Faculty of Science and Engineering

Abdullah Ali

BSc (Hons) Mathematics

Neural Networks Applied to Financial Mathematics

May 2021

Department of Computing and Mathematics

Abstract

Neural networks are used in many areas such as weather forecasting, banking, business analytics and finance. We assess how neural networks are used in financial mathematics, more particularly on predicting stock prices and exchange rates. Traders and financial instituitions need to be able to predict the price of a stock in order gain a competitive edge against others in the market, its for this reason a good model is needed that can accurately predict exchange rates and prices of financial securities. In this report we applied an LSTM neural network to the US-EU exchnage rate as well as the Facebook stock price extracted from FRED data server and Yahoo Finance. Using MATLABs deep learning toolbox and the Tensorflow and Keras libraries in Python we obtained varying results for the accuracy of the predictions. The prediction accuracy was dependant on the number of iterations being used for each of the models as well as between the programming language used. Our model proved capable of predicting any stock price or exchange rate to be fed into it, however for practical usage it may not be profitable due to the RMSE not being small enough. Suggestions for future work were outlined in order to improve the models.

Plagiarism Declaration

With the exception of any statement to the contrary, all the material presented in this report is the result of my own efforts. In addition, no parts of this report are copied from other sources. I understand that any evidence of plagiarism and/or the use of unacknowledged third party materials will be dealt with as a serious matter.

Signed 

Contents

List of Figures

List of Tables

Listings

Chapter 1

Background and Introduction

1.1 History of Neural Networks

The idea of artificial neural networks came about in 1943 when neurophysiologist Warren McCulloch and mathematician Walter Pitts wanted to model how the neurons in our brains work. As such, they created a basic neural network (NN) with electrical circuits. In their book "A Logical Calculus of the Ideas Immanent in Nervous Activity" they discuss how artificial neurons could perform elementary logical tasks and proposed the first artificial neuron - the Threshold Logic Unit (TLU) (*Wikipedia- Artificial Neuron* 2020). Their model of a neuron had all the elements needed to compute any computable function and thus paved the way to the fundamentals of artificial neural networks.

The McCulloch Pitts Neuron is an extremely simple artificial neuron as it has no learning capability. It has an area say g which takes an input, performs calculations on those inputs and then another part say f which makes a decision and outputs a Boolean value (0 or 1) as shown in Figure 1.1. The inputs x_i are either 0 and 1 can either be excitatory or inhibitory. If an input is '1' and excitatory, the model added one. If the input is '1' and in-

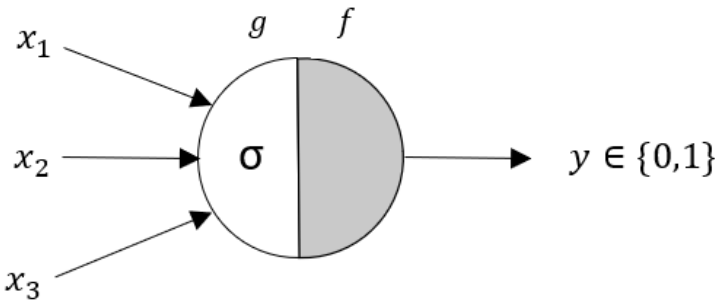

Figure 1.1: A McCulloch Pitts Neuron where x_i are inputs and y is the output. σ is the threshold which $g(x)$ (the sum) must exceed.

hibitory, it subtracted one from the sum. Furthermore, if the final sum is less than a certain threshold the output is '0'. If not, the output is '1'.

Figure 1.2: Warren McCulloch and Walter Pitts.

Warren McCulloch, born in November 1898, was a neurophysiologist from New Jersey who's known for the foundations of some brain theories and work in the cybernetics movement. Having had an interdisciplinary education, studying theology, psychology, philosophy and mathematical physics he did game-changing research in computational and biological computers as well as some research in neural structures and a few other disciplines. *The Cybernetics Thought Collective: A History of Science and Technology Portal Project* (2014)

Walter Pitts, born in April 1923 was a self-taught logician and mathematician who came from a poor family in Detroit, Michigan. He met McCulloch at the University of Chicago in 1942 and they started collaborating on neural networks which led to them publishing their notable paper. (*Walter Pitts* 2020).

In 1949 Donald Hebb, a Canadian psychologist took the idea of neurons further. In his book *The Organization of Behaviour* he proposed that neural pathways are strengthened every time they are used and that if two neurons fire at the same time their connection is deepened. (*A Concise History of Neural Networks* 2016). His 'Hebbian Learning' model (based on neural plasticity) is a major precursor to neural networks as it allows neurons to learn by updating weights between neurons in a network. The idea, which was inspired by the animals neural weight adjustment process has three major points:

- Data is stored between neurons as 'weights'.
- The weight change between neurons is directly proportional to the product of the input values.

$$\Delta w \propto x.y \implies \Delta w = \beta.x.y. \tag{1.1}$$

- As learning occurs, repeated activation of weakly linked neurons progressively alters the pattern and the strength of the weights, which leads to stronger connections.

As computers started to bloom in the 1950's Nathaniel Rochester from IBM research laboratories tried to simulate a neural network. He failed on the first attempt, but later attempts were successful. Other researchers were also working on translating the existing networks onto computational machines and as a result, the first Hebbian network was effectively implemented in 1954 at MIT.

In 1958 the idea of a Perceptron was proposed by psychologist Frank Rosenblatt. Inspired by the operation of the eye of a fly, his research led him to introduce the Mark I Perceptron which was built in hardware and is the oldest neural network still used today (*Brief History of Neural Networks* 2019) The perceptron was based of a modified version of the McCulloch-Pitts Neuron. Using Hebbs findings, it took in weighted inputs and summed it together outputing a binary value. It also took an additional constant input (denoted as b in the figure below), this is known as the bias.

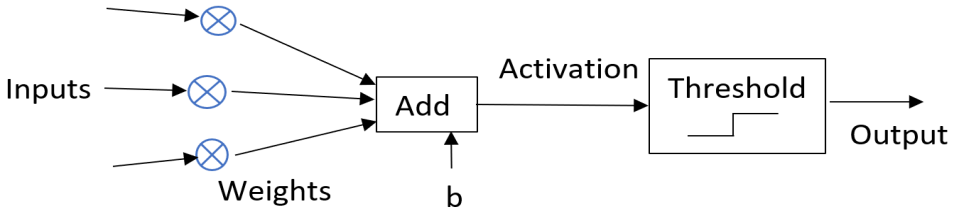

Figure 1.3: A simple artificial neuron.

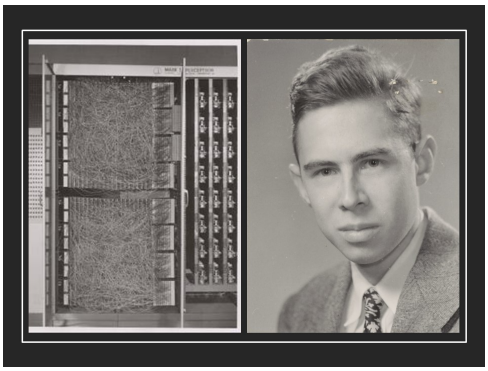

Figure 1.4: Frank Rosenblatt and the Mark 1 Perceptron (*Frank Rosenblatt* n.d.).

Frank Rosenblatt was an American psychologist born in New Rochelle 1928. He went to Cornell University where he acquired his A.B in 1950 and his Ph.D. in 1956 (*Frank Rosenblatt* n.d.). He had a wide research interests one of which were models of brain function. In 1958 he described the Perceptron which was built based on biological principles and had an ability to learn. He developed his approach and published many papers and also a book *Principles of Neurodynamics*. In 1966 he was performing experiments on the transfer of learned behaviour using rat brain extracts and published comprehensively in this area. Rosenblatt also had an interest in politics and astron-

omy.

During the late 1950's research in neural networks began to move quickly. Bernard Widrow and Marcian Hoff constructed models called "ADALINE" and "MADALINE". They were named after their use of Multiple ADAptive LINear Elements. MADALINE was the first neural network that was successfully applied to a real-world problem. It used an adaptive filter to remove echoes on phone lines and albeit old, is still in use today (*Brief History of Neural Networks* 2019).

The hype around neural networks came to an end in 1969 when Minsky and Papert published their book "Perceptrons". The book came to a conclusion that Rosenblatt's single perceptron viewpoint couldn't be translated productively into multi-layered networks. The book had only one proven result - linear functions are unable to model non-linear ones. This had a clear effect on research funding as well as the community and the age referred to as the 'AI winter' started, where progress on neural networks was halted for 10-12 years.

Neural networks re-emerged in the 1980's when Jon Hopfield presented a paper to the national Academy of Sciences on what came to be known as the Hopfield Net. Japan also announced their fifth generation effort on Neural Networks at the US-Japan conference of Neural Networks, which led the US worried about being left behind. This got the funding to flow again (*A Concise History of Neural Networks* 2016).

Figure 1.5: From left, D.E. Rumelhart, G.E. Hinton and R.J. Williams (1986): *Learning Representations by Back-propagating Errors.*

The concept Hopfield discussed was actually already in existence since the 60's and one that had been constantly worked on in the AI winter. The method was backpropagation - which assigned reducing significance to each event that were farther back in the course of events. Paul Webros and Freud wrote a PhD thesis showing the importance of backpropagation for neural nets. However their work didn't gain much traction, until being re-re-discovered by Rumelhart, Hinton and Williams where they republished an explicit and comprehensible framework on the technique D. Rumelhart (1986) which led to it taking off in the community.

Backpropagation and gradient descent formed the foundations of neural networks. Gradient descent repeatedly updates the weigths and bias towards the minimum of the cost function whereas backpropagation assesses the gradient of the cost function with respect to the biases and weights (*A Concise History of Neural Networks* 2016).

In 1987 the Institute of Electrical and Electronic Engineers (IEEE) held their first International Conference on Neural Networks which amassed 1800 attendees.

Around the 1990's a lot of varieties of Neural Networks started popping up, such as recurrent neural nets (RNNs), long short term-memory recurrent neural networks (LSTMs) and Boltzmann machines. The convolutional neural network (used to analyze images) was also invented by Yann LeCun in 1989 (LeCun et al. 1989) and his bank check recognition system that he helped develop was widely used by U.S banks.

However, there were difficulties. As people started developing really deep networks, backpropagation stopped working. The reason was because backpropagation relies on obtaining the error at the output layer and splitting the cause of that error to prior layers. Training deep networks like this was too hard an optimization problem and were seen as a hassle to work with. In 1997, Schmidhuber and Hochreiter proposed LSTM (Long Short Term Memory), a recurrent neural network framework that helped solve the problem of training RNNs (Schmidhuber 1997). However, this did little to fix the problem that neural nets were unreliable when dealing with deep architectures. This led to interest shifting to new machine learning algorithms such as support vector machines and random forests, whereby a new winter began for neural nets.

Three researchers, Hinton, Bengio and LeCun kept neural networks alive during this winter. Hinton secured funding from the Canadian Institute for Advanced Research (CIFAR) and kept working on Neural Networks. Hinton and a small group of researchers tried to 'rebrand' neural networks as 'deep learning' and in 2006 they published a breakthrough paper - A fast learning algorithm for deep belief nets (Hinton et al. 2006) which rekindled interest in neural networks.

The idea was that neural networks with a lot of layers can indeed be trained well if the correct activation functions are used and if the weights are initialized in a smart way. Each layer of the neural network can be trained by unsupervised training (which starts of the weights better than just assigning them random values) and then finishing off with a round of supervised learning.

There were a few main reasons why researchers hadn't been able to train deep neural networks

before this time:

- The weights were not initialized in an optimal way.
- Incorrect activation functions were used.
- Training data sets were too small.
- Computing power was not enough.

Researchers figured out that the sigmoid activation functions caused problems when applying backpropagation to deep neural networks i.e the gradient of the loss function approached zero as more layers of the activation function were added. (*The Vanishing Gradient Problem* 2020). A simple solution was to use ReLU which improved the problems and also required less computing power.

As data started becoming more available during the 2000s when the internet started becoming more prominent researchers realised they had a lot of data (images in particular) which could be used to train these networks. GPUs also started becoming widely available which helped in computing power required for training deeper networks.

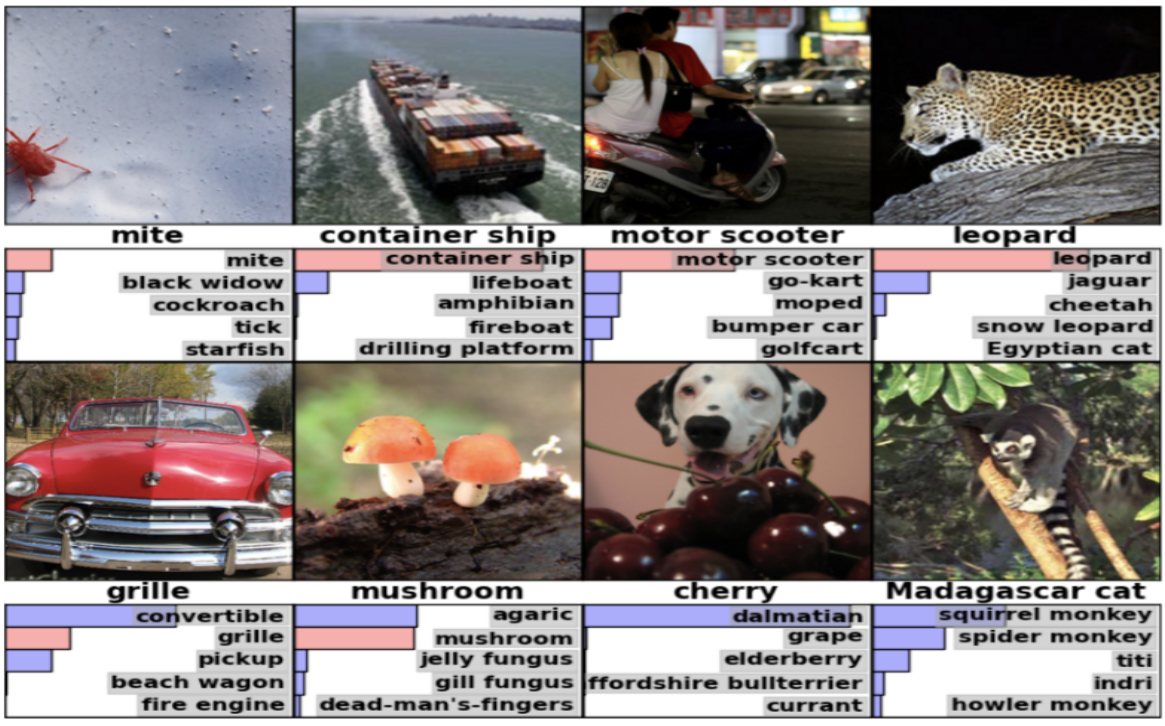

Figure 1.6: ImageNet challenge (Krizhevsky et al. 2012).

In 2010 the *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)* (2020) was set up.

This allowed researchers to compete in evaluating the accuracy of their image classification algorithms and object detection systems. The ImageNet dataset contains over 15 million images of objects spanning 22 categories from mammals to food and sport.

In 2012 Alex Krizhevsky, Geoffrey Hinton, and Ilya Sutskever developed AlexNet (Krizhevsky et al. 2012) which was the first convolutional neural network to enter the ILSVRC. AlexNet won the ILSVRC-2012 challenge with an error rate of 15.3% compared to the second place error rate of 26.2%.

People realized neural nets were a powerful tool for image classification. AlexNets domination in the 2012 challenge cemented the usefulness of neural networks in the AI community and people accepted it as state-of-the-art.

1.2 History of Financial Math

Financial mathematics was first proposed by French Mathematician Louis Bachelier. His scholarly work on mathematical finance 'Theory of Speculation' (Bachelier 2011) was published in 1900 in which he introduced a mathematical model of Brownian motion applied to valuing stock market options. In his thesis he introduced concepts of what is now known as stochastic analysis. He used this to valuate stock and option markets.

Bachelier had derived the price of an option where the share price fluctuation is modelled by a Wiener process and derived the price of what is now called a barrier option - the option that depends on whether the share price has crossed a barrier (*Barrier option* 2020).

Figure 1.7: Louis Bachelier (*Louis Bachelier* 2020).

Louis Bachelier born in Le Havre March 1870, was a French Mathematician who's known as the father of financial mathematics. After secondary school, he lost both his parents and had to get involved with family business. It is here where he was introduced to financial markets. He defended his thesis Théorie de la Spéculation at age 30 before Paul Appell. Joseph Boussinesq and Henri Poincaré, with the favourable report being written by Poincaré - one of the most distinguished mathematicians at the time. (*MacTutor - Louis Bachelier* 2020). His future work wasn't taken with much

importance by the French mathematical élite but in fact he was ahead of his time as evidenced by how much financial mathematics is used in today's international derivative exchange.

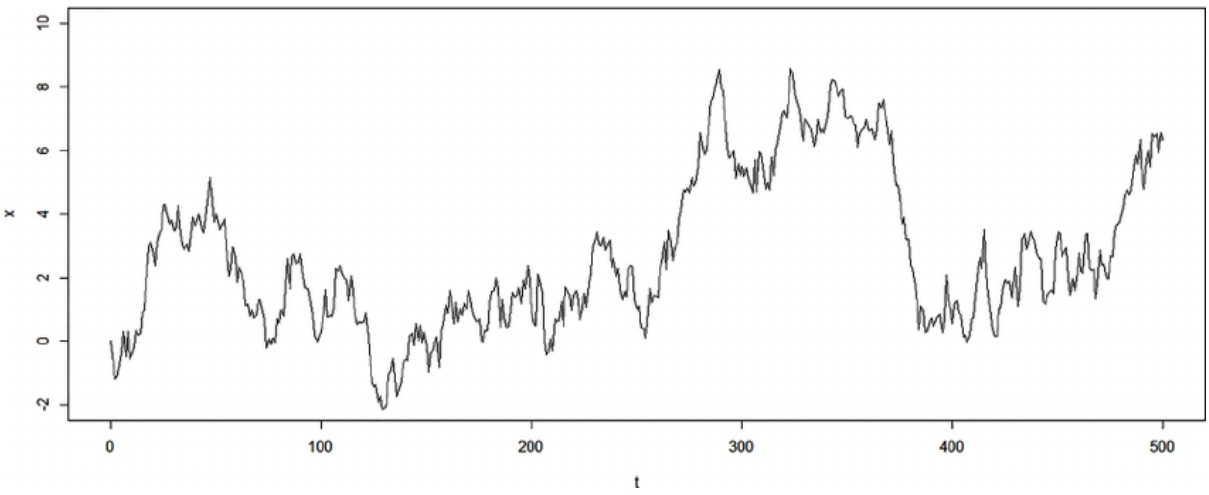

Figure 1.8: Brownian Motion Process (Özel Kadilar 2015).

We can see the Brownian Motion Process and the GBP/USD historical chart looks very similar.

Figure 1.9: (*Pound Dollar Exchange Rate (GBP USD) - Historical Chart* 2020).

Modern financial math started in 1960s when American economist Paul Samuelson published 2 papers that argued how stock market price movement is random. One paper along with Famas paper **fama** formed the base of what is known as the efficient market hypothesis - that in an efficient and sophisticated capital market, asset price fluctuation is described by a model where the best possible estimate for an assets price in the future is its current price. (R. Jarrow and Protter 2004) However, under their hypothesis using past price data or publicly available

information on assets to predict future security price will not succeed.

In the other paper, he worked with Henry McKean to show that stock price movement is modelled well by Geometric Brownian Motion. Samuelson explained that Bacheliers model failed to assume that stock prices would always stay positive, whereas Geometric Brownian Motion avoids this issue. (R. Jarrow and Protter 2004)

The most important breakthrough was the Black-Scholes model for option pricing. In 1973 Fischer Black and Myron Scholes published *The pricing of options and corporate liabilities* (Black and Scholes 1973). This paper, along with Robert Merton's 1974 paper derived an equation that led to the first purely mathematical call and put option pricing model. The Black-Scholes formula is shown below:

$$\boldsymbol{C = N(d_1)S_t - N(d_2)Ke^{-rt}}. \tag{1.2}$$

$$\text{where } d_1 = \frac{ln\frac{S_t}{K} + (r + \frac{\sigma^2}{2})t}{\sigma\sqrt{t}}. \tag{1.3}$$

$$\text{and } d_2 = d_1 - \sigma\sqrt{t}. \tag{1.4}$$

$\boldsymbol{C}$ = Call option price.

$\boldsymbol{N}$ = CDF of the normal distribution.

$\boldsymbol{S_t}$ = Spot price of an asset.

$\boldsymbol{K}$ = Strike price.

$\boldsymbol{r}$ = Risk-free interest rate.

$\boldsymbol{t}$= Time to maturity.

$\boldsymbol{\sigma}$ = Volatility of the asset.

The key variable in the Black-Scholes model is the volatility of the asset. These equations normalized the use of mathematics in the pricing of derivatives. Their method opened the way for economic measurements in many areas and formed new financial instruments. It also enabled better risk management in society. The Royal Swedish Academy of Sciences awarded Scholes and Merton along with the late Fischer Black the 1997 Nobel Prize in Economics "for a new method to determine the value of derivatives." (*Prize in Economic Sciences 1997* n.d.).

In April 1973, the Chicago Board Options Exchange (CBOE) began trading options publicly, just one month before the publication of the Black-Scholes model. By 1975 the CBOE were

using the model to price and hedge their options positions. Hedging of an option is the fundamental insight underlying the Black-Scholes model. The idea is that a dynamic portfolio trading approach in the stock can replicate returns from an option on that stock.

Figure 1.10: From left to right, F.Black, M.Scholes and R.C.Merton (*Black-Scholes/Merton 40* 2020).

Fischer Black, born in 1938 was a Harvard graduate. He received his Ph.D. in applied mathematics in 1964 and got into monetary policy during the 1970s (*Fischer Black* 2020). He died in 1995, two years before his method for valuing financial derivatives would have won a Nobel Prize. **Myron Scholes**, was born in Ontario, Canada in 1941. He earned his MBA at the Booth School of Business in 1964 and his Ph.D. in 1969. He wrote his dissertation under the Supervision of Eugene Fama and Merton Miller - researchers who were involved in developing the new field of financial economics. **Robert Merton** is an American economist, Nobel Prize laureate and professor at the MIT Sloan School of Management. He earned his doctorate in economics at MIT in 1970. He is known for his contributions to continuous-time finance the most notable one being the Black-Scholes-Merton model.

In the 1980s, two researchers Harrison and Pliska introduced the idea of martingales into mathematical finance when they published *"Martingales and Stochastic integrals in the theory of continuous trading"* (Harrison and Pliska 1981). They showed that a more intellectual formulation of the Black-Scholes solution as a mathematical model called a martingale provides better generality. A martingale is a sequence of random variables for which the expected value for the next value in the sequence is equal to the present value.

In 1960's adoption of financial technology wasn't widespread as the volatility of the US markets was low, stock market rose steadily and exchange rates were fixed. However, during the 1970's a few events caused changes to the market and increased volatility. New derivative-security exchanges traded options on stocks, futures on currencies and futures on US bills and bonds. This led to increasing demand for financial models. The Black-Scholes model was particularly

suited for these new fast-paced exchange listed options markets and large trading volumes (Dunbar n.d.).

