complex activation recurrent network is to capture the long term dependency. In the case of a short term dependency, the performance of a Simple RNN and any other complex activation network will not be too different. The question we must then ask is what constitutes a large dataset?

In addition to the accuracy of forecasts of these models, we have been able to show that using the proposed strategy with the Simple RNN forecast yields better profit than the buy and hold strategy. Specifically, we observe that the Simple RNN model strategy seldom makes loss and its profit margins far outweigh its loss. In the simulation of the profit of both strategies, we have not taken into account transaction or frictional cost (cost of buying and selling etc). Frictional cost will definitely influence the feasibility of using the Simple RNN strategy in practice. It is difficult to take into account this cost, since the cost varies from person to person. Nevertheless, future work can investigate this aspect.

Acknowledgements

My sincere gratitude goes to the African Institute of Mathematics (AIMS) for the post graduate scholarship that enabled me to undertake this masters program. Generally, I will like to appreciate the AIMS family for making my experience in South Africa worthwhile. I gladly admit that it was the best of times, and there is no doubt that this will form part of my most cherished moments.

The completion of this essay was made possible due to the academic guidance of Professor Ronnie Becker and Dr. Bubacarr Bah. I sincerely appreciate your time, patience, willingness to teach and principles of prompt response that eased the burden of this essay phase.

To my lovely Dad, I am dedicating this essay to you, in memory of the beautiful time you spent teaching me invaluable life principles. To the fantastic four ladies in my life; my adorable Mum, amiable Glory, voluble Rebecca and benign Precious, you are my jewels of inestimable value. I love you all.

Finally, I give glory to the Almighty God for his mercy, grace and strength in time of need.

References

A.M. Saxe, J.L. McClelland and S. Ganguli. *Exact Solutions to the Nonlinear Dynamics of Learning in Deep Linear Neural Networks*. Neural and Evolutionary Computing (cs.NE) 2014 (https://arxiv.org/pdf/1312.6120.pdf), Accessed April 2017.

K. Funahashi and Y. Nakamura. *Approximation of Dynamical Systems by Continuous Time Recurrent Neural Networks*. *Neural Networks*, 6:801–806, 1993.

H. Zimmermann, R. Neuer and R. Grothmann. *Modelling and Forecasting Financial Data*. Springer US, 2002.

S. Hochcreiter and J. Schmidhuber. *Long-Short Term Memory*. *Journal of Neural Computation*, 9: 1735–1780, 1997.

J. Chung, C. Gulcehre, K. Cho and Y. Bengio. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. CoRR abs/1412.3555, 2014.

M. Lukosevicius and H. Jaeger. *Reservoir Computing Approaches to Recurrent Neural Network Training*. *Computer Science Review*, 3:127–149, 2009.

M. Dixon, D. Klabjan and J.H. Bang. *Classification-based Financial Markets Prediction Using Deep Neural Networks*, 2016. Available from https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2756331.

M. Henaff, A. Szlam and Y. Lecun. *Recurrent Orthogonal Networks and Long-Memory Tasks*. Neural and Evolutionary Computing (cs.NE) 2016 (https://arxiv.org/abs/1602.06662.pdf), Accessed April 2017.

J. Martens and I. Sutskever. *Learning Recurrent Neural Networks with Hessian-Free Optimization*. In *ICML*, 2011.

S. Niaki and S. Hoseinzade. *Forecasting S&P 500 index Using Artificial Neural Networks and Design of Experiement*. *Journal of Industrial Engineering International*, 9:1, 2013.

R. Pascanu, T. Mikolov and Y. Bengio. *On the Difficulty of Training Recurrent Neural Networks*. Learning (cs.LG) 2012 (https://arxiv.org/abs/1211.5063.pdf), Accessed April, 2017.

S. Shen and H. Jiang. *Stock Market Forecasting Using Machine Learning Algorithm*. CS229 Machine Learning Autumn 2016 (http://cs229.stanford.edu/proj2012/ShenJiangZhang-StockMarketForecastingusingMachineLearningAlgorithms.pdf), Accessed April, 2017.