By 1980s the use of finance theory models in practice became nearly immediate. Also, the mathematical models became as advanced as the ones in academic research. The financial deregulation of markets accelerated the adoption of sophisticated mathematical financial models.

New breakthroughs in this area wasn't as important as the discoveries in the 1960s and 1970s however, there was a greater amount of resources devoted to the development of mathematical models in finance. In addition, with better computer technology being developed, new financial markets emerged and existing ones expanded. The numerical solutions for complex models were also made possible by the faster computer speed and larger memory. Also the speed to calculate solutions to existing models was reduced which allowed for real-time calculations of price and hedge ratios.

1.3 History of Neural Network uses in Finance

The first person to use neural networks for finance was White (1988) who used feed-forward neural networks (FFNN) for market forecasting. He was curious as to whether neural nets can detect non-linear regularities from economic time series and hence decode regularities in asset price movement. He used daily IBM stock prices as an example. However found that his training results were too optimistic - a result of over-fitting of the model. He concluded that "the present neural network is not a money machine".

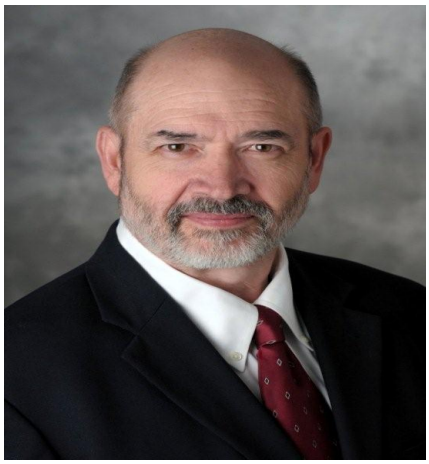

Figure 1.11: *Halbert White* (2012).

In 1996, Chiang et al used a backpropagation neural network to forecast the end of year net asset value for mutual funds (Chiang et al. 1996). A 3-layer network was used with 15 neurons in the input layer, 20 neurons in the hidden layer and 1 in the output. They trained their model with historical economic data and found that neural networks performed 40% better than linear regression models and 60% better than non-linear regression models. This shows neural networks achieved significantly better results than regression models in cases of limited data availability.

Kim S.H. (1998) used a refined probabilistic neural network (PNN), called an arrayed probabilistic network (APN) to predict stock market prices. They circumvented the problem of a bipolar output (an output that only yields Yes or No; Up or Down) by used a graded forecast of multiple discrete values. A "mistake chart" which benchmarks against a constant prediction was used to compare the accuracy of feed-forward neural networks with back-propagation (BP) models against APN, PNN, a recurrent neural network (RNN) and case based reasoning. They came to a conclusion that APN's outperformed recurrent and BP methods but the case based reasoning tended to perform better than all networks.

In 1999, Garliauskas developed a NN computational algorithm linked with the kernel function approach and recursive prediction error method and applied it to stock market time series forecasting. The main reason for the use of the kernel function with the NN model was that the function activates changes to the weights in order to attain convergence of the target and output functions. He concluded that NNs for financial time series forecasting is superior to classic statistical and other methods (Garliauskas 1999).

Anders et al. (1996) investigated the use of NNs in pricing call options on the German stock index DAX. They obtained insights into the pricing process of the options market by testing for the explanatory power of a number of NN inputs. They used a multi-layer perceptron with four input variables and 3 hidden units for their model to price options. The results indicated that statistical strategies led to 'parsimonious' NNs with better out-of-sample performance when compared the the Black-Scholes model. The estimated networks showed a higher pricing accuracy according to the performance measures R^2, MAE, RMSE and MAPE than the theoretical Black-Scholes model.

Odom and Sharda applied NNs to bankruptcy prediction in 1990 in their paper "A Neural Network Model for Bankruptcy Prediction" (Odom and Sharda 1990). They compared their neural network model against a multivariate discriminant analysis (MDA) and found that the neural networks had at least the same level of accuracy as discriminant analysis. They state that the neural network model has significant advantages over other methods and can analyze complex plans better than statistical-based ones. It also had no need for restrictive statistical assumptions and hence provided higher levels of accuracy.

Study	Area	Statistical Model	Performance:Statistical Model vs NN	Conclusion
Chiang et al. (1996)	Mutual fund net asset value forecasting	Regression	15.17% vs 8.76& mean-absolute-percent forecasting error	NNs outperform both linear and non-linear regressions
Odom and Sharda (1990)	Bankruptcy prediction	Discriminant analysis	59.26% vs 81.48% prediction accuracy	NNs perform better
Bennell and Sutcliffe (2004)	Pricing options	Black-Scholes	A mean standard deviation of 872 vs 321.9	The MLPs performed better on the out-of-money options but worse on the in-money options
Yoon et al. (1993)	Predicting stock price performance	Multiple discriminant analysis (MDA)	75% vs 91% prediction accuracy on training data. 65% vs. 77% prediction accuracy on testing data	NNs outperform MDA
Yao and Tan (2000)	Forecasting foreign exchange rate	Regression (ARIMA)	A best return of 6.94% vs 32.36% in profits	NN models perform better than the ARIMA model

Table 1.1: Many comparative studies show how neural networks outperform typical statistical econometric models. The name(s) of the author(s) and the research year are listed in column 1, the application area is listed in column 2, and the statistical model for which neural networks were compared is listed in column 3. Column 4 shows the performance of the statistical model vs NN as detailed in the study and column 5 gives a conclusion.

1.4 Introduction

With the ever increasing amount of data being gathered in the world, it is important to make use of the data in clever and sophisticated ways. This is where neural networks come into play. Nowadays, artificial neural networks are being used in different fields like weather forecasting, image classification, economy as well as in business and industry where trillions of bytes of data are being gathered.

Neural networks allows for utilizing data in a useful way and make accurate predictions as it 'learns' the same way a human brain does. This makes neural networks a very robust tool in mathematical finance. They are used in areas of semi and non-parametric regression as well as pattern classification, such as time series prediction, credit scoring and risk estimation. NNs allows one to use data and let the data decide the parameters of a model through a 'learning' process in order to give the best predictions to a particular problem.

Traditional time series methods and other statistical methodologies struggle with non-linear (noisy) relationships and high-dimensional multivariate problems such as in financial markets. These problems are much more easily resolved with deep neural networks. Deep neural networks

are particularly useful for solving complex problems as they have a lot of layers that can be trained. Each node in a layer will have a particular representation of the data. For example, for time series data, the input layer may be comparable to a matrix or vector of numerical values.

$$\begin{bmatrix} w_1 & w_2 & w_3 \\ w_1 & w_2 & w_3 \\ w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix} = \begin{bmatrix} w_1x_1 + w_2x_2 + w_3x_3 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + b \end{bmatrix} \xrightarrow{\text{activation function}} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

The input vector consists of x_1, x_2 and x_3 which are numeric values which could represent certain features. This vector is multiplied by the weights w_1, w_2 and w_3 and then a bias b is added to it. The activation function performs an operation on the resulting calculations which can help the network learn complex patterns in the data.

For categorical data however, "one-hot encoding" could be used. This is where a binary vector is created with its size equal to the number of categories. All the vector values are 0 with the exception of the vector position corresponding to the category, which takes on a value of 1 instead (Koenecke and Gajewar 2020). An example of this is shown below:

Colour
Red
Red
Green
Yellow
Green

$\Rightarrow$

Red	Green	Yellow
1	0	0
1	0	0
0	1	0
0	0	1
0	1	0

This kind of encoding to train the neural network is useful for data that is very categorical. It essentially converts categorical data into integer data ready for the computer to read. Powerful predictions can be made using neural networks in this way.

With that said, a disadvantage of using neural networks is it's 'black box' nature which can be problematic for statistical interpretability. It also requires a lot of data - thousands if not millions of labeled samples are needed to effectively train a neural network which is not an easy problem to deal with.

There are a few general equations for describing a neural network.

Feedforward: $\text{Output}_n = \text{Activation}(\text{Weights}_{n-1} * \text{Output}_{n-1})$.

Back-propagation: $E_{n-1} = W_n^T * E_n$.

Updating the weights: $W_n = W_n - Lr * \frac{\delta E_n + 1}{\delta W_n}$.

The notation will be gone into more detail in Chapter 2.

There are several learning rules that neural networks use to improve the their performance and apply it to the whole network. Learning rules update the weights and bias levels of a network as the network takes in data. A few learning rules are shown below:

Hebbian learning Rule: $w_{ij} = x_i * x_j$.

Perceptron Learning Rule: $\sum_i \sum_j (E_{ij} - O_{ij})^2$.

Delta Learning Rule: $\Delta w = \eta(t - y)x_i$.

This report will discuss what neural networks are, as well as the types of neural networks for example Feed-Forward Neural Networks (FFNN), Convolutional NNs (CNN) and Recurrent NNs (RNN). Furthermore, we'll introduce a use case for each of the neural network types discussed. We will also delve into financial mathematics and the various areas of this field starting with a brief explanation of financial assets and then discussing time series and how we can use neural networks for time series prediction.

Financial data can be highly non-linear and sometimes with stock price data, completely random. We can see in Figure 1.12 how random stock price movement can be. In noisy and

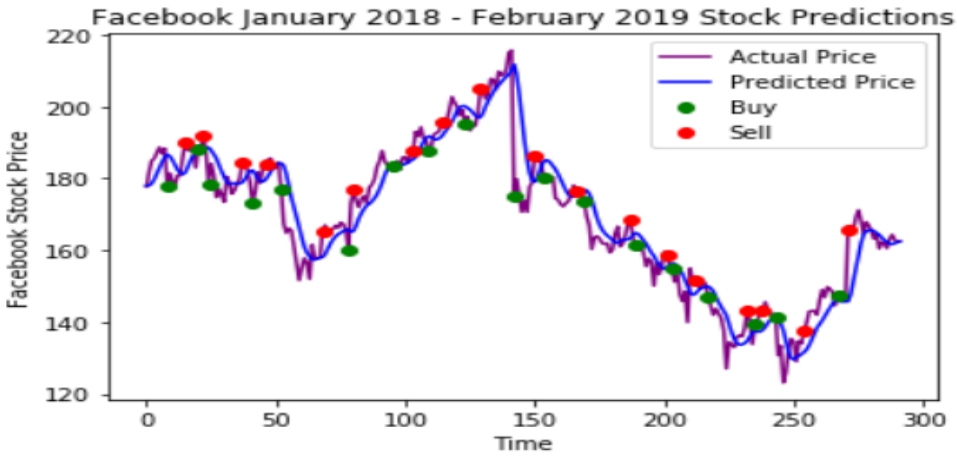

Figure 1.12: Actual price vs predicted price for Facebook stock. The predicted price was obtained using a LSTM neural network. (*Using AI to Predict Facebook's Stock Price* 2019).

non-linear data, such as these financial markets, it is increasingly important to branch away from traditional statistical methodologies as they do not work as well as NNs. In chapter 4, we consider how neural networks are applied to various areas of financial mathematics such as in pricing financial derivatives, stock market prediction, bankruptcy prediction and algorithmic trading.

The report will come to a conclusion with a methodology for predicting US/EU exchange rate as well as stock prices. We will use a Long-Short-Term-Memory (LSTM) neural network in a MATLAB and Python program and train the model using stock market data obtained from Yahoo Finance. The results of each prediction will be evaluated and compared with each of the neural network methods used. A conclusion will be made and suggestions for future research outlined.

Chapter 2

Neural Networks

2.1 Non-Linear Dynamical Systems

In mathematics, a non-linear system is one where the change of output does not correlate with the change of input. These systems may look disordered, unpredictable and contrasting when compared to linear systems. An example of this could be stock market prices, where fluctuations are common and the prices of each stock don't follow a linear pattern. This is where neural networks (NNs) come into the picture. NNs provide the ideal means of modelling sophisticated non-linear systems as it has the ability to learn advanced non-linear relationships and make predictions.

2.2 What are Neural Networks?

Neural networks are a mathematical system or algorithm that attempts to discover patterns in a set of data in a way similar to the human brain. Our brains comprise of billions of neurons that communicate with each other using electrical signals. These signals arriving at the neurons are summed in a certain way and must exceed some threshold which will then trigger an impulse in the next neuron (Sardi et al. 2017). The artificial equivalent of a neuron are often called a unit or node which is shown in Figure 1.3

Combining many units of neurons into several layers makes a neural network. Neurons of one layer interacts with neurons of the next layer through weighted connections. If the output of one node exceeds a specified threshold value it sends a signal to the next layer of the network,

otherwise no data is sent to the next layer. The data that's passed through undergoes a computation by the activation function which determines the output in a range of 0 to 1.

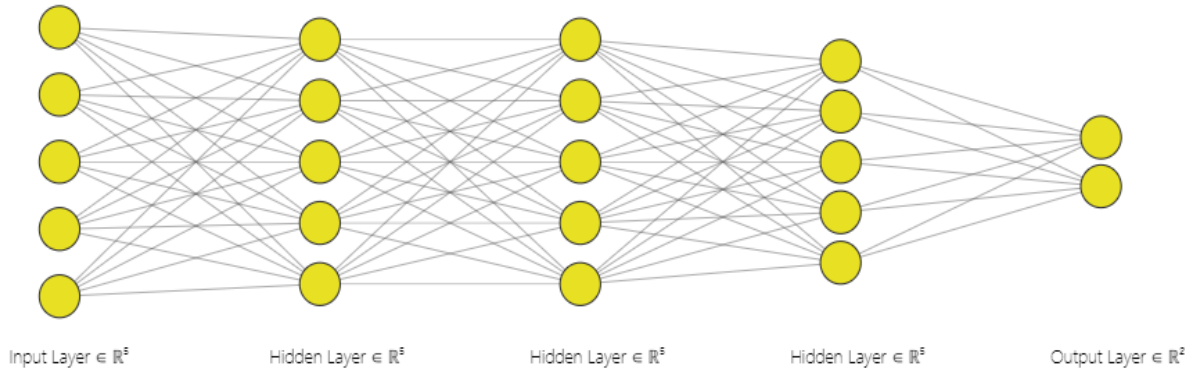

Figure 2.1: A deep neural network

Neural networks rely on vast amounts of data to train and improve their accuracy over time. Once they do, they are a very strong tool in computer science and artificial intelligence as they allow us to classify high-velocity data as well as make predictions with the data.

Each individual node of a neural network can be thought of as a small linear regression model - consisting of an input, weights, a bias and an output.

$$\sum_{i=1}^{n} w_i x_i + b = w_1 x_1 + w_2 x_2 + w_3 x_3 + b \tag{2.1}$$

where b is the bias

$$output = f(x) = \begin{cases} 1, & \text{if } \sum w_1 x_1 + b >= 0 \\ 0, & \text{if } \sum w_1 x_1 + b < 0 \end{cases} \tag{2.2}$$

When the inputs are determined the weights are assigned. These weights help to gauge the importance of any particular variable, with the larger ones having a greater change on the output.

If we look at the output as a simple binary value for example Yes or No (1 and 0), we can see that the sum of the inputs and weights plus the bias has to be above a certain threshold (in this case above 0) in order to activate or 'fire' the node and output a 1 so that data can be passed to the next layer. The activation function $f(x)$ shown in 2.2 is the Heaviside (or binary) step function. It is the most simplest of activation functions and used mostly in primitive NNs.

2.3 Activation Functions

Activation functions add non-linearity to a NN system. They are applied to the sum of products of inputs and their weights to get the output of the present layer so that it (the output) can be passed as inputs to the subsequent layer. The activation function will decide whether to 'fire' the neuron based on whether the inputs passed a certain threshold.

In neural networks, we update the weights and bias according to the error of the outputs (also known as back-propagation). The activation functions allow for back-propagation as the gradients are provided with the error to update the biases and weights.

Let's look at the binary step function below, which corresponds to equation 2.2 but in graphical form.

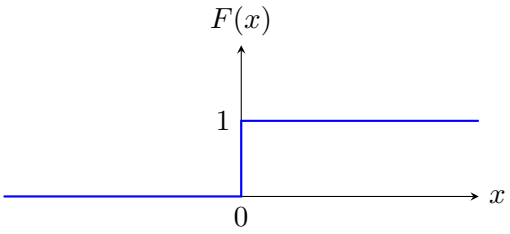

Figure 2.2: Binary step function

This simple activation function can be used in a very limited number of cases for example a binary classifier. It can't be used in a multi-variable neural network. Also the gradient of the function is zero which could cause a problem during the back-propagation step, as the derivative is 0.

Now let us take a look at the linear activation function A linear activation function is similar to the equation of a straight line i.e $A = cx$. It is better than the binary step function as it does not output binary values, instead the output is not bounded by any range. The linear activation

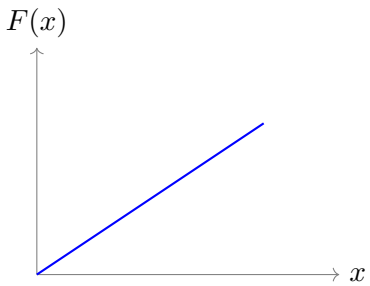

Figure 2.3: Linear function

function however, is still not very useful as the derivative would equal to the constant used; the derivative of $A = cx$ w.r.t x is c. This would mean that the weights and biases won't be updated as the gradient would stay constant - there is no relation between the gradient and x.

There are many different types of activation functions that are used in neural networks. Non-linear activation functions are used in the average neural network which has complex data. Below are some non-linear activation functions.

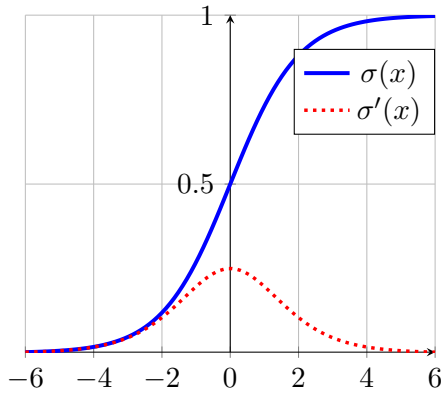

Figure 2.4: Sigmoid function

The Sigmoid Function

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2.3}$$

A smooth S-shaped function. It always gives outputs in a range (0,1) The function is also differentiable which means we can find the gradient of the curve at any two points.

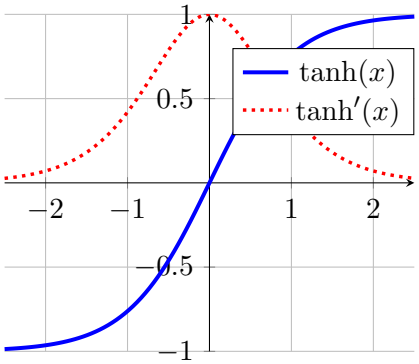

Figure 2.5: Tanh function

The Tanh Function

$$f(x) = tanh(x) = \frac{1}{1 + e^{-2x}} - 1. \tag{2.4}$$

Like the sigmoid function it is also S-shaped, however its gradient is steeper. The output values are in a range (-1,1). Mostly used in hidden layer of NNs as it is zero-centered, making learning easier for the next layer.

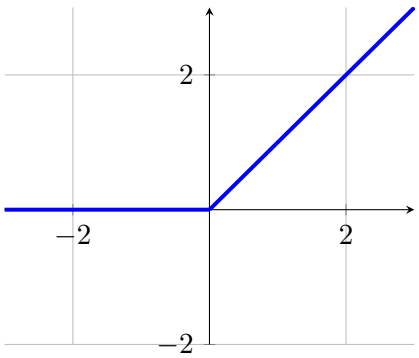

Figure 2.6: ReLu function

The ReLu Function

$$f(x) = max(0, x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}. \tag{2.5}$$

The output is in a range (0,inf). The inputs which are negative are ignored and the positive inputs are treated linearly. This filter like mechanism allows the network to converge very easily.

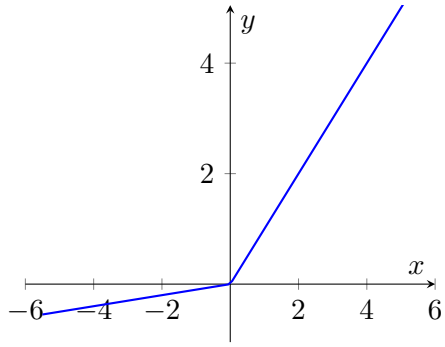

Figure 2.7: Leaky ReLu function

The Leaky ReLu Function

$$f(x) = max(0, x) = \begin{cases} 0.01x, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases}. \tag{2.6}$$

This function allows for small negative values when the input is less than 0.

Softmax function

This function is a combination of Sigmoid functions. It is normally used in the output layer of a NN.

$$S(z_i) = \frac{e_j^z}{\sum_{k=1}^{K} e_j^z} \tag{2.7}$$

Unlike the Sigmoid function, which is used for binary classification, this activation function can be used for multiclass classification.

2.4 The Perceptron

The perceptron is the basis of all neural networks. It is an element that weights and sums up the inputs and compares it with a threshold value σ which it must exceed to produce a binary output as shown in equation 2.2.

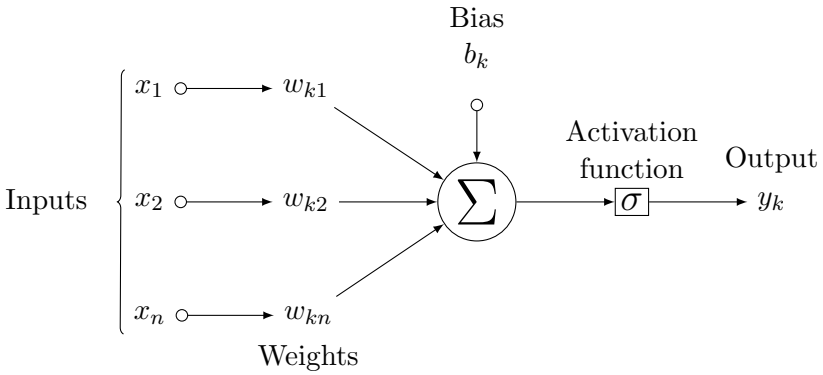

Figure 2.8: Detailed view of a perceptron - weights are denoted as $\mathbf{w}_k = \mathrm{w}_{kj}$ for $j = 1, 2, ..., n$ and y_k is the output of neuron k.

As we can see above, the inputs are fed in and multiplied with a particular weight. All the products are summed together with a bias, an activation function performs a computation and

an output is produced. This is shown in equation 2.1. Typical activation functions used are the identity function 2.3, sigmoid function 2.5 and hyperbolic tangent function 2.4.

In general a perceptron with activation function $\sigma(x)$ has output

$$y = \sigma(\overrightarrow{w} \cdot \overrightarrow{x} + b). \tag{2.8}$$

A perceptron will learn to accurately classify a set of input-output pairs $(\overrightarrow{x}, y)$ by adjusting the weights and bias.

Error function

The learning process for a perceptron requires an error function E. This is the difference of the target value t (the true value we want to reach) and the output value y computed by the perceptron for a set of input-output pairs $(\overrightarrow{x}, y)$. This error function is usually the mean-squared-error (MSE) which is established for a set of N input-output pairs $X = \{(\overrightarrow{x_1}, y_1), (\overrightarrow{x_2}, y_2), ..., (\overrightarrow{x_N}, y_N)\}$ as,

$$E(X) = \frac{1}{2N} \sum_{i=1}^{N} (y_i - t_i)^2. \tag{2.9}$$

where y_i represents the output for certain input $\overrightarrow{x_i}$. The objective is to minimise $E(X)$ as best as possible, which can be done by updating $\overrightarrow{w}$ and b so that $E(X)$ is as close to 0 as possible. **Note:** MSE is usually used for regression problems. For classification problems the cross-entropy error function is usually used.

Delta Rule

The error $E(X)$ can be minimized using gradient descent. This is an iterative process and it reduces the value of the error function till it converges to a local minimum. The values of $\overrightarrow{w}$ and b are randomly initialized and updated with gradient descent. Lets take $\overrightarrow{w_0}$ and b_0 as the initial random values, then gradient descent updates these values according to the equations:

$$\overrightarrow{w}_{i+1} = \overrightarrow{w_i} - \eta \frac{\partial E(X)}{\partial \overrightarrow{w_i}}. \tag{2.10}$$

$$b_{i+1} = b_i - \eta \frac{\partial E(X)}{\partial b_i}. \tag{2.11}$$

where the values of $\overrightarrow{w}$ and b are $\overrightarrow{w_i}$ and b_i after the ith iteration of gradient descent. η is the learning rate which decides the step size of gradient descent. If value of η is too small the learning will will be slow and it would take too long to converge, whereas if η is too large the learning will be suboptimal as it will fail to converge.

The weight delta (difference) $\Delta\overrightarrow{w} = \overrightarrow{w_{i+1}} - \overrightarrow{w_i}$ and bias delta $\Delta b = b_{i+1} - b_i$ are calculated using the delta rule, which can be derived by application of the chain rule and power rule for the calculation of the 2 partial derivatives $\frac{\partial E(X)}{\partial \overrightarrow{w_i}}$ and $\frac{\partial E(X)}{\partial b_i}$. This rule is the single-layer version of the backpropagation method. For a perceptron with activation function f and the mean squared error function the delta rules are given by,

$$\Delta\overrightarrow{w} = \frac{1}{N}\sum_{i=1}^{N}\eta(t_i - y_i)\sigma'(h_i)\overrightarrow{x_i}. \tag{2.12}$$

$$\Delta b = \frac{1}{N}\sum_{i=1}^{N}\eta(t_i - y_i)\sigma'(h_i). \tag{2.13}$$

where $h_i = \overrightarrow{w} \cdot \overrightarrow{x_i} + b$

So, for a set of N input-output pairs $X = \{(\overrightarrow{x_1}, y_1), (\overrightarrow{x_2}, y_2), ..., (\overrightarrow{x_N}, y_N)\}$ learning involves continual updating of $\overrightarrow{w}$ and b values as per the delta rule. This includes the:

1. Calculating the forward values: Namely, y_i and h_i for all x_i.

2. Calculating the backward values: Updating the weight vector and bias in accordance to gradient descent. Use the delta rules and the values computed in the forward stage to calculate the deltas (for each weight and bias).

When the backward values are calculated it can be used to update the weight vector and bias. This is done iteratively until the value of $E(X)$ converges.

2.5 Types of Neural Networks

There are many kinds of neural networks that are used for a variety of things. The main branches are classification and regression problems. In classification, neural networks can be used for image and speech recognition, document categorization, email filtering and more. NNs can also be used for regression problems such as sales forecasting, stock price prediction and other time series related problems.

2.5.1 Feedforward Networks

Feedforward (FF) neural networks are ANNs where the connections between each of the nodes do not form a cycle. These were the first type of NNs to be invented as discussed in Chapter 1. Information only travels in one direction in these networks, first from the input nodes then through the hidden nodes (if existing) and then onto the output nodes. Every perceptron is connected to a node in the next layer which means all the nodes are fully connected. The hidden layers can have any amount of neurons (or nodes) and the number of hidden layers depends on the nature of the problem.

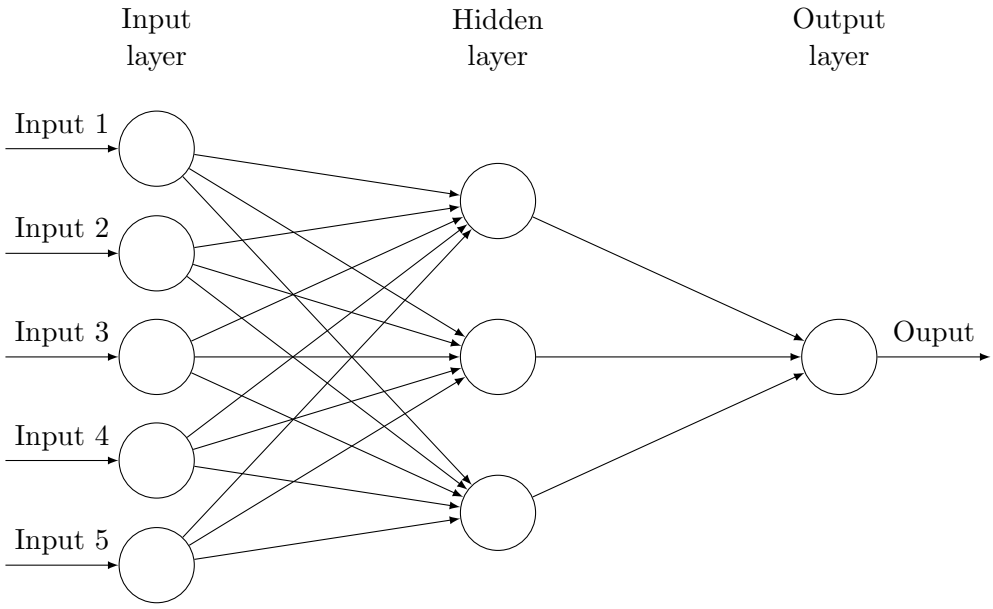

Figure 2.9: Feed-forward neural network structure.

This is a feed-forward neural network with five inputs, one hidden layer and one output. The outputs from the input layer is fed into each neuron in the hidden layer. Then this information passes onto the neuron in the output layer and an output is computed.

The main characteristics of a feed-forward network are as follows:

1. Perceptrons are arranged in layers. The first layer takes in input whereas the last layer gives outputs. The middle layers do not connect to the external environment and hence are called hidden layers.

2. Each perceptron is connected to another perceptron in another layer, hence information is 'fed-forward'.

3. Perceptrons in the same layer are not connected.

Back-Propagation Method

In feed-forward neural networks, the type of learning that's used is supervised learning. This is where pairs of input-output values are fed into the network for many rounds so that it 'learns' the correlation between the input and output.

The back-propagation algorithm is used to help the network learn. It is based on the delta rule discussed in section 2.4. Essentially, the delta rule was a special case of backpropagation for single layer perceptrons.

Let us take o_k as the input to current neuron k i.e output of previous neuron j. If neuron k is the output neuron and the activation function is given by equation $y = \sigma(o_k)$, then the generalized delta rule can be given by,

$$w_{kj}(n+1) = w_{kj}(n) - \eta g_{kj}, \tag{2.14}$$

where

$$g_{kj} = (y_k - t_k)\frac{\partial \sigma}{\partial o_k}. \tag{2.15}$$

In this case, as neuron k is an output neuron, equation 2.14 can be applied to adjust the weights of the synapses. If however, neuron j is a hidden neuron in a layer before neuron k then another algorithm is needed.

When neuron j is in a hidden layer, the backpropagation algorithm is formulated as follows,

$$w_{ji}(n+1) = w_{ji}(n) - \eta g_{ji}. \tag{2.16}$$

where

$$g_{ji} = \sum_k \left((y_k - t_k)\frac{\partial \sigma}{\partial o_k} w_{kj} \right) \frac{\partial \sigma}{\partial o_j} u_i. \tag{2.17}$$

Let us take a look at a straightforward example.

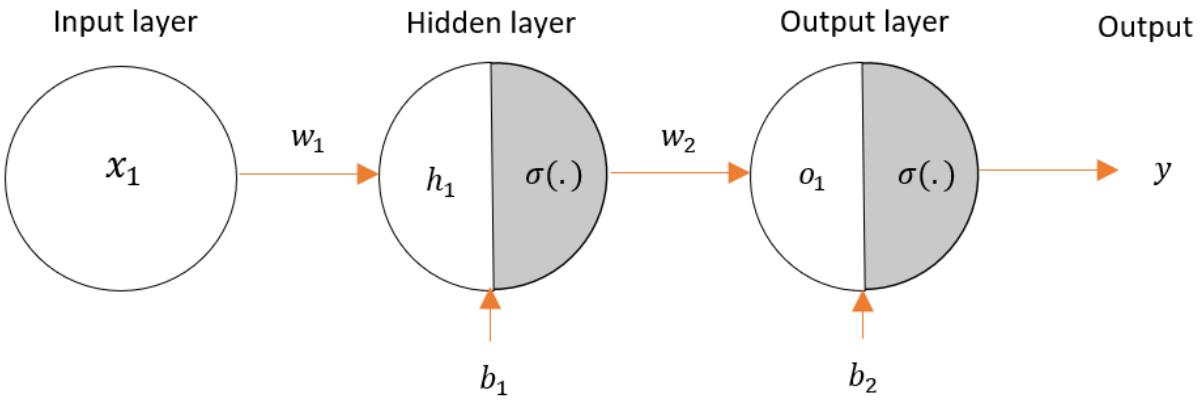

Figure 2.10: Backpropagation: Feedforward.

where,

$h_1 = x_1 w_1 + b_1$.

$o_1 = w_2 \cdot \sigma(h_1) + b_2$.

$y = \sigma(o_1)$.

$\sigma(h_1) = \frac{1}{1+e^{-h_1}}$ which is the Sigmoid function 2.5.

Error $= E(X) = \frac{1}{2}(t-y)^2$.

Then for weight w_2 the backpropagation is,

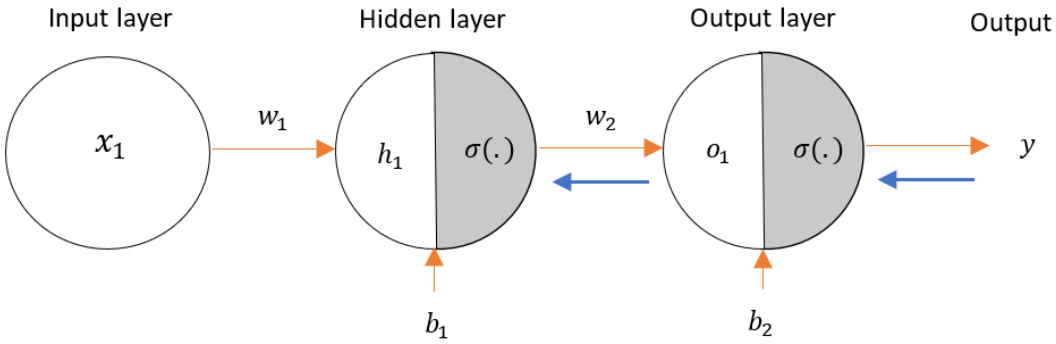

Figure 2.11: Backpropagating - the blue arrows show the information travelling backwards.

We can update the weight w_2 with equation 2.10. The partial derivative $\frac{\partial E(X)}{\partial w_2}$ is given by,

$$\frac{\partial E(X)}{\partial w_2} = \frac{\partial E(X)}{\partial y}\frac{\partial y}{\partial w_2} = \frac{\partial E(X)}{\partial y} \cdot \frac{\partial y}{\partial o_1} \cdot \frac{\partial o_1}{\partial w_2}$$

$$\frac{\partial E(X)}{\partial w_2} = (t-y) \cdot \sigma'(o_1) \cdot \sigma(h_1)$$

$$\frac{\partial E(X)}{\partial w_2} = (t-y) \cdot \sigma(o_1)(1-\sigma(o_1)) \cdot \sigma(h_1) \tag{2.18}$$

Now we can move backwards through weight w_1

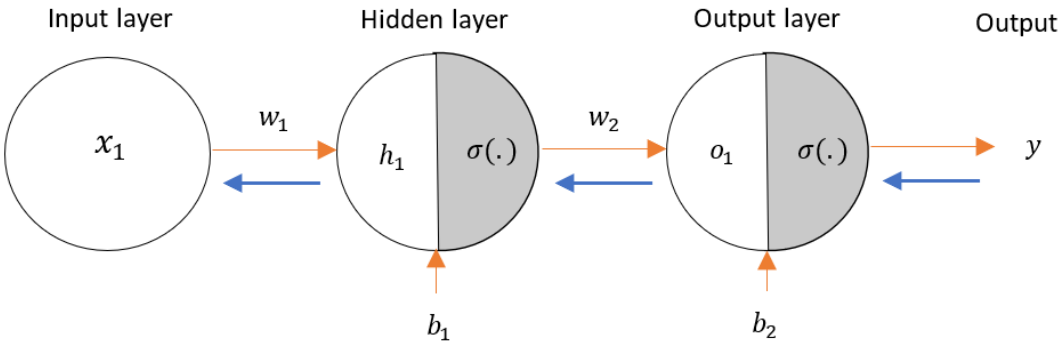

Figure 2.12: Backpropagating - backward pass for w_1.

The partial derivative $\frac{\partial E(X)}{\partial w_1}$ is given by,

$$\frac{\partial E(X)}{\partial w_1} = \frac{\partial E(X)}{\partial y} \cdot \frac{\partial y}{\partial o_1} \cdot \frac{\partial o_1}{\partial \sigma(h_1)} \cdot \frac{\partial \sigma(h_1)}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1}$$

$$\frac{\partial E(X)}{\partial w_1} = (t-y) \cdot \sigma(o_1)(1-\sigma(o_1)) \cdot w_2 \cdot \sigma(h_1)(1-\sigma(h_1)) \cdot x_1 \tag{2.19}$$

Now that we have the partial derivatives $\frac{\partial E(X)}{\partial w_2}$ and $\frac{\partial E(X)}{\partial w_1}$ we can calculate the weights w_2 and w_1 as,

$$w_2 = w_2 - \eta \cdot \frac{\partial E(X)}{\partial w_2}$$

$$w_1 = w_1 - \eta \cdot \frac{\partial E(X)}{\partial w_1}$$

The new weights can then be used in the calculations for feedforward phase.

Applications of Feedforward Neural Networks could include: speech recognition, data compression, handwritten character recognition, pattern recognition and more.

2.5.2 Radial Basis Network

The radial basis network (RBN) neural network is a 3-layer feedforward network with an input layer, hidden layer (which consists of Gaussian activation functions) and an output layer.

These networks are usually used for function approximation and have a faster learning rate than other NNs. The key difference between RBNs and Feedforward NNs are that they use the Radial Basis Functions (RBF) as an activation function. This is due to the fact that logistic (sigmoidal) activation functions cannot be used for continuous values as they give an output between 0 and 1 for yes or no. Hence, using the non-linear RBF, better approximations can be made for continuous values.

Applications

- Function approximations
- Classification
- Time Series prediction

2.5.3 Recurrent Neural Network

Recurrent neural networks (RNN) are a variation to FF networks. They are renowned for their 'memory' as they use information from previous inputs to determine the current inputs and outputs (information cycles in a loop). For example if we are trying to know the next word in a sentence, we need to know the words used prior. This makes it ideally suited to solve machine learning problems involving sequential data.

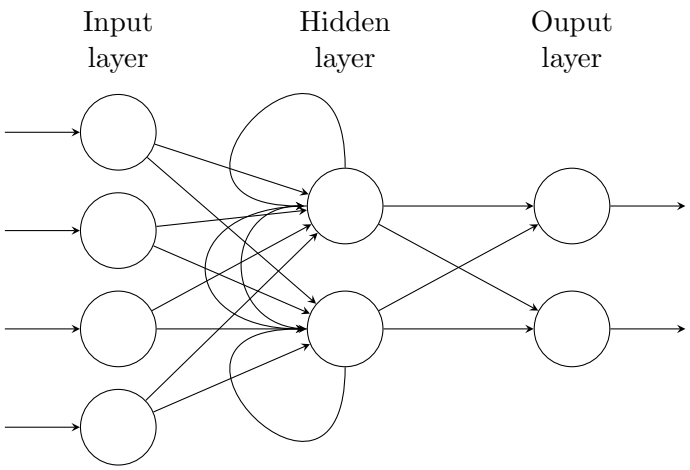

Figure 2.13: Recurrent neural network structure

In RNNs, neurons in the hidden layer receive inputs with a delay in time. It produces output which is copied and then fed back into the network. Hence, a RNN has 2 inputs: the present and the recent past. Weights are applied to both inputs and they are adjusted using gradient descent and backpropagation through time (BPTT).

BPTT is another word for backpropagation on unrolled recurrent neural networks. With BPTT,

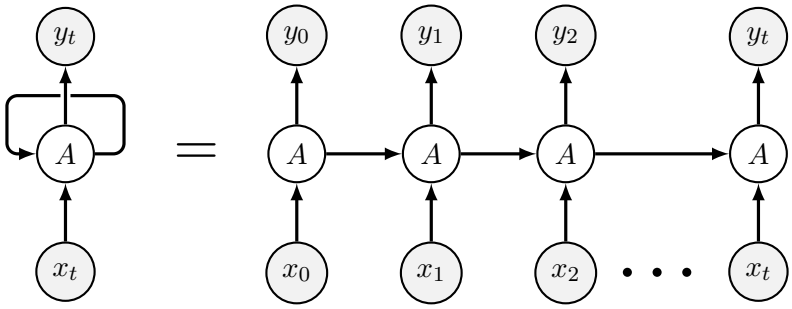

Figure 2.14: An unrolled RNN: we can view an RNN as a sequence of NNs that we train one by one. On the left, the RNN is unrolled after the equal sign.

the perception of unrolling is needed as calculation of the error on the present timestep depends on the previous time step. The error is backpropagated through all the time steps starting from the last timestep to the first timestep.

However, there are cons whilst using standard RNNs. They are:

Exploding gradients

Exploding gradients occur when the network assigns senselessly high updates to the models weights during training without much rationality. This in turn cases the model to be unstable and unfitted to learn from training data. Fortunately, this problem can be solved by truncating or squashing the gradients.

Vanishing gradients

This happens when the gradient is too small (due to the partial derivative of error with respect to weight being less than 1) and the model stops learning or takes too long to learn. This problem can be solved with the Long Short-Term Memory RNN proposed by Sepp Hochreiter and Juergen Schmidhuber.

2.5.4 Long Short-Term Memory

Long Short-Term Memory (LSTM) networks are an extension of recurrenct neural networks which increases the memory, hence making them suited to learning long-term dependencies. By the introduction of Constant Error Carousel (CEC) units, LSTM can deal with the vanishing gradient problem.

A usual LSTM comprised of a cell input, output and forget gates. The cell can remember values over random time intervals whilst the gates control what information goes into and out of the cell.

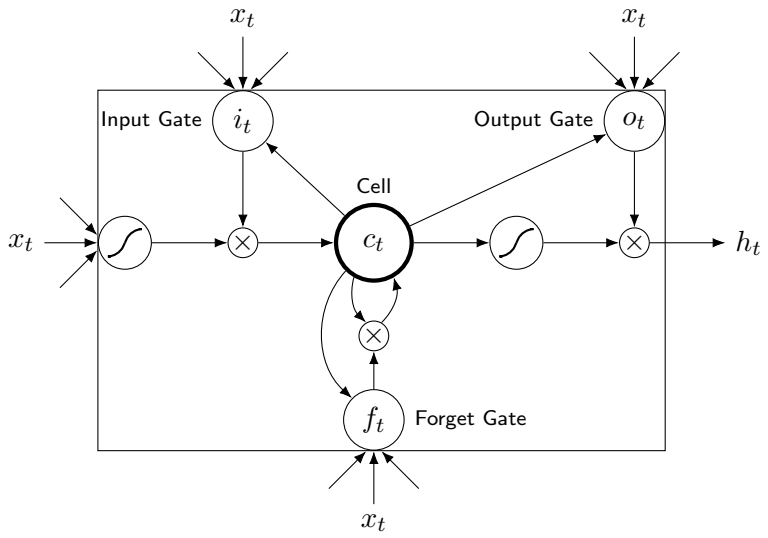

Figure 2.15: Long Short-Term Memory network. x_t represents the input into a particular gate, x represnts pointwise multiplication, i_t is the input gate output, f_t the forget gate output, o_t the output gate output. c_t is the cell state and h_t is the hidden state.

First, the forget gate determines what information should get kept or thrown away. Then with the input gate, the previous hidden state and current state is passed through a sigmoid function and a tanh function to help in updating of vales of regulation of network. Finally the sigmoid and tanh output are multiplied together. With the cell state, the forget vector gets pointwise multiplied with the cell state, then the input gate output gets added which updates the cell state to new values relevant to the network. Lastly, we have the output gate which decides what the next hidden state should be. The previous hidden state and current input is put through a sigmoid function,, then the cell state is passed through a tanh function. The multiplication of both outputs, tanh and sigmoid decide what the next hidden state should be.

Applications

- Time series prediction

- Speech recognition
- Music generation

2.5.5 Convolutional Neural Networks

Convolutional neural networks (CNN) are a group of NNs that are very effective in areas such as image recognition, classification, clustering and object recognition.

Convolutional neural networks have a lot of distinct layers. They are, the input layer, convolutional layer, pooling layer, fully connected layer, softmax/logistic layer and the output layer.

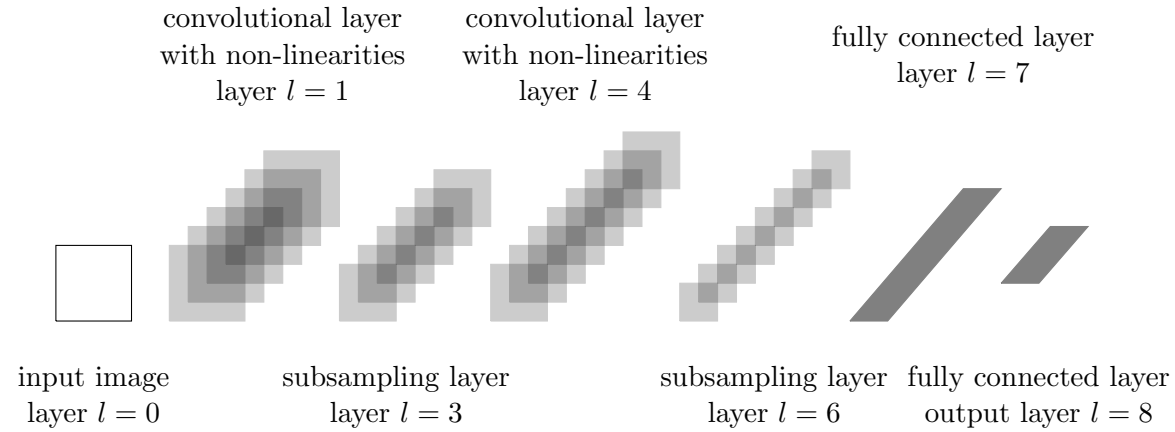

Figure 2.16: The architecture of a simple convolutional neural network (without details of number of channels or neurons or input image size).

The input layer of the CNN takes in image data. Next, the convolutional and pooling layer breaks down an image into features and analyzes them. The fully connected (FC) layer consists of weights, biases and neurons. It takes the output from the convolutional/pooling layer and estimates the best label to describe the image.

The softmax/logistic layer is situated at the end of the FC layer. Softmax is used for multi-classification whilst logistic is used for binary classification.

Applications

- Identify faces, tumors, street signs
- Video analysis
- Natural language processing

Chapter 3

Financial Mathematics

Financial mathematics (FM) is the use of mathematical models and large datasets to analyze and model financial markets and securities. It employs tools from statistics and probability as well as stochastic processes and economic theory.

Areas in which FM is used include investment banks, hedge funds, insurance and investment management companies, regulatory agencies and corporate treasuries. These institutions apply mathematical methods to problems such as risk management, portfolio structuring, derivative pricing and stock price forecasting.

3.1 Understanding Financial Assets

A financial asset is a non-physical asset that gets its worth from a contractual right or claim for example bonds, bank deposits and stocks. They're an asset that is liquid and hence can be sold quickly (unlike physical assets such as property and art). The value of financial assets depends on the supply and demand in the market and the risk they carry. There are many different types of financial assets:

- **Stocks** - Has no set ending or expiration date. A person buying stocks owns part of a company and shares in its profits and losses. Stocks can be sold to other investors or held.

Figure 3.1: Example of companies where you can buy stocks in. (*Marketwatch photo illustration/istockphoto* 2020)

- **Financial derivatives** - Complex in nature, these instruments derive their value from stocks

Figure 3.2: Financial derivatives (*Derivatives, With Their Risks and Rewards* 2020)

- **Bonds** - A fixed income instrument. It is a loan to a company or government that pays interest over time, the company/government then repays the principal sum of the loan at the bond maturity date.
- **Certificate of deposit** - Allows investors to deposit money into a bank for a certain period with a guaranteed interest rate.

Investors are constantly looking to make the best return on their capital. Although investing is

much as an art as a science, rational decisions must be taken which can be done by evaluating the risk and return of different strategies. People who want to take less risk may put their wealth in bonds or cash. Whereas the more risk-taking individuals will invest in stocks or long-term bonds

3.2 Financial Derivatives and the Stock Market

Financial derivatives are financial instruments where the value is derived from an underlying asset. Such an asset could be shares, currencies, commodities or stock indices. Derivatives are usually distinguished as contractual obligations between two groups with opposing views of market conditions (Stankovska 2016). Most derivatives are also leveraged, which increases the risk-to-reward ratio.

3.2.1 Stock Options

Options are the most popular types of financial derivatives available to traders and investors. A stock option is the right to buy or sell a certain stock at a specific price (known as the 'strike' price) at any time before the contracts expiration date*. A right to buy a share is known as a 'call' option whereas a right to sell a share is known as a 'put' option. For example a $140 Apple call of June 2021 gives the holder the right to buy a share of Apple stock at $140 before or on the third Friday of June 2021.

*Note: For european options, the investor can only carry out the trade on the expiration date and not any time before.

A well established problem in finance is the valuation of option contracts. This problem is solved by the Black-Scholes model.

Let us assume a stochastic model for the share price of an underlying asset:

$$\frac{dS}{S} = \sigma dZ + \mu dt \tag{3.1}$$

Where dZ stands for a standard Weiner process. Suppose V is the value of a call option dependent on S. Now since V must be a function of S and t we apply Ito's lemma and obtain,

$$dV = (\frac{\partial V}{\partial S}\mu S + \frac{\partial V}{\partial t} + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}\sigma^2 S^2)dt + \frac{\partial V}{\partial S}\sigma S dZ \tag{3.2}$$

Let us consider a portfolio, in this case a delta hedge portfolio which consists of being short one option and long $\frac{\partial V}{\partial S}$ shares. The value of this portfolio is given as,

$$P = -V + \frac{\partial V}{\partial S}S \tag{3.3}$$

Over the time period Δt the change in the value of the portfolio is given by,

$$\Delta P = -\Delta V + \frac{\partial V}{\partial S}\Delta S \tag{3.4}$$

We can discretise Equation 3.1 and 3.2 by replacing the differentials with Δs:

$$\Delta S = \sigma S \Delta Z + \mu S \Delta t \tag{3.5}$$

and

$$\Delta V = (\frac{\partial V}{\partial S}\mu S + \frac{\partial V}{\partial t} + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}\sigma^2 S^2)\Delta t + \frac{\partial V}{\partial S}\sigma S \Delta Z \tag{3.6}$$

Substituting Equation 3.5 and 3.6 into Equation 3.4 gives us,

$$\Delta P = (-\frac{\partial V}{\partial t} - \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2})\Delta t \tag{3.7}$$

Now since this equation does not have ΔZ, the risk has essentially been eliminated from the portfolio in the time interval Δt. Using no arbitrage arguments the return on this portfolio must be equal to the risk free return. Assuming the risk free rate of return is r, then over the time period Δt we have,

$$\Delta P = rP\Delta t$$

Substituting from Equations 3.3 and 3.7 we get,

$$(-\frac{\partial V}{\partial t} - \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2})\Delta t = r(-V + \frac{\partial V}{\partial S}S)\Delta t$$

so that,

$$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2})\Delta t = rV \tag{3.8}$$

Equation 3.8 is the celebrated Black-Scholes equation for derivative pricing. The solution to the equation depends on the boundary conditions that are established by the type of derivative.

Usually, analytical solutions to partial differential equations do not exist, however for European call and put options, a solution exists. The analytical solution for a European call option is given by Equation 1.2 as discussed in Chapter 1.

Black-Scholes - Pricing of a Put Option

The put-call parity relationship is,

$$C + Ke^{(-r(T-t))} = P + S$$

We can use this put-call parity relationship to price a European put option. We know that,

$$\begin{aligned} c + Ke^{-}rT &= p + S_0 \\ \Rightarrow p = Ke^{-rt}(1 - N(d_2)) - S_0(1 - N(d_1)) \\ = Ke^{-rt}N(-d_2) - S_0N(-d_1)) \end{aligned} \tag{3.9}$$

Example: Let's say a share price is currently £5 and the strike price K is £4. The volatility σ is 20% per annum and the risk-free rate r is 5% per annum. Find the price of six month call and put options on the share. We have (from equations 1.3 and 1.4),

$$d_1 = \frac{\ln(5/4) + (0.05 + 0.2^2/2) \times 0.5}{0.2 \times \sqrt{0.5}} = 1.825$$

$$d_2 = 1.825 - 0.2\sqrt{0.5} = 1.684$$

and from tables, $N(1.825) = 0.97$, $N(1.684) = 0.95$, $N(-1.825) = 0.03$, $N(-1.684) = 0.05$. And $4 \times e^{-0.05 \times 0.5} = 3.901$ so that,

$c = 5 \times 0.966 - 3.901 \times 0.954 = 1.14$

$p = 3.901 \times 0.05 - 5 \times 0.03 = 0.045$

So the call option has a price of £1.14 and the put option has a price of £0.045

3.2.2 Futures Contracts

A futures contract is an agreement to buy or sell an asset or commodity at a specified price in the future. Unlike options, where the buyer gets the right (not the obligation) to buy or sell an underlying asset before the expiration date or on the date (for European options), with

futures the buyer is obligated to take ownership of the underlying asset at the expiration date and cannot do it anytime before (R. A. Jarrow and Oldfield 1981).

These derivatives allow traders to speculate on the future price of an underlying entity, be it gold, oil currencies or other economic vehicles. Futures are 'cash-settled' meaning if the holder of the futures contract decides to execute its terms, they just receive cash should their speculation be correct. Taking a 'long' position means the trader owns the security, whereas taking a 'short' position means the trader owes those shares to someone however does not own them yet i.e the trader must buy those shares at the settlement price (the underlying price at the time of the contracts maturity).

If say we have K as the delivery price and S_T as the settlement price. Then the value of the futures position at the maturity date is

- For long positions, the outcome is $S_T - K$ and a gain will be made from a higher settlement price.
- For short positions, the outcome is $K - S_T$ and it will benefit from a lower settlement price.

3.2.3 Forward Contracts

Forward contracts are similar to futures in that the buyer and seller agree to trade an asset at a specified price at a future date. However, with forwards the there is just one settlement date meaning they all settle at the end of the contract (*Forward Contracts vs. Futures Contracts* 2020), unlike futures where daily changes are settled day by day and settlement for contracts can happen over a range of dates.

Forwards contracts are private and do not trade on an exchange and so are not easily accessible to retail traders. The details of the contract are also kept private between the buyer and seller which means there is a high risk as either party can default (fail to meet their obligations of the contract).

Example

Assume that an agricultural producer has 1 million ears of corn to sell three months from now. The producer feels that the corn prices will fall and hence enters a forward contract with a

financial institution to sell 1 million corn ears at a price of $1 three months into the future.

After three months there are 3 possibilities for the spot price:

1. It is exactly $1 per corn ear. In this case neither the producer nor the financial institution gain anything and the contract is closed.
2. It is lower than the contract price, for example $0.85 per corn ear. The producer will be paid $150,000, or the difference between the contract price and the spot price.
3. It is higher than the contract price, for example $1.10 per corn ear. The producer now owes the institution $100,000, or the price difference of the current spot price and and contract rate price.

3.2.4 Advantages of Financial Derivatives

1. Hedging Risk - The value of a derivative is linked to the underlying asset price and so investors can use it to reduce their losses. For example an investor buys contracts which move in the opposite direction to the underlying asset. Hence, losses in underlying assets can be offset by profits in contracts. (*Derivatives* 2021)
2. Improve market efficiency - Derivatives are considered to increase the markets efficiency. Contracts can be used to replicate the payoff of assets.
3. Determine Underlying Asset Price - Derivatives can be used to determine the underlying price of an asset. E.g the spot price of a futures contract can be used to approximate a commodity price.

3.2.5 Disadvantages of Financial Derivatives

1. High Risk - Due to the high volatility of derivatives, they are prone to resulting in potentially large losses. Estimating and valuing these contracts are very difficult and sometimes impossible, therefore carry a big risk.
2. Needs expertise - Investors require a deep understanding of trading in these instruments in contrast to other assets like normal stocks and metals.
3. Counter-party risk - There is a chance for counter-party default as a lot of the contracts traded off an exchange (over-the-counter) do not have a benchmark for due diligence.

3.3 Time Series

A time series is a sequence of numerical points all spaced out equally in time. Hence, it is a sequence of discrete-time data. Examples of time series arise in various fields such as economics, physical sciences, demography and marketing. These examples could be closing share prices, monthly product sales, weather prediction, disease rates etc.

A time series can be taken on any variable over time. In financial maths, a time series can be used to model the price of a security over time. This can be tracked by the minute over the course of a day for the short-term, or for the long-term, the closing price of the security on every day over 1 year.

A time series that can be predicted exactly is called deterministic otherwise if there is uncertainty or errors involved it is stochastic. Statisticians are for obvious reasons interested in the stochastic types.

3.3.1 Components of a Time Series

A time series can be seen as made up of different components.

- Trend component (long term direction)
- Seasonality component (calendar related movements)
- Irregular component (unsystematic, short term variation)
- Cyclical component (periodic variations, not seasonal)

Several time series are prone to increase or decrease over long time periods. This is known as a 'trend'. For example, inflation figures tend to fall slowly over a few years. However, this trend will reverse and will naturally increase over time (*United Kingdom Inflation Rate* 2020).

Let us consider the closing price of Alphabet Inc stock over 2 years.

Figure 3.3: Plot made using data obtained from *Yahoo Finance* (2021)

Observing the plot, we can notice an evident trend - the stock price is gradually increasing over time. There is also an irregular component as we can see the roughness within the time series.

A lot of time series comprising of quarterly or monthly patterns shows what is known as 'seasonality'. As in, we see the same patterns repeating in certain months each year. For example, consider the number of airline passengers flying an airline each month.

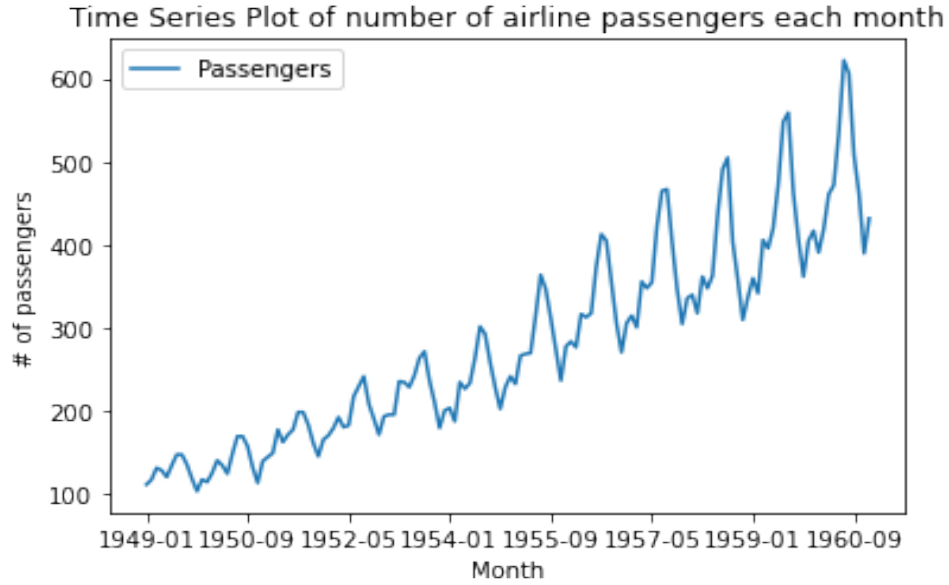

Figure 3.4: Dataset obtained from *Air Passengers Dataset* (2020)

We can observe that their is a frequent pattern from year to year. More passengers use the

airline in the summer months than in the winter months. Also there is an overall uptrend for the number of passengers for the airline over time and its visible that there are peaks during December - January which is because people go for their winter break during that period.

Some time series have a cyclical pattern which is uncorrelated to seasonal behaviour. For example number of housing sales each month.

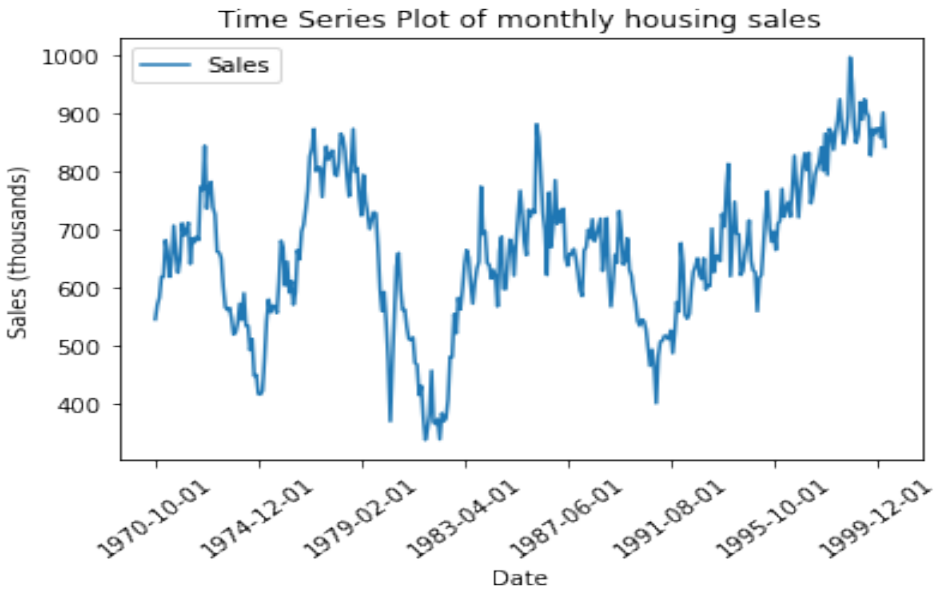

Figure 3.5: Using data obtained from *New One Family Houses Sold* (2021)

This series has no particular trend but a regular cyclical pattern. The period for the housing sales cycle is not constant. In economics, the cyclical patterns can match business cycles in the economy as a whole.

3.3.2 Time Series Analysis and Forecasting

Time series analysis involves analysing time series data to obtain meaningful statistics and other attributes of the data. It can be used to see how the data points change in relation to other variables that shift in that same time period.

For example, if sales of a company were good last month due to favourable economic conditions it is unlikely that sales would be poor in the current month as an abrupt change of the economic conditions would be very unlikely.

Another feature of time series is seasonality. For example you could make the regular observation that retail sales increase during December and can hence be likely to remain like that for future

years. Time series are hence said to be dependent. We can normally make use of the fact that if sales in the current month are akin to the levels of sales in previous months then sales in the future months will also be similar.

An additional example would be how the stock price of a particular company goes through highs and lows at regular times throughout the year (*Fresnillo Share Price* 2021). This information can be used to predict that these levels would be similar during the same seasons in other years.

Alternately, the stock price can be affected by economic variables like the interest rate or the rate of unemployment. By correlating the data points with the designated variable we can detect patterns between the data points and the variables.

Time Series Forecasting

Time series forecasting uses information from historical values and patterns to predict the future values. Usually this relates to cyclical fluctuation analysis, trend analysis and cases of seasonality.

The most common time series (non-neural network) forecasting algorithms are

- Autoregression (AR) - Uses a linear combination of past values to forecast future. It is a regression of the variable against itself. An auto-regressive model of order p can be written as

$$x_t = \sum_{i=1}^{p} \alpha_i x_{t-i} + \epsilon_t \tag{3.10}$$

  Where ϵ_t is white noise. This is similar to a multiple regression model but with delayed values of x_t as predictors.

- Moving Average (MA) - This model merely takes an average of all the values from time $t_i - k$ through t_i for every time index $i > k$. This results in a smoother time series and makes trends more evident. A larger k value causes a smoother trend.

- Autoregressive Integrated Moving Average (ARIMA)- Is one of the most typical parametric models. The models goal is to forecast future values by exploring the differences between the values in the series instead of using the actual values (Koenecke n.d.).
  Given a time series with data X_t where t is an integer index and X_t are real numbers, the ARMA(p, q) model can be written as

$$X_t - \alpha_1 X_t - 1 - ... - \alpha_p X_{t-p} = \epsilon_t + \theta_1 \epsilon_{t-1} + ... + \theta_q \epsilon_{t-q} \tag{3.11}$$

or equivalently it can be written as

$$\left(1 - \sum_{i=1}^{p} \alpha_i L^i\right) X_t = \left(1 + \sum_{i=1}^{q} \theta_i L^i\right) \epsilon_t \tag{3.12}$$

where L is the lag operator, α are the parameters of the autoregression, θ are the parameters of the moving average and the ϵ are the error terms deduced to be independent and identically distributed centered at zero.

Now we can progress to ARIMA, which incorporates integration. More precisely, it considers the order of integration. Integration I(d) uses the parameter d, the unit root. And the ARIMA model is generalized as

$$\left(1 - \sum_{i=1}^{p-d} \alpha_i L^i\right) (1 - L)^d Y_t = \left(1 + \sum_{i=1}^{q} \theta_i L^i\right) \epsilon_t \tag{3.13}$$

- Exponential Smoothing (ES) - past observations are used with weights to predict into the future. It's different to ARIMA models as it assigns exponentially decreasing weights to the historical observations.
  Exponential smoothing is defined as:

$$s_t = \alpha x_t + (1 - \alpha) s_{t-1}, \;\; t > 0 \tag{3.14}$$

  Where the algorithm output is s_t which is formatted to $s_0 = x_0$ and x_t is the original time series sequence. The smoothing coefficient $0 \leq \alpha \geq 1$ enables one to set how quickly weights decrease over past observations. We can also carry out exponential smoothing iteratively, that is introducing a second coefficient, β as the trend smoothing factor and call it 'double exponential smoothing' (*A Gentle Introduction to Exponential Smoothing* 2021). For 'triple exponential smoothing' we introduce a third coefficient γ which is the smoothing factor for the seasonality. A negative of this model is that past data are forgotten quite quickly.

Time series forecasting is not always guaranteed to be successful, as is the case with all forecasting methods.

3.3.3 Financial Time Series

Financial time series is affiliated with the theory and application of pricing assets over time. It is a particularly hands-on field of study but forms the foundation of making conclusions. There is a key attribute that differentiates financial time series from other time series. Both financial and empirical time series have an element of uncertainty. However in financial time series, for example in a stock return series the volatility is not directly evident. Due to the extra volatility, statistical methods play a more important role in financial time series.

Examples of financial time series:

- The daily values of IBM stock price over a period of time
- The GBP-USD exchange rate
- S&P 500 index price
- VIX index of option prices

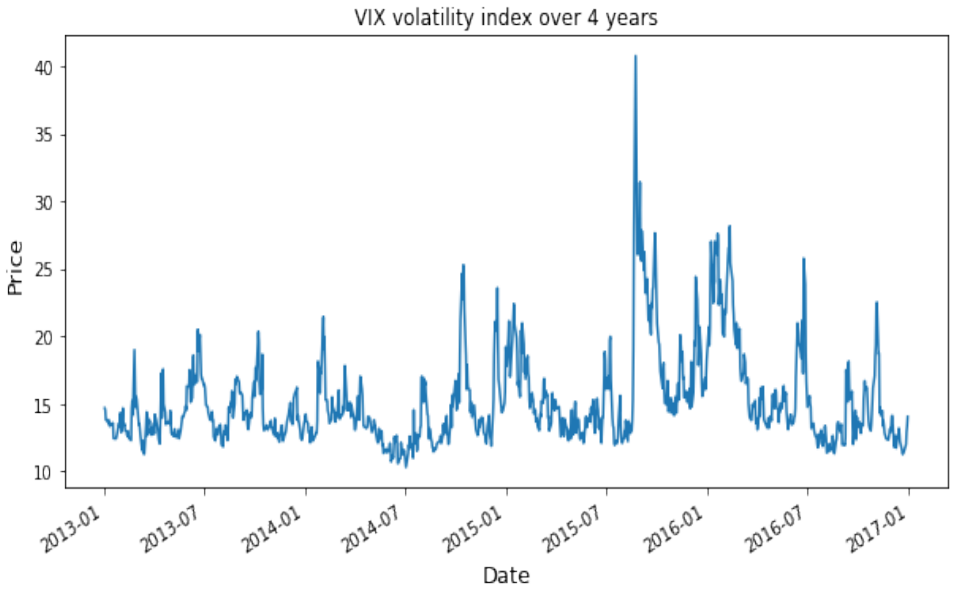

Figure 3.6: The VIX index i.e CBOE volatility index from 2013 to 2017.

Chapter 4

Application of Neural Networks in Finance

There are various areas of finance where neural networks can be used to improve the processes within financial industries. For example it can be used in currency prediction, futures prediction, bankruptcy predictions and more.

4.1 NNs to Predict Stock Price

Neural networks can be used to predict the price of a stock as White (1988), Chiang et al. (1996) and Gao et al. (2020) have shown in their papers.

There are multiple types of neural networks that are suited to predicting stock prices. They include MLPs, LSTMs and CNNs where the performance of each depend on the type and length of time series.

4.1.1 Using Multi-Layer Perceptrons (MLPs)

MLPs are a simple type of Feed-Forward Neural Network. It can be used to predict short-term stock prices (Turchenko et al. 2011). It is also not as accurate as other methods as we can see with Hussain (2020) work.

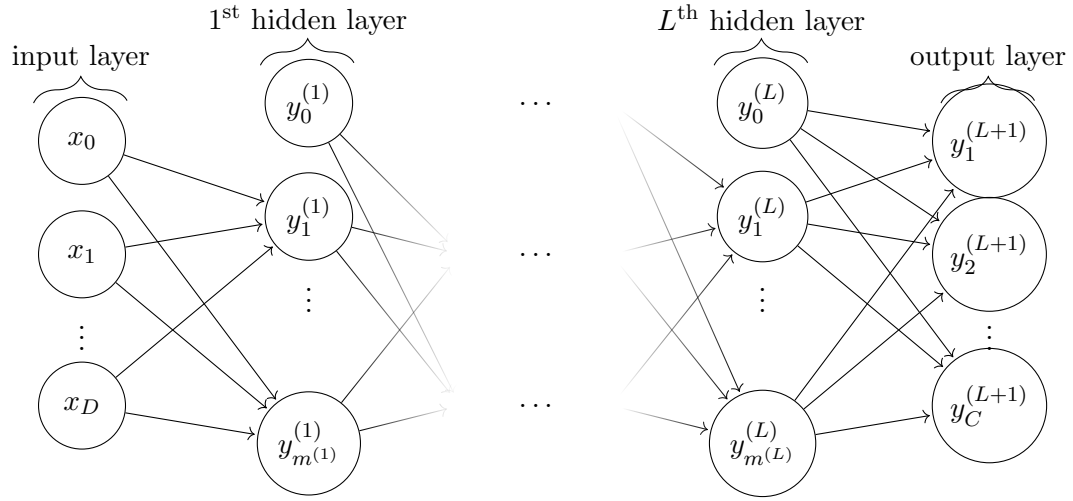

Figure 4.1: Graph for a multi-layer perceptron with $(L+1)$-layers. It has D input units and C output units. The l^{th} hidden layer contains $m^{(l)}$ hidden units.

A multi-layer perceptron like above can be used to forecast stock prices. It can take in features such as

- Date
- Daily open
- Daily high
- Price/earnings ratio
- Market capitalization
- Volume

If only the date is used as the feature to train on, we can get a general trend (training on data of 1200 days).

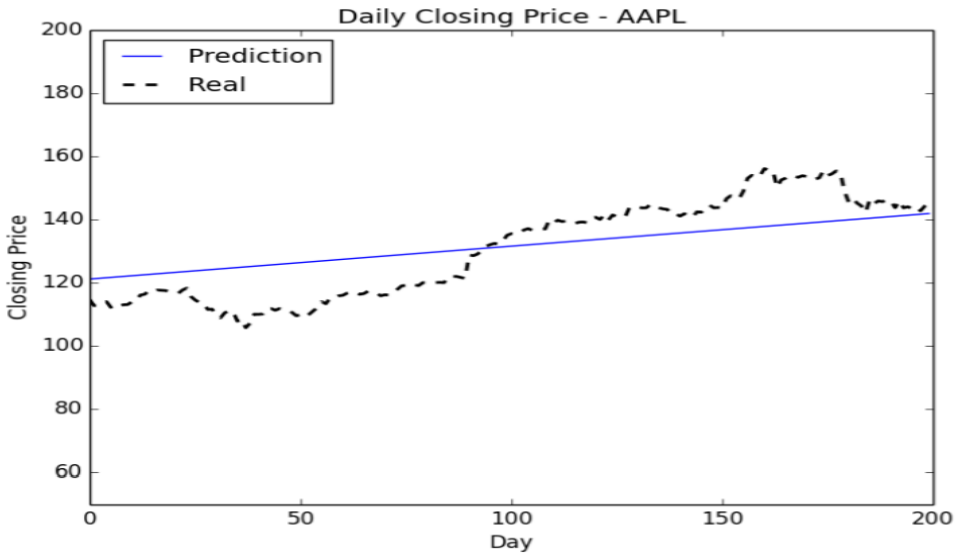

Figure 4.2: Predicting general price movement over a period of time (Hussain 2020)

If the MLP is trained with all features taken 15 days before the target price and used to predict 300 days into the future, the predictions come as follows.

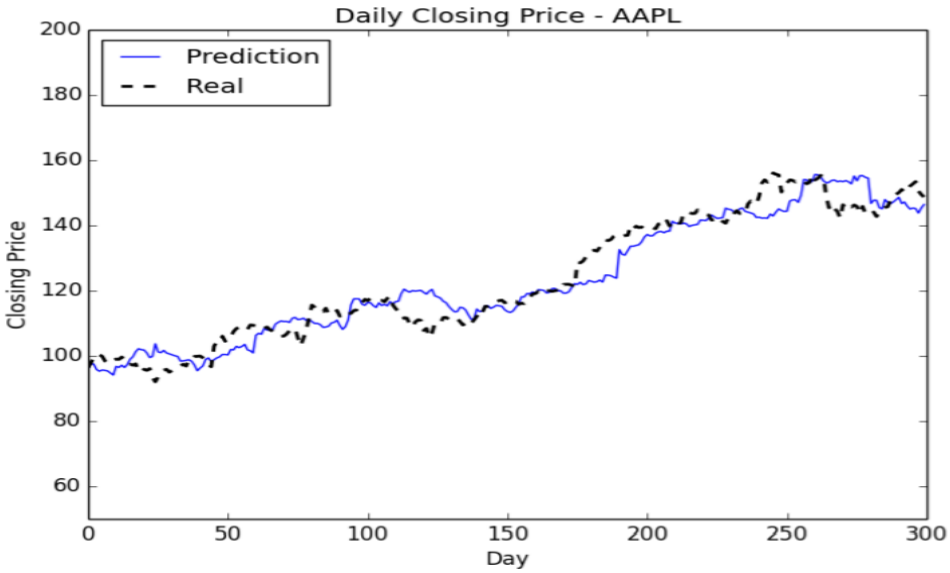

Figure 4.3: Predicting stock price based on features 15 days behind the target price. (Hussain 2020)

Hussain (2020) observed that the MLP is simply predicting the price of the day the features were taken from and that the price pattern of the predictions is shifted 15 days to the right. Meaning the NN is predicting past prices instead of future prices.

The MLP can also be used to train on all the other features except price. For example date, P/E ratio, market capitalization and volume. This will provide results that are less correlated

to features 15 days in the past, however will not be very accurate. Hussain (2020) found their to be errors of up to 24%. It is for this reason MLPs are only good to predict the general price trend of a stock but not accurately predict stock price.

4.1.2 Using CNN-LSTM model

CNNs can also be applied to predicting stock prices. It is composed of two main components: convolution layers and pooling layers. Each convolution layer has a number of convolution kernels, and the formula for calculating them is shown below:

$$l_t = tanh(x_t k_t + b_t) \tag{4.1}$$

where l_t is the output value after convolution, x_t is the input vector, k_t is the weight of the convolution kernel, b_t is the bias and $tanh$ is the activation function.

After features are extracted from the convolutional layer, the pooling layer is added due to the feature dimensions of the extracted data being too larger. The pooling layer solves this problem and reduces the cost of training the network.

LSTMs have been used vastly in speech recognition, text analysis and sentiment analysis. More recently, it has also been used to forecast stock prices The LSTM calculation process will be shown in 5.1 when we describe our methodology.

The process for the CNN-LSTM network are given in the diagram below:

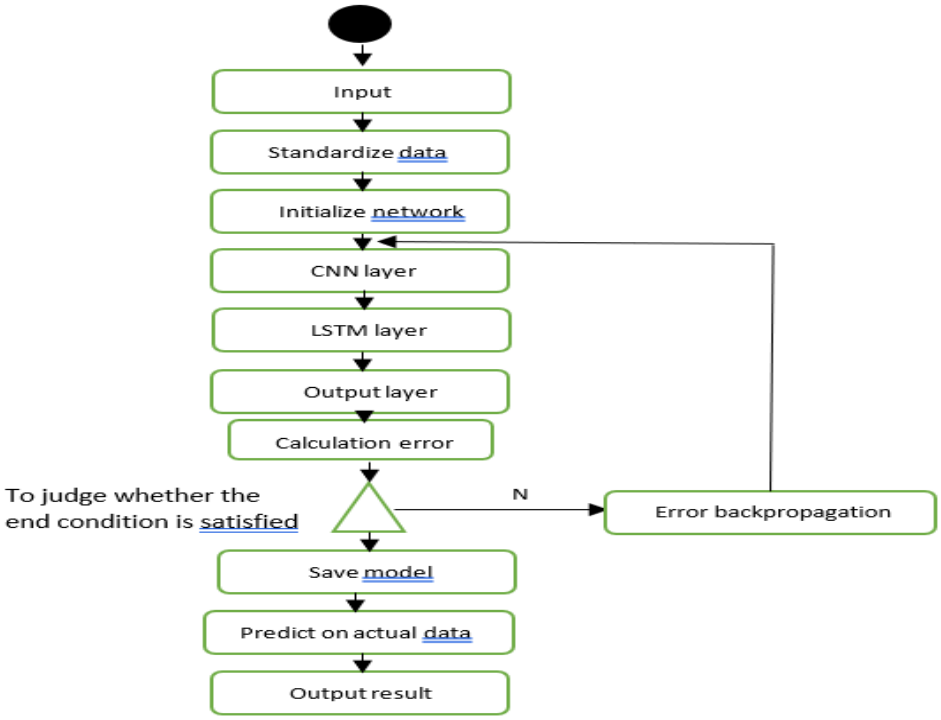

Figure 4.4: Predicting stock price based on features 15 days behind the target price.

The main stages are:

1. Input the data : Input the data needed for the CNN-LSTM network
2. Standardize data: In order to train the model better.
3. Initialize data: initialize the weights and biases of each layer
4. Calculation in CNN layer: The input data is passed through the convolution layer pooling layer of the CNN, the input datas features are extracted and the output value is collected.
5. Calculation in LSTM layer: The LSTM layer is used to measure the CNN layer's output data, and the output value is obtained.
6. Calculation in output layer: The output of the LSTM layer is fed into the fully connected layer to get an output value.
7. Calculation of error: The output values are compared with the real values of this group of data and the error is calculated
8. To decide whether the end condition has been met, it checks whether the model has completed a predetermined number of cycles and whether the weights and error are below a certain threshold. If the conditions are met, the training is complete and the CNN-LSTM

is updated. Move to step 10. If not go to stage 9.

9. Error backpropagation: Continue to train the network by propagating the measured error in the opposite direction, updating the weight and bias of each layer, and returning to stage 4.

10. Save model: save the completed model for predicting.

11. Prediction: Standardize data and input it into the trained model and get the output values

12. Output result: Restore the standardized data to its original values and output the results to finish the prediction process.

Wenjie employed this CNN-LSTM network and got results that were really accurate when compared to using MLPs. It was also shown how MLPs had the highest error in predicting stock prices compared to all the other neural networks attempting the same.

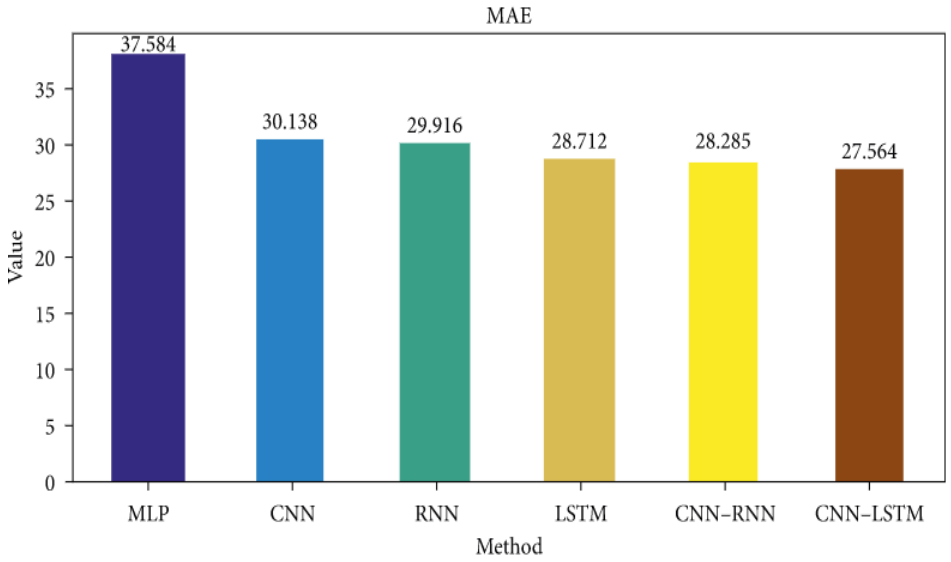

Figure 4.5: The mean absolute error of different types of NNs when predicting stock price. (**Wenjie**)

This shows that the MLP network is the least effective at predicting, followed by CNNs, RNNs, LSTMs. Whereas the most accurate types of networks were those that were a combination of 2 types of networks. That is, the LSTM-CNN and CNN-RNN networks.

4.2 Pricing Financial Derivatives with NNs

many pricing techniques have been used to price financial derivatives such as options. The most well-known of these techniques is the Black-Scholes option pricing model.

Researchers such as Yao, Li, et al. (2000) have used ANNs in forecasting the price of optins. The looked at the performance of backpropagation neural network on the Nikkei 225 index futures data. They grouped the data in different ways to find the best input mix. Also volatility wasn't used as an input in their model, however they specified volatility to be captured by the network. The researchers concluded that dividing the data differently generates various degrees of accuracy and that NNs are better at forecasting options prices than Black-Scholes when dealing with highly volatile markets.

The most famous models that have been used in the past for option pricing other than the Black-Scholes model are the Cox and Ross' pure jump model, Mertons mixed diffusion model (Merton 1976) as well as Hull and White's model where they addressed the constant variance assumption by modelling variance as a stochastic mechanism that allows for time variations.

When pricing options with neural networks a few variables need to be considered as input to the NN model. They are:

- Exercise price
- Time to expiration
- Volatility
- Closing price
- Interest rate

These are the same as the variables in the Black-Scholes model however, 2 additional variables can be added for the NN. E.g yesterdays market closing price and yesterdays market price of the option. For classification and prediction, feed-forward neural networks with single-hidden layer neurons have worked well. This model can be used for experiments on option pricing, with the backpropagation algorithm.

Malliaris and Salchenberger (1993) found that Black-Scholes model overprices options that are out-of-the money* and that NNs tend to underprice these options. Also, both the models

underpriced options that were in the money.

*Options that are out of the money are ones where the underlying price is trading below the stike price of the call or above strike price if its a put option. Call options that are in the money allow the option holder to buy the security at a price below the market price.

4.3 Algorithmic trading

Algorithmic trading could be used to 'buy' or 'sell' a certain security in order to make profit. This technique of trading can be done with many algorithms and deep neural networks are one of them.

4.3.1 Feature selection

For algorithmic trading, the features that are inputted to the neural network are a bit different to normal stock prediction as we work with a higher number of time steps in a shorter period of time. Four features could be used for inputting into the Deep NN as outlined by Arévalo et al. (2017): the Current Time, last n pseudo log returns, last n standard deviation of prices and last n trend indicators.

Each of the input groups are described as follows:

Current Time

This is split into two inputs: Current hour and Current minute. Lots of changes in price can happen within hours and minutes of a trading day. For this reason having the minute and the hour as an input can greatly help in the predictive process.

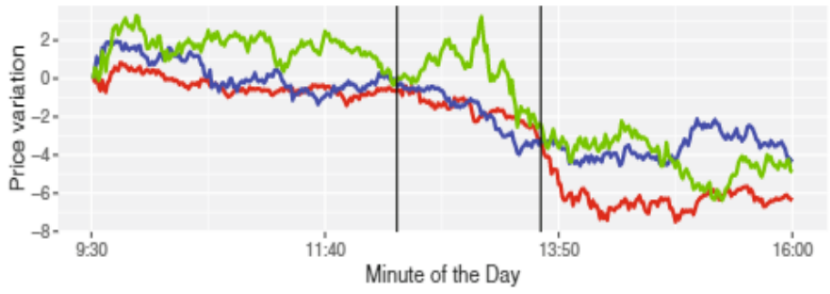

Figure 4.6: Price changes that occur within a single day. Blue, red and green line represent different days in the week. (Arévalo et al. 2017)

Last n psuedo log returns

We need to understand what a pseudo log return is - a logarithmic difference between **average** prices on successive minutes opposed to the log return which is the logarithmic difference between the **closing** prices of successive minutes.

Log return : Let's say p_t is the current 1 minute closing price and p_{t-1} is the previous 1 minute closing price. Then the log return is given as

$$\hat{R} = ln\frac{p_t}{p_{t-1}} \cdot 100\% = (lnp_t - lnp_{t-1}) \cdot 100\% \tag{4.2}$$

Pseudo log return :Now let's say $\hat{p_t}$ is the current 1 minute average price and $\hat{p_{t-1}}$ is the previous 1 minute average price then,

$$\hat{R} = ln\frac{\hat{p_t}}{\hat{p}_{t-1}} \cdot 100\% = (ln\hat{p_t} - ln\hat{p}_{t-1}) \cdot 100\% \tag{4.3}$$

For financial time series it is not recommended to use untransformed data to forecast future observations. This is because with financial markets, an asset price can trade for e.g in between \$150 - \$180 and the neural network will be trained for data in this range. However, when the model is tested in different market conditions e.g when the price is \$80 -\$110 the general performance of the model will drop due to the data not being transformed.

Now if the data is transformed into logarithmic returns, not only will the variance be equilibrated, but the time series will also not need to depend on past prices. For example, a trend may be discovered at the start of the chosen cycle, such as where the price increases by 10 cents, falls 40 cents, and then rises 65 cents. Such a pattern likely to continue in the future, hence it is good to use last n one-minute pseudo log returns as inputs

Last n standard deviations of prices

Another input to the deep neural network are last 1 minute standard deviation of prices

Last n trend indicators

This is also fed as an input to the NN. A 1 minute trend indicator is described as follows: A mathematical indicator derived from the gradient of a linear formula fitted to transaction prices

for a certain minute.

$$price = at + b$$

where t is time in milliseconds inside the individual minute.

4.3.2 Strategy

Arévalo et al. (2017) proposed a high-frequency trading strategy using a deep neural network.

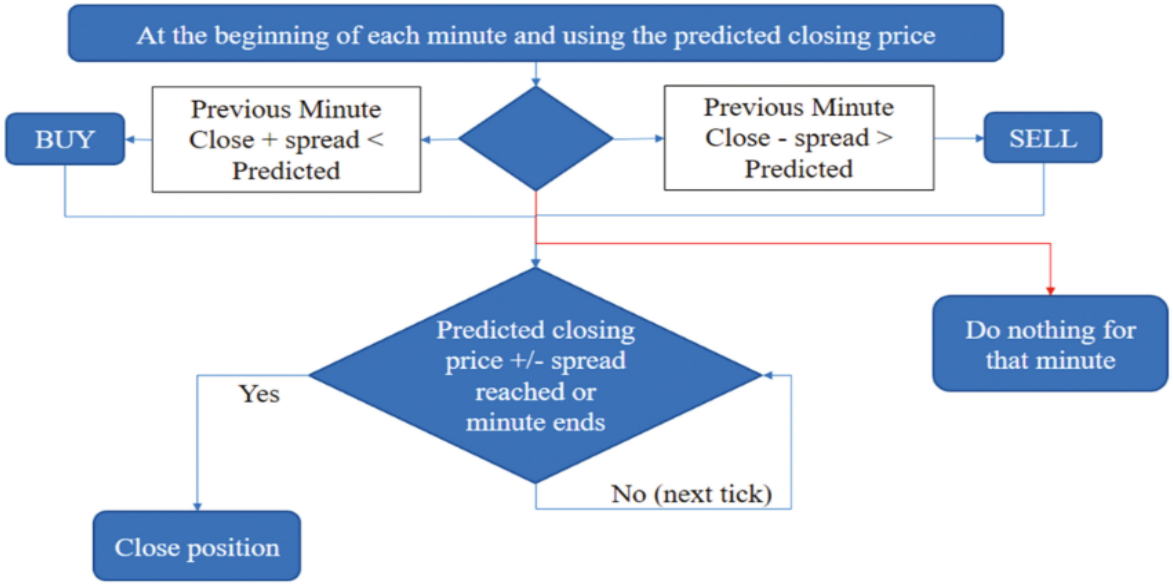

Figure 4.7: Strategy flowchart (Arévalo et al. 2017)

With this strategy, for each trading minute, the model 'buys' (or sells) the stock if the next predicted closing price is above (or below) the last closing price. If the stock price reaches the predicted price then the model 'sells' the stock to realize the gains. If the price does not reach the expected price it 'sells' the stock at the minute's last closing price and the position is closed.

After applying their strategy on Apple stock dataset by simulation, the model only yielded $0.28 in 8 days. They came to a conclusion that it has a good consistent performance and that the addition of time as an input proved to be an enhancement to the performance.

4.4 Using Neural Networks for Bankruptcy Prediction

The high personal, fiscal, and social costs associated with corporate defaults or bankruptcies have resulted in attempts to gain a greater understanding of and forecast bankruptcy events

(Chen 2011). Predicting bankruptcies has been a well studied area by many researchers. Classical multivariate and discriminant analysis as well as stochastic models have been used in predicting bankruptcies. However, with statistical methods as such, the practical applications are limited due to firm presumptions such as of normality, linearity, independence among predictor variables and other assumptions (Hua et al. 2007).

Neural networks are also an interesting application to the prediction of bankruptcies. They have significant advantages over the usual statistical techniques. For example Odom and Sharda (1990) stated that their NN model had at least the same level of accuracy as discriminant analysis.

4.4.1 Variables

When building a NN for bankruptcy prediction the following financial ratios can be used as input:

- Profit ratio: the profit from sales and equity
- Current ratio: This ratio simply divides the current assets by the current liabilities. A higher current ratio indicates a healthier company. A ratio of < 1 is bad.
- Debt/Equity ratio: A key indicator of a company's ability to satisfy financial commitments as well as the nature of its funding, and whether it is mostly funded by equity holders or by debt financing. Generally, a ratio that's higher than 2 is considered unhealthy
- Cash flow to debt ratio: Calculated by dividing the cashflow from operations by the total debt. A higher ratio means a company is well equipped to cover its debt. A ratio greater than 1 is considered healthy and a ratio below 1 means the company might face bankruptcy in a few years.

Non-financial ratios can also be used for inputs to the neural network. For example:

- Price-to-book ratio
- Insider holding ratio
- Dividend payout ratio
- Company history and size

4.4.2 Model

Many variables can be used as input into a bankruptcy prediction model. Odom and Sharda (1990) only used 5 variables; working capital to assets ratio, RE/TA ratio, ROTA ratio, debt to equity ratio and sales to total assets ratio. A general NN that is used for bankruptcy prediction is shown below.

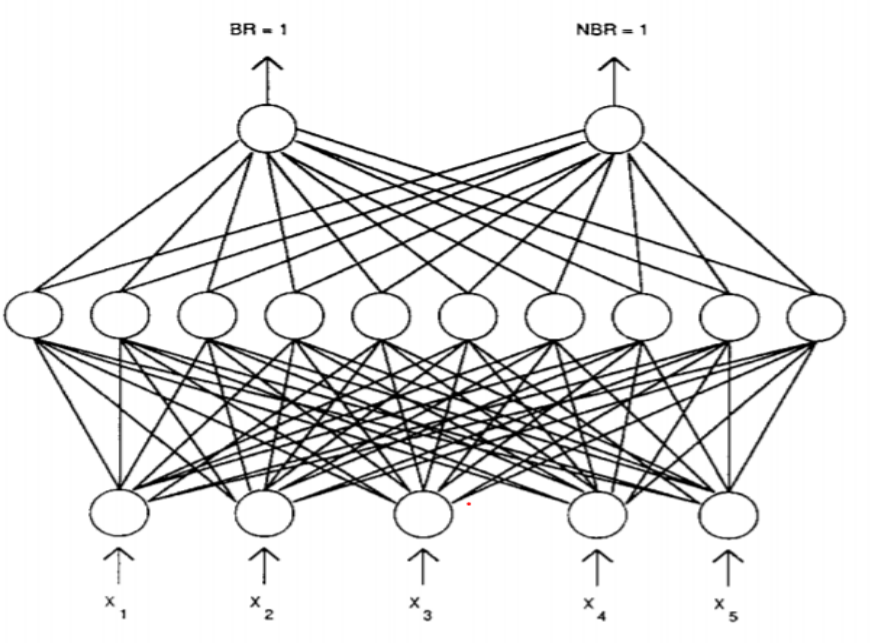

Figure 4.8: A typical neural network for bankruptcy prediction (Odom and Sharda 1990).

4.5 Neural Networks for Loan Application Evaluation

In this part, we will take an overview of credit risk. Habitually, when awarding credit, a lender provides goods and services to the borrower on the basis of the certainty that the borrower will repay the lender at some point in the future. In replacement for providing goods or services to the current borrower, the lender takes repayment benefit (*What Is Credit Risk? Why Is Credit Risk Important?* 2021). The most popular types of loans that lenders offer to borrowers are financial credit cards and homeownership loans. The borrower is obligated to pay off the corresponding amount to the bank with a bonus, and this is how the bank makes an interest. Another model of credit is asset financing that profitable banks provide to companies.

Asset financing is when a company uses a loan to obtain equipment. They usually have to fund a regular price to use it within an acceptable period of time rather than paying the full price of the equipment in advance (*What Is Credit Risk? Why Is Credit Risk Important?* 2021). The

likelihood that the borrower will default on his loan to the lender is called credit risk, in which case the lender will not receive the outstanding principal. In addition, they will not receive the interest owed and therefore incur a large loss. Also, the lender is likely to incur significant costs in trying to collect the outstanding debt. These fees are called collection costs, and the inability of the borrower to make the payments necessary to pay off his debts is called default. To protect against borrowers not repaying the debt, lenders must thoroughly evaluate the credit risk associated with each borrower. One way that lenders can reduce losses due to a borrower's default is to request collateral to cover the outstanding debt. Another approach is for the lender to raise the loan rate on the funds, which expresses the interest rate for high-credit risk borrowers. This is known as risk-based pricing.

When a bank receives a loan application, it must decide whether or not to approve the loan based on the applicant's profile. There are two kinds of risks the bank needs to take:

- If the borrower is likely to repay the loan, the company may lose business if the loan is not approved.
- If the applicant is not likely to repay the loan, i.e., he/she is likely to default, then approving the loan may lead to a financial loss for the company

Data and General Preprocessing:

When receiving a small amount of loan applications, it is easy for a person to evaluate the applicants profile. However, when receiving a huge number of applications, the company needs to evaluate the applicant's profile quickly and accurately. Here comes the power of machine learning and especially neural networks. Predicting if an applicant will default on a loan or not is a supervised learning task which means that we need historical data of applicants labeled as 'charged-off, the 'defaulters' and applicant labeled as 'Fully Paid'. The company trains a neural network model to predict future applicants. In general, the features can be grouped into three categories:

1. Features linked to the candidate (demographic variables such as profession, profession details, etc.).
2. Features linked to loan attributes (amount of loan, credit rate, the goal of the loan, etc.).
3. Features created by the Lender company experts.

In general, the data needs to be preprocessed before feeding it to the ANN model.

Artificial Neural Networks for Loan Evaluation

There a lot of techniques used for loan evaluation and ANNs are found to be the best of them. Regardless of which form of strategy the borrower chooses to reduce losses due to borrower defaults, the most important thing is to be able to accurately predict each borrower's credit risk. Lenders know that there is a certain amount of credit risk associated with every borrower. Expected loss comes as a result of different types of factors, borrower-specific factors, the economic environment, and the combination of the two. Using one type of feature is misleading and the interaction of the three types of features is very difficult to even for experts. Artificial Neural Networks (ANNs) are found to be very useful and accurate in predicting loan defaulters.

Now since this is a binary classification problem (we need to predict if a customer will default 1 or not 0) the ANN model will return a value between 0 and 1, which represents the probability of the customer defaulting. Experts need to set a threshold to decide if a customer will default or not, for example setting a threshold of 0.5 means that if the returned value is less the customer will not default and if it is high then the customer will default. Many of the best performing NN models have shown to achieve 76%- 93% accuracy in predicting if a customer will default or not (Dushimimana et al. 2020).

Example of Company using ANNs for Loan Evaluation

A good example of a loan company that use ANNs to predict loan defaulters is LendingClub. LendingClub is a US peer-to-peer lending company, headquartered in San Francisco, California. It was the first peer-to-peer lender to register its offerings as securities with the Securities and Exchange Commission (SEC), and to offer loan trading on a secondary market. LendingClub states that they use "machine learning technology to analyze risk and credit decisions" (*New AI-Driven Credit Model for LendingClub Spooks Investors* 2021) They have also made there data available from 2007 to 2018 and many machine learning practitioners such as Turiel and Aste (2019) have built NN models to predict loan defaulters.

Chapter 5

Methodology and Conclusions

5.1 Using LSTM to Predict USD-EUR Exchange Rate

We will use a Long-Short-Term-Memory NN to forecast the US Dollar and Euro exchange rate. 2 years of data (1/09/17 - 1/09/19) will be retrieved from the FRED data server.

The equations for a forward pass of an LSTM unit (with a forget gate) are given as

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$

$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

$$h_t = o_t \circ \sigma_h(c_t)$$

where $c_0 = 0$ and $h_0 = 0$ are the initial values and the operator $\circ$ stands for element-wise product and t is the time step.

i_t: Input gate activation vector

f_t: Forget gate

o_t: Output gate

h_t: Hidden state vector (or output vector of the LSTM unit)

x_t: Input vector to $\tilde{c}_t$: cell input activation vector

c_t: Cell state vector

W, U and b: weight matrices and bias vector parameters

5.1.1 Using MATLAB to Predict

Our methodology is split into 4 stages.

- Stage 1 - Obtain raw data: We retrieve raw data of the US-EUR exchange rate from the FRED data server and use the historical data to predict future rates.
- Stage 2 - Data preprocessing: This stage entails
  - Data reduction: We only use 2 years of data (Sep 2017 - Sep 2019)
  - Data cleaning: We replace missing values with values of the previous timestep
  - Data transformation: Standardization of data to turn it into a format the computer can learn from.
- Stage 3 - Feature extraction: We need to only use the features that will be fed into the neural network. This is the Date and the Close price.
- Stage 4 - Training the NN: Here we feed the data into the LSTM NN so training can take place for prediction by assigning random biases and weights. Our model is comprised of a sequece input layer followed by an LSTM layer and fully connected layer. The output layer is a regression layer with mean-squared error as the loss function.
- Stage 5 - Output: The output values generated from the output layer is compared with the true values and the error between these 2 values is minimized by iterations of the backpropagation algorithm. This algorithm adjusts the weights and biases of the network as discussed in Chapter 2

First we implement the network in MATLAB using the Deep Learning Toolbox and Datafeed toolbox.

The data from the FRED server will be split into 90% training data and 10% test data. Missing data will be replaced by data in the previous timestep.

The exchange rate data between Sept 2017 and Sept 2019 is shown below.

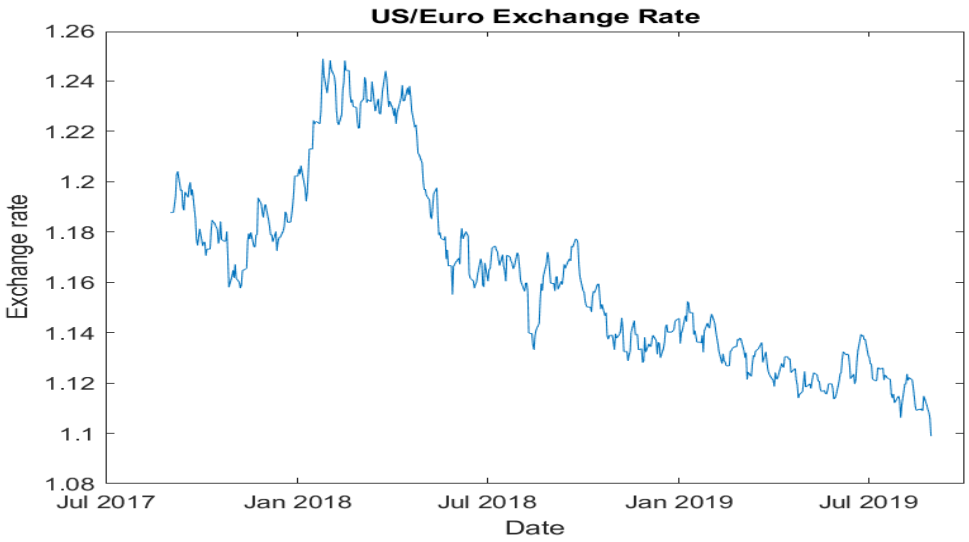

Figure 5.1: USD/EUR exchange rate (Sep 2017 - Sep 2019).

The graph shows a decreasing trend between September 2017 and September 2019. We will use the first 90% of the data in our neural network.

A MATLAB program A.1 incorporating the LSTM NN was used to predict the exchange rate. We used the deep learning toolbox to build the LSTM structure. Using this, we were easily able to generate the 5 lines of code for the LSTM network architecture.

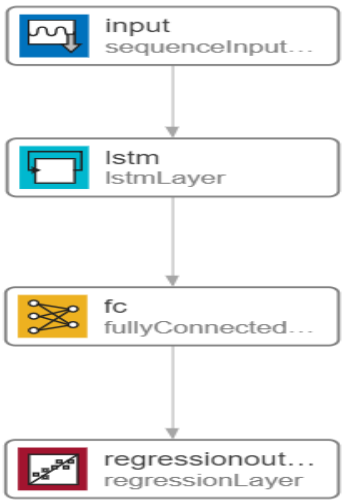

Figure 5.2: Designing the neural network using the deep network designer (part of MATLABs Deep Learning Toolbox). The number of hidden units for the LSTM layer was set to 200.

Listing 5.1: Code produced by exporting the LSTM design.

```
layers = [
```

```
    sequenceInputLayer(12,"Name","sequenceinput")
    lstmLayer(200,"Name","lstm")
    fullyConnectedLayer(9,"Name","fc")
    regressionLayer("Name","regressionoutput")];
```

This code was then modified by instead having our custom number of features (instead of 12 for Input layer we passed in a variable 'numFeatures' and instead of 9 for FC layer we passed in the variable, 'numResponses').

The modified code is shown below.

Listing 5.2: Modified code.

```
numFeatures = 1;
numResponses = 1;
numHiddenUnits = 200;

layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits)
    fullyConnectedLayer(numResponses)
    regressionLayer];
```

The training results obtained from training the neural network with 250 iterations are shown in 5.3. The root-mean-squared-error is given as:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_{pred} - y_{test})^2}{n}} \tag{5.1}$$

where y_{pred} are the predicted vaues and y_{test} are the observed (true) values.

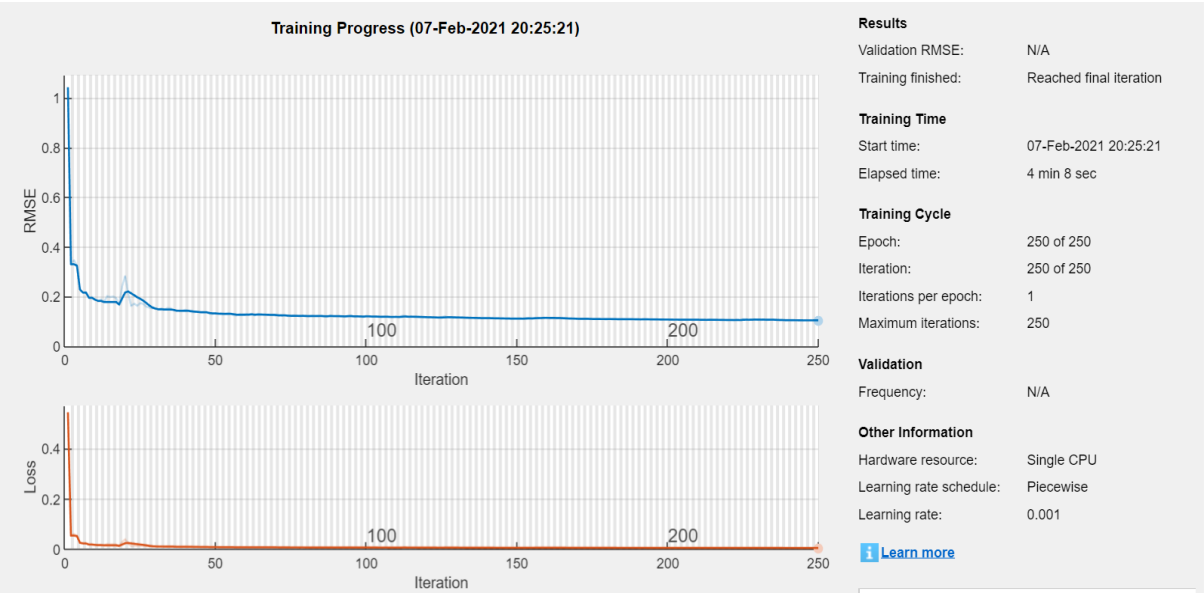

Figure 5.3: Training results with 250 iterations.

The RMSE quickly decreases as the number of iterations increase upto around 40 iterations where the RMSE starts to flatten out. The loss (the difference between the predicted output and the actual output) also decreases and stays constant after $\sim$ 35 iterations.

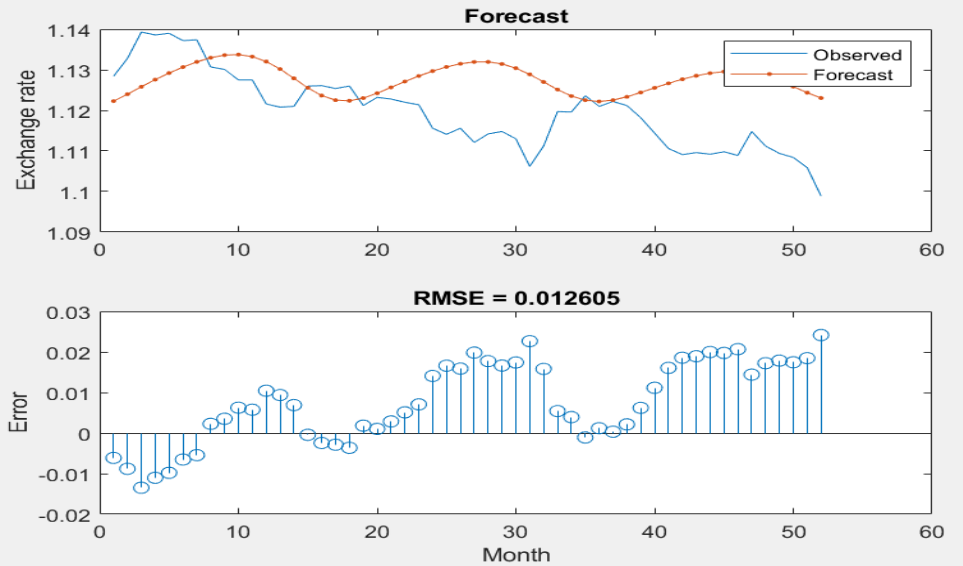

Figure 5.4: The forecast and the RMSE given as a chart.

We can observe that the forecast is not very accurate and the trend is smoothed out rather than rugged like the actual values. The root-mean-squared error is 0.012605

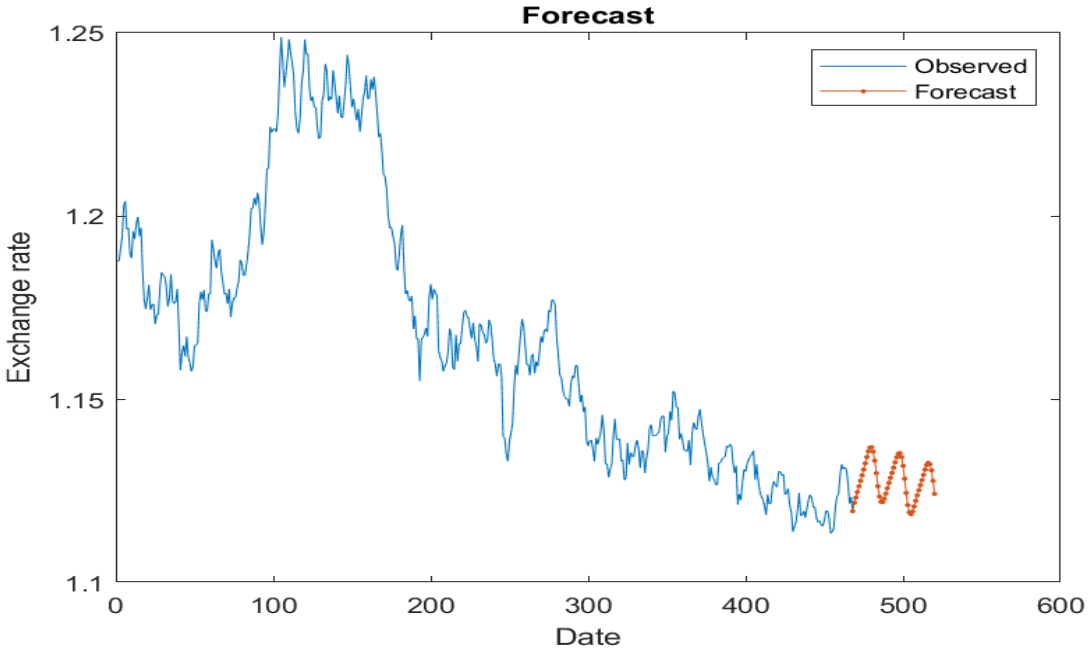

Figure 5.5: The exchange rate forecast.

From Figure 5.5 we can see that MATLAB forecasts a general trend rather than accurate

predictions. We visually compare the actual and predicted graphs below.

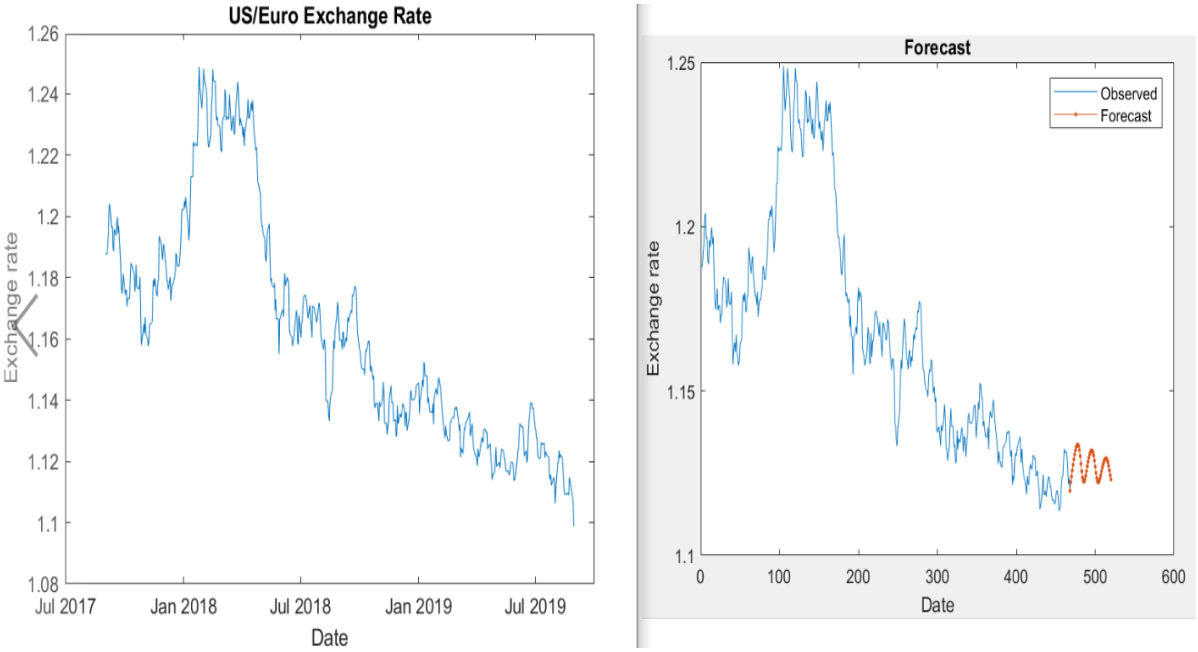

Figure 5.6: The actual values compared with the predicted values.

350 iterations

MATLAB seems to only predict the direction of the trend. We now increase the number of iterations to 350 to see if there is an improvement.

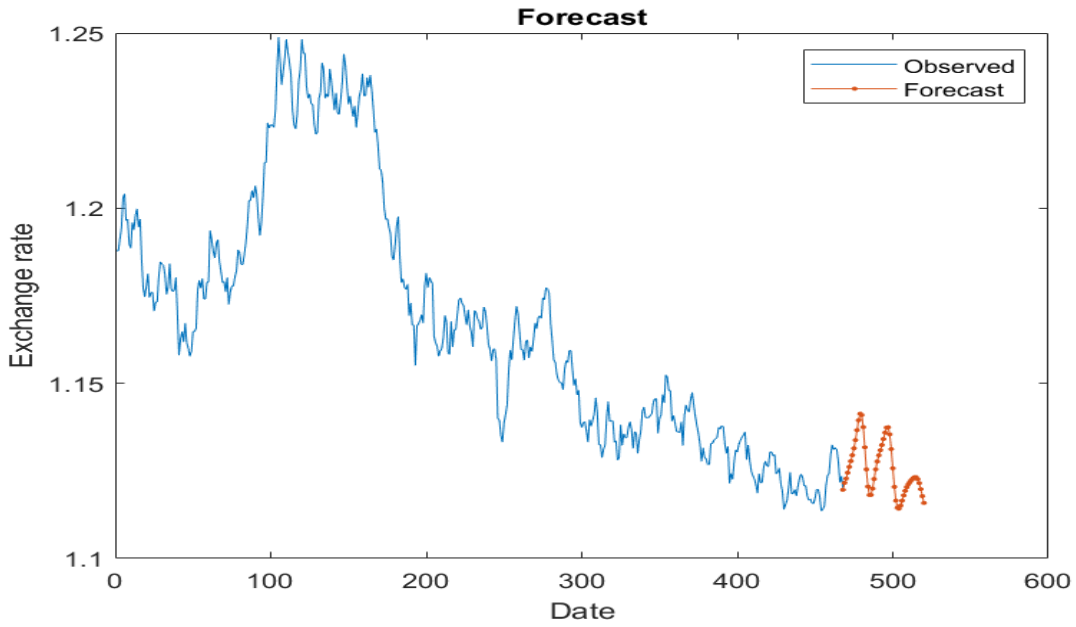

Figure 5.7: US-EUR forecast using 350 iterations.

The forecast seems to be better with a higher number of iterations

5.1.2 Using Python to Predict

We now use a Python program to predict the US-EUR exchange rate. We train on the same 2 years of data however, we split it into 80% training data and 20% testing data. The program A.3 was used with the same neural network (LSTM) as with the MATLAB program.

Table 5.1: Raw data.

Date	FX rate
2017-09-01	1.1878
2017-09-04	NaN
2017-09-05	1.1911
2017-09-06	1.1943
2017-09-07	1.2028
...	...
2019-08-26	1.1112
2019-08-27	1.1094
2019-08-28	1.1084
2019-08-29	1.1059
2019-08-30	1.0989

The data above is not ready for use as it has missing values ('NaN'). We clean this data by using Pythons 'fillna' function and replace each missing value with the previous value.

After training our model with 250 iterations we get the Loss graph as:

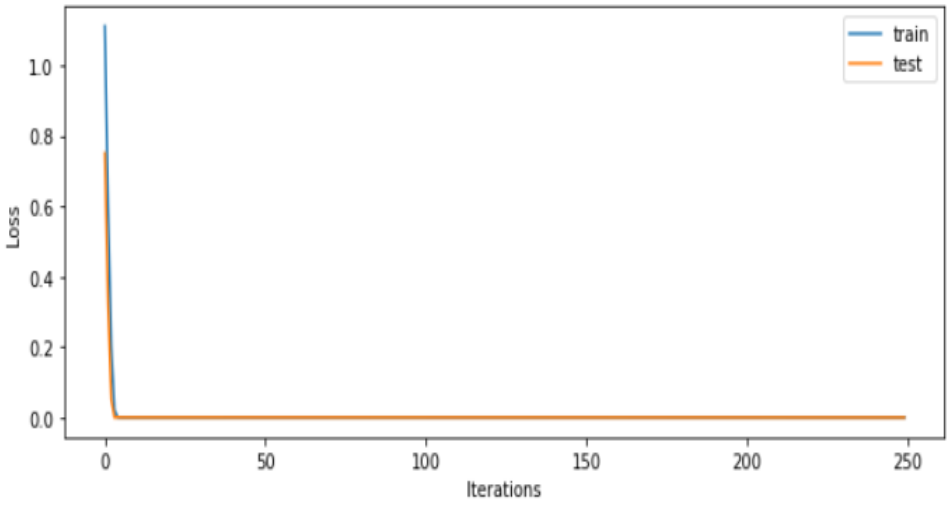

Figure 5.8: Loss over 250 iterations.

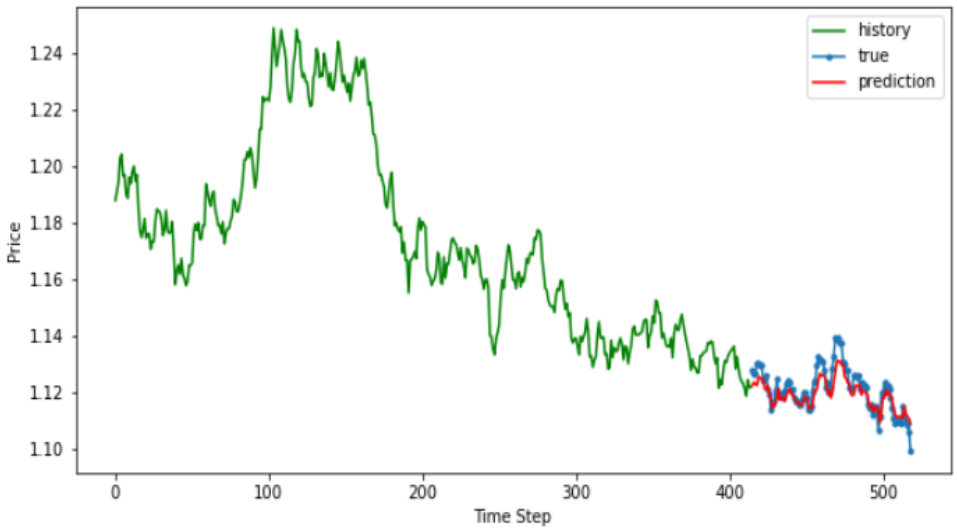

Figure 5.9: US-EUR prediction for the last 10% of data with 250 iterations.

The predictions with Python turn out to be quite accurate as we can see in the graph and we achieve a root mean-squared-error of 0.00918 compared to the 0.01261 RMSE obtained from the MATLAB model.

The true and predicted values are shown in tabular form below.

Table 5.2: Actual and predicted values of the exchange rate.

	True	Predicted
0	1.1278	1.125076
1	1.1266	1.126132
2	1.1262	1.125340
3	1.1304	1.125076
4	1.1304	1.127848
...	...	...
99	1.1112	1.117555
100	1.1094	1.115181
101	1.1084	1.113995
102	1.1059	1.113336
103	1.0989	1.111688

We now see how our Python model predicts the exchange rate with 350 iterations. All other variables were kept the same.

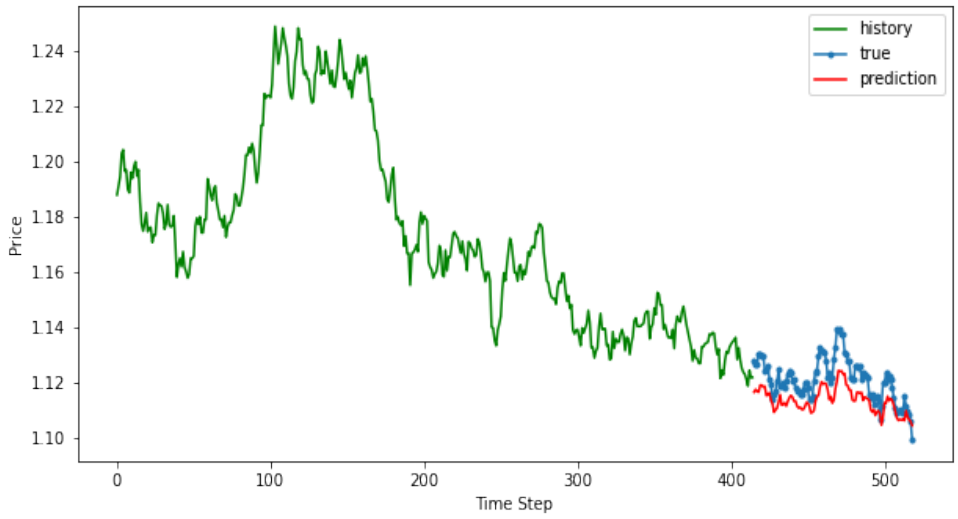

Figure 5.10: US-EUR prediction for the last 10% of data with 350 iterations.

In Python, using more iterations seems to make the predictions diverge from the true values.

A Concern Within the Model

There is however, a problem in our model. We have not scaled our training and test data which means it will have trouble in predicting when the input values have a large range for example between 100 - 300. These large inputs can slow down the convergence of the network and can also stop the network from effectively learning the problem. Standardizing the data is a way to scale large values. It involves rescaling the value distribution such that the measured mean is 0 and the standard deviation is 1.

5.2 Using LSTM to Predict Stock Price

Lets take Facebook stock for example between 2017 and 2020. Its stock price ranges from approximately 120 to 220 during this period. Using an unstandardized LSTM model like what was done for the exchange rate predictions (with Python) will give us obscure predictions as shown below:

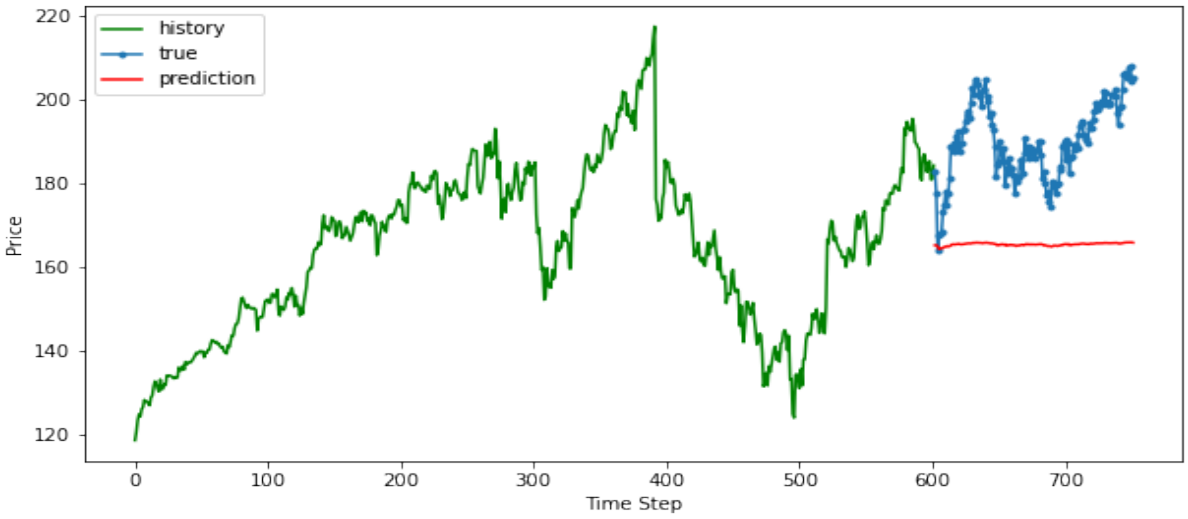

Figure 5.11: Faulty Facebook stock price predictions due to using unstandardized data.

This is a terrible prediction - as expected due to the values being large and the data not being standardized.

We can standardize our training and test data by using the 'StandardScaler' function from Pythons sklearn toolkit.

Listing 5.3: Code for standardizing the training data and transforming the test data.

```
# Scaling and standardizing the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.reshape(-1, 1))

#Using the scaler to transform the test data
X_test_scaled= scaler.transform(X_test.reshape(-1, 1))
```

Now that we have standardized the data we can try and predict the Facebook stock price (using data from 2017 to 2020) again. The full Python program can be found at Listing A.4

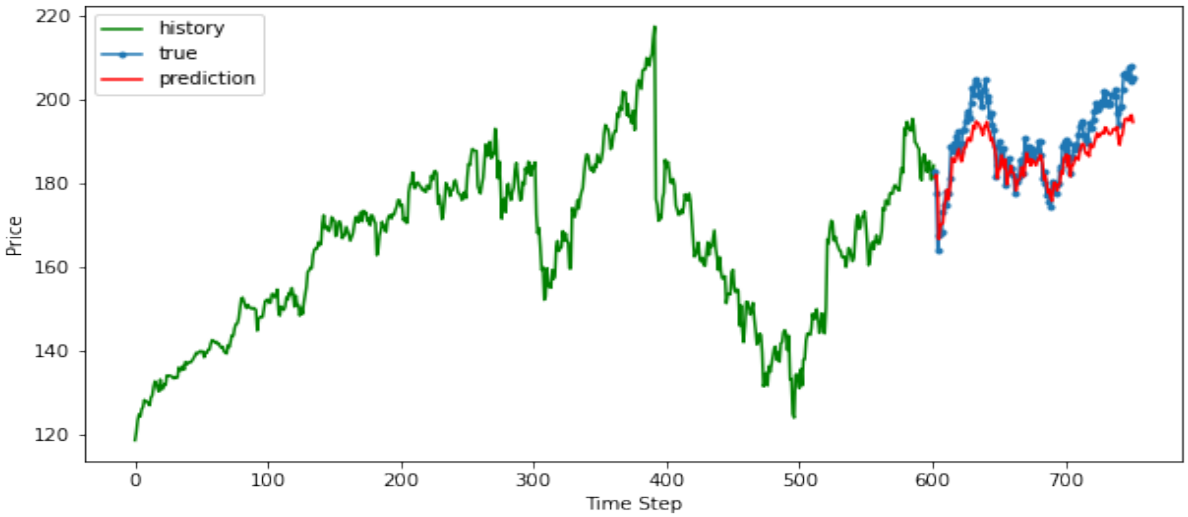

Figure 5.12: Facebook stock price prediction between 2017-01-01 to 2020-01-01.

This is a more sensible prediction with an RMSE of 11.6657 and the loss graph converges at around 130 iterations which seems reasonable (the validation loss is slightly higher than the train loss) unlike Figure 5.9 where the model seems to be overfitted.

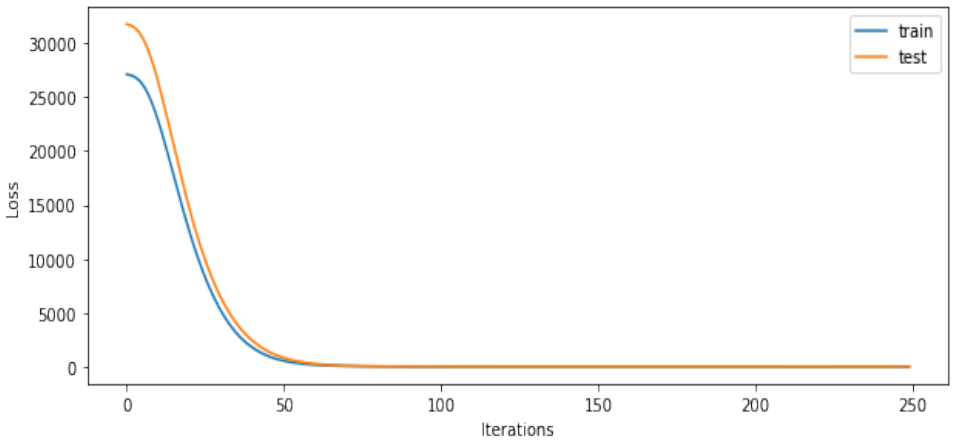

Figure 5.13: Loss over 250 iterations for Facebook stock price prediction.

Table 5.3: Actual and predicted values for the stock price.

	True	Predicted
0	183.01	181.52
1	177.47	182.10
2	164.15	178.04
3	167.50	166.68
4	168.17	169.76
...	...	...
99	205.12	195.46
100	207.79	194.96
101	208.10	196.20
102	204.41	196.34
103	205.25	194.62

5.3 Conclusion

5.3.1 Summary

The ability to predict future outcomes of market movements using neural networks is an active field of research. The work presented in this project explores the history of neural networks and financial maths as well as the applications. We then went on to detail what neural networks are and the types of neural networks in Chapter 2. We also cover the problems that occur when using particular NNs such as exploding and vanishing gradients. In Chapter 3 we investigated financial mathematics and time series. Financial derivatives were a big part of financial math and we went through the types as well as the advantages and disadvantages. There are many areas in financial maths and not all could be covered. Examples of time series plots were plotted with Python using example datasets from online.

Next, we studied the applications of neural networks to financial maths. A comparison was made between using Multi-layer perceptrons and LSTM on predicting stock price. We also showed how we can price financial derivatives using NNs in addition to bankruptcy prediction and loan application evaluation. Overall NNs do help a lot of companies in predicting financial matters.

There were many types of neural networks to use but for our methodology we constructed and developed a LSTM NN to predict the US-EU exchange rate. First we used MATLAB and managed to obtain predictions with an RMSE of 0.012605 with 250 iterations. We then increased the number of iterations to 350 and found the predictions to be a bit better. We also

implemented the LSTM network in Python which gave us more accurate predictions with an RMSE of 0.000918.

5.3.2 Challenges and Successes

Problems arose when we tried to use the Python LSTM network to predict larger values e.g Facebook's stock price. Here we had not standardized the training and test data (and so the mean was not zero and variance wasn't one) which caused the predictions on high range data, e.g 120-220 to fail.

However, we countered this by standardizing our data and hence achieved suitable stock price predictions as shown in Section 5.2. Increasing the number of epochs to 350 would only decrease the RMSE slightly and so the compromise of time for minimally better accuracy was not worth it.

With MATLAB, data had been standardized but we were unable to predict stock prices as the datafeed toolbox had trouble connecting to yahoo finance. We used a custom function built by the MATLAB community: 'getMarketDataViaYahoo',

Listing 5.4: Function built by community.

```
data = getMarketDataViaYahoo('FB', '01-Jan-2017', '01-Jan-2019', '1d');
```

however the model did not properly predict the stock price and so was omitted from the report.

Another challenge that we needed to regulate was the model overfitting. This is where the model replicates the training data too closely. If we have too many epochs, our model would overfit and so optimizing the number of epochs for the model was essential in producing accurate results. We found an optimal number of epochs to be 250.

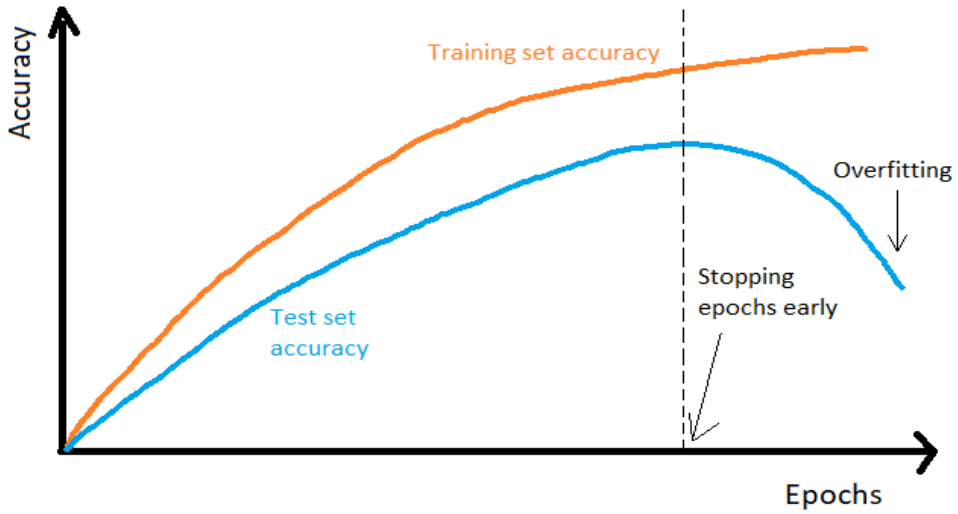

Figure 5.14: Training and test set accuracy against the number of epochs.

Figure 5.14 neatly summarises how increasing the number of epochs can improve the accuracy of the training set but not the test set. The optimal amount of epochs can be found by locating the test sets peak accuracy. With more epochs after the optimal, the model would start to overfit.

For the models that uses the FRED server (which is very limited), only data from that server can be used for predictions. The model that uses Yahoo Finance can be applied to not only Facebook, but any stock and index price as well as exchange rate data by just changing the ticker symbol and dates in the first line of code. Hence, by creating a reproducible model it allows people who don't know how to code to use it to predict any economic related time series prices such as the Facebook stock price, SP500 index or an exchange rate.

5.3.3 Future work and Recommendations

If more time was allowed, we could have seen the effect of changing the batch size in the network. The batch size is the number of training examples in one epoch. We used a batch size of 16, but it would be interesting to see how the model training time increases or decreases with changing the batch sizes.

Further work on the model can be carried out by using the High/Low prices of the day or by using the volume of shares traded on each day.

Date	Open	High	Low	Close	Adj Close	Volume
2017-01-03	116.029999	117.839996	115.510002	116.860001	116.860001	20663900
2017-01-04	117.550003	119.660004	117.290001	118.690002	118.690002	19630900
2017-01-05	118.860001	120.949997	118.320000	120.669998	120.669998	19492200
2017-01-06	120.980003	123.879997	120.029999	123.410004	123.410004	28545300
2017-01-09	123.550003	125.430000	123.040001	124.900002	124.900002	22880400

Figure 5.15: The opening, high, low and closing price for each day along with the Volume for Facebook stock for 5 days in 2017

As we can see a financial asset traded in a market can have an opening price, high and low prices as well as a closing price and volume. We only used the closing price in our model. To improve accuracy in predictions more factors like these should be incorporated into the model.

The number of layers for the LSTM could also be increased to study how it affects the accuracy. However, researchers must be aware that increasing the number of layers will increase the time it takes to train the model. Additionally, other types of neural networks (for example, MLPs and CNNs) can be used to compare against the LSTM network and thereby the best performing network can be chosen. This network can then further be optimized to obtain the lowest errors possible.

Additionally, an extended financial time series can be used to train on. In our model, we only trained the model on 2-3 years of data. To improve the training performance and accuracy, more years of data can be used for training the LSTM and other networks.

Overall, this report was a thorough introduction to the applications of neural networks to financial maths. We must keep in mind that although neural networks are a powerful tool in quantitative finance, it still does not mean that the models will be good enough to predict for example stock prices, as their just just too much noise in such data. At best, these networks can give traders a bit of an edge against others in the market - be that 0.3% or 1%. In addition, since the price of a stock/exchange rate is not determined by the past but by factors in the economy ideally traders and investment firms would need access to large amounts of good clean data . This would include not just price, but variables such as related stocks/currencies social media

posts, news, fundamentals and even weather data! This, in combination with a sophisticated NN model would certainly help in increasing profitability in financial markets.

References

A Concise History of Neural Networks (2016). Towards Data Science. URL: https://towardsdatascience.com/a-concise-history-of-neural-networks-2070655d3fec (visited on 10/02/2020).

A Gentle Introduction to Exponential Smoothing (2021). Machine Learning Mastery. URL: https://machinelearningmastery.com/exponential-smoothing-for-time-series-forecasting-in-python/ (visited on 04/07/2021).

Air Passengers Dataset (2020). Kaggle. URL: https://www.kaggle.com/rakannimer/air-passengers (visited on 04/03/2020).

Anders, Ulrich, Korn, Olaf, and Schmitt, Christian (1996). "Improving the pricing of options: a neural network approach". In: 96-04.

Arévalo, Andrés, Nino, Jaime, Hernandez, German, Sandoval, Javier, León, Diego, and Aragón, Arbey (Aug. 2017). "Algorithmic Trading Using Deep Neural Networks on High Frequency Data". In: pp. 144–155. ISBN: 978-3-319-66962-5. DOI: 10.1007/978-3-319-66963-2_14.

Bachelier, L. (Jan. 2011). "Louis Bachelier's theory of speculation: The origins of modern finance". In: *Louis Bachelier's Theory of Speculation: The Origins of Modern Finance*, pp. 1–188.

Barrier option (2020). Wikipedia. URL: https://en.wikipedia.org/wiki/Barrier_option (visited on 11/16/2020).

Bennell, Julia and Sutcliffe, Charles (Oct. 2004). "Black-Scholes Versus Neural Networks in Pricing FTSE 100 Options". In: *University of Southampton - Department of Accounting and Management Science, Papers* 12.

Black-Scholes/Merton 40 (2020). MIT Sloan. URL: http://bsm40.mit.edu/ (visited on 11/20/2020).

Black, Fishcer and Scholes, Myron (1973). "The pricing of options and corporate liabilities". In: *Journal of political economy* 81.3, p. 637.

Brief History of Neural Networks (2019). medium. URL: https://medium.com/analytics-vidhya/brief-history-of-neural-networks-44c2bf72eec.

Chen, Mu-Yen (2011). "Bankruptcy prediction in firms with statistical and intelligent techniques and a comparison of evolutionary computation approaches". In: *Computers Mathematics with Applications* 62.12, pp. 4514–4524.

Chiang, W. -C., Urban, T. L., and Baldridge, G. W. (1996). "A neural network approach to mutual fund net asset value forecasting". In: *Omega* 24.2, pp. 205–215.

D. Rumelhart G E. Hinton, R. J. Williams (1986). "Learning representations by back-propagating errors". In: *Nature* 1.323, pp. 533–536.

Derivatives (2021). Corporate Finance Institute. URL: https://corporatefinanceinstitute.com/resources/knowledge/trading-investing/derivatives/ (visited on 03/20/2021).

Derivatives, With Their Risks and Rewards (2020). the balance. URL: https://www.thebalance.com/what-are-derivatives-3305833 (visited on 04/05/2020).

Dunbar, Steven R. (n.d.). *Stochastic Processes and Advanced Mathematical Finance.*

Dushimimana, Bernard, Wambui, Yvonne, Lubega, T., and McSharry, P. (2020). "Use of Machine Learning Techniques to Create a Credit Score Model for Airtime Loans". In:

Fischer Black (2020). Wikipedia. URL: https://en.wikipedia.org/wiki/Fischer_Black (visited on 11/20/2020).

Forward Contracts vs. Futures Contracts (Jan. 18, 2020). Investopedia. URL: https://www.investopedia.com/ask/answers/06/forwardsandfutures.asp (visited on 04/07/2021).

Frank Rosenblatt (n.d.). machinelearningknowledge. URL: https://machinelearningknowledge.ai/timeline/perceptron-neuron-that-can-learn/.

Frank Rosenblatt (n.d.). Cornell University. URL: https://ecommons.cornell.edu/bitstream/handle/1813/18965/Rosenblatt_Frank_1971.pdf;jsessionid=806241AAE028B29DCB2A5956F65A99B4?sequence=2.

Fresnillo Share Price (2021). Yahoo Finance. URL: https://finance.yahoo.com/quote/FRES.L/chart?p=FRES.L (visited on 04/07/2021).

Gao, Penglei, Zhang, Rui, and Yang, Xi (Aug. 2020). "The Application of Stock Index Price Prediction with Neural Network". In: *Mathematical and Computational Applications* 25, p. 53.

Garliauskas, A. (1999). "Neural network chaos and computational algorithms of forecast in finance". In: *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems* 2, pp. 638–643.

Halbert White (2012). American University Washington. URL: https://www.american.edu/cas/economics/info-metrics/white-memorium.cfm (visited on 11/24/2012).

Harrison, J.Michael and Pliska, Stanley R. (1981). "Martingales and stochastic integrals in the theory of continuous trading". In: *Stochastic Processes and their Applications* 11.3, pp. 215–260.

Hinton, G. E., Osindero, S., and Teh, Y. W. (2006). "A Fast Learning Algorithm for Deep Belief Nets". In: *Neural Computation* 18, pp. 1527–1554.

Hua, Z., Wang, Y., Xu, X., Zhang, B., and Liang, L. (2007). "Predicting corporate financial distress based on integration of support vector machine and logistic regression". In: *Expert Syst. Appl.* 33, pp. 434–440.

Hussain, Raamis (Aug. 2020). "Predicting Stock Prices using Multi-Layer Perceptron". In: p. 53.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (2020). ImageNet. URL: http://www.image-net.org/challenges/LSVRC/ (visited on 11/10/2020).

Jarrow, R. and Protter, P. (2004). "A short history of stochastic integration and mathematical finance the early years, 1880-1970". In: *IMS Lecture Notes* 45, pp. 75–91.

Jarrow, Robert A. and Oldfield, George S. (1981). "Forward contracts and futures contracts". In: *Journal of Financial Economics* 9.4, pp. 373–382.

Kim S.H., S.H. Chun (1998). "Graded Forecasting using an Array of Bipolar Predictions: Application of Probabilistic Neural Networks to a Stock Market Index". In: *INTERNATIONAL JOURNAL OF FORECASTING* 14.3, pp. 323–337.

Koenecke, Allison (n.d.). "Applying Deep Neural Networks to Financial Time Series Forecasting". In: *Stanford University* ().

Koenecke, Allison and Gajewar, Amita (2020). "Curriculum Learning in Deep Neural Networks for Financial Forecasting". In: *Lecture Notes in Computer Science*, pp. 16–31. ISSN: 1611-3349.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., pp. 1097–1105.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (Winter 1989). "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4, pp. 541–551.

Louis Bachelier (2020). People pill. URL: `https://peoplepill.com/people/louis-bachelier/` (visited on 11/11/2020).

MacTutor - Louis Bachelier (2020). MacTutor. URL: `https://mathshistory.st-andrews.ac.uk/Biographies/Bachelier/` (visited on 11/12/2020).

Malliaris, Mary and Salchenberger, Linda (Sept. 1993). "A neural network model for estimating option prices". In: *Appl. Intell.* 3, pp. 193–206. DOI: `10.1007/BF00871937`.

Marketwatch photo illustration/istockphoto (2020). Market Watch. URL: `www.marketwatch.com/story/these-5-giant-stocks-are-driving-the-us-market-now-but-bigger-is-not-always-better-2020-06-26` (visited on 04/05/2020).

Merton, Robert C. (1976). "Option pricing when underlying stock returns are discontinuous". In: *Journal of Financial Economics* 3.1, pp. 125–144.

New AI-Driven Credit Model for LendingClub Spooks Investors (2021). Bank Automation News. URL: https://bankautomationnews.com/allposts/payments/new-ai-driven-credit-model-for-lending-club-spooks-investors/ (visited on 04/13/2021).

New One Family Houses Sold (2021). FRED Economic Data. URL: https://fred.stlouisfed.org/series/HSN1F (visited on 04/03/2021).

Odom, M. D. and Sharda, R. (1990). "A neural network model for bankruptcy prediction". In: *1990 IJCNN International Joint Conference on Neural Networks*. Vol. 2, pp. 163–168.

Özel Kadilar, Gamze (Jan. 2015). "Stochastic Processes for the Risk Management". In:

Pound Dollar Exchange Rate (GBP USD) - Historical Chart (2020). Macrotrends. URL: https://www.macrotrends.net/2549/pound-dollar-exchange-rate-historical-chart (visited on 11/17/2020).

Prize in Economic Sciences 1997 (n.d.). The Royal Swedish Academy of Sciences. URL: https://www.nobelprize.org/prizes/economic-sciences/1997/press-release/.

Sardi, Shira, Vardi, Roni, Sheinin, Anton, Goldental, Amir, and Kanter, Ido (Dec. 2017). "New Types of Experiments Reveal that a Neuron Functions as Multiple Independent Threshold Units". In: *Scientific Reports* 7.

Schmidhuber, Hochreiter (1997). "LONG SHORT-TERM MEMORY". In: *Neural Computation* 9, pp. 1735–1780.

Stankovska, Aleksandra (Jan. 2016). "Global Derivatives Market". In: *SEEU Review* 12.

The Cybernetics Thought Collective: A History of Science and Technology Portal Project (2014). University of Illinois. URL: https://archives.library.illinois.edu/thought-collective/cyberneticians/warren-s-mcculloch/ (visited on 09/28/2020).

The Vanishing Gradient Problem (2020). Towards Data Science. URL: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484 (visited on 11/10/2020).

Turchenko, Volodymyr, Beraldi, Patrizia, De Simone, Francesco, and Grandinetti, Lucio (Sept. 2011). "Short-term stock price prediction using MLP in moving simulation mode". In: pp. 666–671.

Turiel, Jeremy D. and Aste, Tomaso (2019). *P2P Loan acceptance and default prediction with Artificial Intelligence*.

United Kingdom Inflation Rate (2020). Trading Economics. URL: tradingeconomics.com/united-kingdom/inflation-cpi (visited on 04/05/2020).

Using AI to Predict Facebook's Stock Price (Mar. 5, 2019). GoodAudience. URL: https://blog.goodaudience.com/using-ai-to-predict-ais-stock-price-e6b574e06afa (visited on 11/24/2012).

Walter Pitts (2020). Wikipedia. URL: https://en.wikipedia.org/wiki/Walter_Pitts (visited on 09/24/2020).

What Is Credit Risk? Why Is Credit Risk Important? (2021). 365 FinancialAnalyst. URL: https://365financialanalyst.com/knowledge-hub/credit-analysis/what-is-credit-risk-why-is-credit-risk-important (visited on 04/12/2021).

White, H. (1988). "Economic prediction using neural networks: the case of IBM daily stock returns". In: *IEEE 1988 International Conference on Neural Networks*, 451–458 vol.2.

Wikipedia- Artificial Neuron (2020). Wikipedia. URL: https://en.wikipedia.org/wiki/Artificial_neuron (visited on 09/24/2020).

Yahoo Finance (2021). Yahoo Finance. URL: https://uk.finance.yahoo.com/quote/GOOG/ (visited on 04/03/2021).

Yao, Jingtao, Li, Yili, and Tan, Chew Lim (2000). "Option price forecasting using neural networks". In: *Omega* 28.4, pp. 455–466.

Yao, Jingtao and Tan, Chew Lim (Sept. 2000). "A case study on using neural networks to perform technical forecasting of forex". In: *Neurocomputing* 34, pp. 79–98.

Yoon, Youngohc, Jr., George Swales, and Margavio, Thomas M. (1993). "A Comparison of Discriminant Analysis versus Artificial Neural Networks". In: *Journal of the Operational Research Society* 44.1, pp. 51–60.

Appendix A

Programs

A.1 MATLAB code

Listing A.1: A MATLAB program to predict the US-EUR exchange rate using a LSTM neural network.

```
% We'll first connect to the FRED data server using the url:
url = 'https://fred.stlouisfed.org/';
c = fred(url);
%%
% Adjust the display data format for currency.
format bank
%%
% Retrieve historical data for the US/EUR exchange rate
% series.
series = 'DEXUSEU';
%%
% Fetch 2 years of data from Sept 1, 2017 through Sept 1, 2019.
startdate = '09/01/2017';
enddate = '09/01/2019';
d_input = fetch(c,series,startdate,enddate)
Input= (d_input.Data)
% replace missing data (N/A) values with data at previous timestep
clean_input= fillmissing(Input,'previous');

DATA=clean_input
% converting the dates from datenum format to datetime format
TimeDT = datetime(DATA(:,1), 'ConvertFrom', 'datenum', 'Format', 'yyyy-MM-dd');
% visualizing our data
figure
plot(TimeDT,DATA(:,2))
xlabel("Date")
ylabel("Exchange rate")
title("US/Euro Exchange Rate")

%%
% Partition the training and test data. Train on the first 80% of the sequence
% and test on the last 20%.
```

```
% We'll make an array pricedata which only has the prices (exchange rates).
% We also transpose this, to make a 1xN array
pricedata= (DATA(:,2))'
% splitting the data
nTimeStepsTrain = floor(0.9*numel(pricedata));

trainData = pricedata(1:nTimeStepsTrain+1);
testData = pricedata(nTimeStepsTrain+1:end);

%% Standardize Data
%We'll standardise the training data for a better fit and to
%prevent the training from diverging.

mu = mean(trainData);
sig = std(trainData);

trainDataStandardized = (trainData - mu) / sig;

%% Prepare Predictors and Responses
% For forecasting values of future timesteps we describe the responses to
% be the training sequences with values moved by one time step
X_train = trainDataStandardized(1:end-1);
Y_train = trainDataStandardized(2:end);
%% *Here, we'll define the LSTM Network Architecture*
% Make a regression network using the LSTM algorithm.
% Specify the LSTM layer to have 200 hidden units.

numFeatures = 1;
numResponses = 1;
numHiddenUnits = 200;

layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits)
    fullyConnectedLayer(numResponses)
    regressionLayer];
%%

% Specifying the training options. Solver is set to 'adam'. Gradient
% threshold is set to 1 (to prevent gradients from exploding). Initial
% learn rate is 0.005 which drops after 125 iterations by multiplying with a
    factor of 0.2

options = trainingOptions('adam', ...
    'MaxEpochs',250, ...
    'GradientThreshold',1, ...
    'InitialLearnRate',0.005, ...
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropPeriod',125, ...
    'LearnRateDropFactor',0.2, ...
    'Verbose',0, ...
    'Plots','training-progress');
%% Train LSTM Network
% We now train the network using the specified training options by using
    trainNetwork

network = trainNetwork(X_train,Y_train,layers,options);
%% Forecasting Future Time Steps
% We'll use the predictAndUpdateState function to estimate values of
% multiple time steps in the future. At each prediction, this function updates
    the network state.
% We'll use the previous prediction as the input to the function for each
    prediction.
```

```
% Standardizing the test data using the same parameters as with the
% training data.

testDataStandardized = (testData - mu) / sig;
X_test = testDataStandardized(1:end-1);
%%
% We first predict on the training data |X train| to initialise the network
    state.
network = predictAndUpdateState(network,X_train);
% Make the first prediction using the last time step of the training response |
    Y_train(end)|
[network,Y_pred] = predictAndUpdateState(network,Y_train(end));

% Loop over remaining predictions and input the previous prediction
% to |predictAndUpdateState|.
nTimeStepsTest = numel(X_test);
for i = 2:nTimeStepsTest
    [network,Y_pred(:,i)] = predictAndUpdateState(network,Y_pred(:,i-1),'
    ExecutionEnvironment','cpu');
end

%%
% Unstandardizing the predictions using the paramteres calculated from before
Y_pred = sig*Y_pred + mu;
%%
% The training progress plot reports the root-mean-square error (RMSE)
    calculated
% from the standardized data. RMSE will be calculated from the unstandardized
% predictions.

Y_test = testData(2:end);
rmse = sqrt(mean((Y_pred-Y_test).^2))
%%
% Plot the training time series with the forecasted values.

figure
plot(trainData(1:end-1))
hold on
idx = nTimeStepsTrain:(nTimeStepsTrain+nTimeStepsTest);
plot(idx,[pricedata(nTimeStepsTrain) Y_pred],'.-')
hold off
xlabel("Date")
ylabel("Exchange rate")
title("Forecast")
legend(["Observed" "Forecast"],'Location','northeast')
%%
% Compare the forecasted values with the test data.

figure
subplot(2,1,1)
plot(Y_test)
hold on
plot(Y_pred,'.-')
hold off
legend(["Observed" "Forecast"])
ylabel("Exchange rate")
title("Forecast")

subplot(2,1,2)
stem(Y_pred - Y_test)
xlabel("Month")
ylabel("Error")
```

```
title("RMSE = " + rmse)
```

Listing A.2: Same as Listing A.1 but initial data is Facebook data from Yahoo Finance

```
    % We'll first connect to Yahoo Finance and retreive Facebook stock data
% from 1st Jan 2016 to 1st Jan 2018 using a custom function
% getMarketDataViaYahoo
data = getMarketDataViaYahoo('FB', '01-Jan-2016', '01-Jan-2018', '1d');

%%

% Extracting date from the table and converting datetime dates to datenum
% format. This is so that we can put the data into a nx2 array
DateNum= datenum(data.Date);

% Making an array with the Date and Close (as we only want the closing price of
%the stock for each day) column of the table. We use .Close to extract the
% close price
Input= [DateNum data.Close]

% replace missing data (N/A) values with data at previous timestep
clean_input= fillmissing(Input,'previous');

DATA=clean_input
% converting the dates from datenum format back to datetime format (to plot
% easily)
TimeDT = datetime(DATA(:,1), 'ConvertFrom', 'datenum', 'Format', 'yyyy-MM-dd');
% visualizing our data
figure
plot(DATA(:,1),DATA(:,2))
xlabel("Date")
ylabel("Price")
title("Facebook Stock Price")
```

A.2 Python code

Listing A.3: Python program to predict US-EUR exchange rate using a LSTM neural network.

```
# Importing the relevant libraries and API
import pandas as pd
pd.options.display.max_colwidth = 60

import tensorflow as tf
from tensorflow import keras
import numpy as np
from numpy import mean
from fredapi import Fred #importing the FRED API
import numpy as np
import math
import statistics
import pylab


%matplotlib inline
import matplotlib.pyplot as plt
from IPython.core.pylabtools import figsize
```

```
figsize(20, 5)

fred = Fred(api_key='419401c4103800a51c50e6113b1f7500')
# Retrieve historical data for the US/EUR exchange rate
# series between Sept 2017 and Sept 2019
data = fred.get_series('DEXUSEU',observation_start='2017-09-01',
    observation_end='2019-09-01',name='Price')
# We use data.tail() to view the first 5 values from the dataset
data.head()

#Converting the data series into a dataframe (so we can replace missing values
    easily)
Input= data.to_frame()

# Cleaning the input by replacing missing values with the previous value
clean_input= Input.fillna(method='ffill')
DATA = clean_input
# Renaming the price column from '0' to 'Price
DATA.rename( columns={0 :'Price'}, inplace=True )

# Plotting the data
DATA.plot()
plt.xlabel('Date',fontsize=14)
plt.ylabel('Price',fontsize=14)

# We'll split training and test data as 80/20 respectively
# Getting the number of rows to train the model on.
train_size= int(len(DATA)*0.8)
# The number of rows to test the model on
test_size= len(DATA) - train_size
#Splitting train and test data then printing the size (rows) of each
train, test = DATA.iloc[0:train_size], DATA.iloc[train_size:len(DATA)]


# function:create_dataset
# converts data into numpy arrays
def create_dataset(X,y,time_steps=1):
    Xs, ys= [],[]
    for i in range(len(X)-time_steps):
        v= X.iloc[i:(i+time_steps)].values
        Xs.append(v)
        ys.append(y.iloc[i+time_steps])
    return np.array(Xs), np.array(ys)

time_steps = 1

# Splitting data into X_train and y_train datasets
# (train.Price and test.Price extracts the data from the train and test
    dataframe)
X_train, y_train = create_dataset(train, train.Price, time_steps)

# Splitting test data into X_test and y_test datasets
X_test, y_test = create_dataset(test, test.Price, time_steps)



# Defining the LSTM network architecture
model=keras.Sequential( )
model.add(keras.layers.LSTM(128,input_shape=(X_train.shape[1],X_train.shape[2])
    ))

model.add(keras.layers.Dense(1))
# Compile the model
```

```
model.compile(loss='mean_squared_error',optimizer=keras.optimizers.Adam(0.001))

# Train the model (we use 250 iterations)
history= model.fit(X_train, y_train, epochs=250, batch_size=16,

plt.plot(history.history['loss'], label= 'train')
plt.plot(history.history['val_loss'], label= 'test')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.legend();

# Get the models predicted price values
y_pred=model.predict(X_test)

# Plot the predictions along with the true outcomes
plt.plot(np.arange(0,len(y_train)), y_train, 'g',label="history")
plt.plot(np.arange(len(y_train), len(y_train)+ len(y_test)), y_test, marker='.'
    ,label="true")
plt.plot(np.arange(len(y_train), len(y_train)+ len(y_test)), y_pred,  'r',
    label="prediction")

plt.ylabel('Price')
plt.xlabel('Time Step')
plt.legend()
plt.show();


# Showing the actual and predicted prices
actual= pd.DataFrame(y_test,columns= ['True'])
actual.insert(1, "Predicted", y_pred, True)

# Calculating the RMSE
rmse = sqrt(mean((y_pred-y_test)**2))
print(f'The root mean-squared-error is: {rmse}')
valid
```

Listing A.4: Python program to predict Facebook stock price using an LSTM NN.

```
    # Importing the relevant libraries and API
import pandas as pd
pd.options.display.max_colwidth = 60

import tensorflow as tf
from tensorflow import keras
import numpy as np
from numpy import mean
import yfinance as yf

import numpy as np
from math import sqrt
import statistics
from sklearn.preprocessing import StandardScaler
import pylab


%matplotlib inline
import matplotlib.pyplot as plt
from IPython.core.pylabtools import figsize
figsize(20, 5)

# Get the data for the stock Facebook by specifying the stock ticker, start
```

```
    date, and end date
data = yf.download('FB','2017-01-01','2020-01-01')
# We use data.tail() to view the first 5 values from the dataset
data.head()

#Converting the data series into a dataframe (so we can replace missing values
    easily)
# We are only interested in the close price of each day
Input= data.Close.to_frame()


# Cleaning the input by replacing missing values with the previous value
clean_input= Input.fillna(method='ffill')
DATA = clean_input
# Renaming the price column from '0' to 'Price
DATA.rename(columns={'Close' :'Price'}, inplace=True )
# Plotting the data
DATA.plot()
plt.xlabel('Date',fontsize=14)
plt.ylabel('Price',fontsize=14)

# We'll split training and test data as 80/20 respectively
# Getting the number of rows to train the model on.
train_size= int(len(DATA)*0.8)
# The number of rows to test the model on
test_size= len(DATA) - train_size
#Splitting train and test data then printing the size (rows) of each
train, test = DATA.iloc[0:train_size], DATA.iloc[train_size:len(DATA)]
print(len(train),len(test))


# function:create_dataset
# converts data into numpy arrays
def create_dataset(X,y,time_steps=1):
    Xs, ys= [],[]
    for i in range(len(X)-time_steps):
        v= X.iloc[i:(i+time_steps)].values
        Xs.append(v)
        ys.append(y.iloc[i+time_steps])
    return np.array(Xs), np.array(ys)

time_steps = 1

# Splitting data into X_train and y_train datasets
# (train.Price and test.Price extracts the data from the train and test
    dataframe)
X_train, y_train = create_dataset(train, train.Price, time_steps)

# Scaling and standardizing the train data. Also reshaping X_train as
    StandardScaler expects an
# array dimension <= 2. (We have a 3 dimensional array).
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.reshape(-1, 1))

# Splitting test data into X_test and y_test datasets
X_test, y_test = create_dataset(test, test.Price, time_steps)


# Reshaping the X_train_scaled array back into an 3d array
X_train_scaled=X_train_scaled.reshape(X_train.shape[0],1,1)

# Defining the LSTM network architecture
model=keras.Sequential( )
```

```
model.add(keras.layers.LSTM(128,input_shape=(X_train_scaled.shape[1],
    X_train_scaled.shape[2]
)))

model.add(keras.layers.Dense(1))
# Compile the model
model.compile(loss='mean_squared_error',optimizer=keras.optimizers.Adam(0.001))

# Train the model (we use 250 iterations)
history= model.fit(X_train_scaled, y_train, epochs=250, batch_size=16,
                   validation_split= 0.1,verbose=1,shuffle=False)

plt.figure(figsize=(10,4))
plt.plot(history.history['loss'], label= 'train')
plt.plot(history.history['val_loss'], label= 'test')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.legend();

# Using the scaler to transform the test data.
X_test_scaled= scaler.transform(X_test.reshape(-1, 1))

# Reshaping the X_train_scaled array into an 3d array
X_test_scaled=X_test_scaled.reshape(X_test.shape[0],1,1)
# Get the models predicted price values
y_pred=model.predict(X_test_scaled)


# Plot the predictions along with the true outcomes
plt.figure(figsize=(10,5))
plt.plot(np.arange(0,len(y_train)), y_train, 'g',label="history")
plt.plot(np.arange(len(y_train), len(y_train)+ len(y_test)), y_test, marker='.'
    ,label="true")
plt.plot(np.arange(len(y_train), len(y_train)+ len(y_test)), y_pred,  'r',
    label="prediction")

plt.ylabel('Price')
plt.xlabel('Time Step')
plt.legend()
plt.show();

# Showing the actual and predicted prices
actual= pd.DataFrame(y_test,columns= ['True'])
actual.insert(1, "Predicted", y_pred, True)

# Calculating the RMSE
rmse = sqrt(mean((y_pred-y_test)**2))
print(f'The root mean-squared-error is: {rmse}')
# Outputting the true and predicted values table
actual
```