Evaluating the effectiveness of neural network techniques in the forecasting of South African basic fuel prices

by

Russell Kingwill

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science (Applied Mathematics) in the Faculty of Science at Stellenbosch University

Supervisor: Dr W.H. Brink

April 2019

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: April 2019

Abstract

South Africa has a number of fuel grades available to consumers, one of the most popular being the 95 unleaded standard. The price of this fuel is comprised of many components including transport fees, taxes and the basic fuel price. The basic fuel price is the cost in Rand of Brent crude oil used to refine the unit of petrol fuel, and is often the most significant component of the fuel price as well as the most volatile. Having a reliable forecasting methodology for the basic fuel price would be a helpful planning tool for many individuals and small enterprises. The forecasting of general fuel prices has been studied in the past with various forecasting techniques ranging from machine learning to ARIMA and regression models. In this study various deep learning models, including feed forward, recurrent and convolutional neural networks are assessed for their ability to accurately forecast the basic fuel price. These models are ranked by their ability to reduce the mean absolute percentage error on a common test data set. A number of time series data sets are used as input for the models under review, which include the closing daily price of Brent crude oil and the closing daily US Dollar exchange rate. The effect of inputting the 30 day rolling future contracts for both the closing oil price and exchange rates is also investigated.

Overall it is determined that, of the models evaluated during this study, the recurrent network performs the most favourably. On the final test set, with optimal model and input parameters, the individual observation errors range from less than 1 % to more than 10 %. The average test error of 4.57 % can be a bit misleading due to the observed range of individual errors. Hence it is not as reliable of a forecast as one would hope for. However, the model did prove to have a fairly reliable attribute to correctly forecast the direction of the basic fuel price change. It did so in about 86% of the test data set observations, and was off by only a few cents when an incorrect direction was forecast. It is concluded that neural network models can be used to some degree for the task of forecasting the South African basic fuel price. Such models are sensitive to the amount of data provided and hence future work in this area should prioritise obtaining more data and if possible incorporating additional data sources.

Uittreksel

Suid-Afrika het 'n aantal brandstofgrade, waarvan een van die gewildste die 95-loodvrye standaard is. Hierdie brandstof se prys bestaan ondermeer uit vervoerfooie, belasting en die basiese brandstofprys. Die basiese brandstofprys is die koste in Rand van Brent-ru-olie wat vir verfyning gebruik word, en is dikwels die belangrikste komponent van die brandstofprys sowel as die mees wisselvallige. Om 'n betroubare voorspellingsmetodologie vir die basiese brandstofprys te hê, kan nuttig vir baie individue en klein ondernemings wees. Die voorspelling van algemene brandstofpryse is in die verlede bestudeer met tegnieke wat wissel van masjienleer tot ARIMA en regressiemodelle. In hierdie studie word verskeie diepleermodelle, insluitende voortvoerende, terugkerende en konvolusie neurale netwerke, beoordeel vir hul vermoë om die basiese brandstofprys akkuraat te voorspel. Hierdie modelle word gerangskik volgens hul vermoë om die gemiddelde absolute persentasie-fout op 'n algemene toetsdatastel te verminder. 'n Aantal tydreeksdatastelle is gebruik as intreë wat die sluiting van die daaglikse prys van Brent-ru-olie en van die daaglikse Amerikaanse Dollar-wisselkoers insluit. Die effek van die insluiting van die 30 dae toekomstige kontrakte vir die sluitingsprys en wisselkoerse word ook ondersoek.

Oor die algemeen is vasgestel dat van die modelle wat in hierdie studie geëvalueer is, die terugkerende netwerk die gunstigste presteer. Op die finale toetsstel, met optimale model- en insetparameters, wissel individuele foute van minder as 1 % tot meer as 10 %. Die gemiddelde fout op die toetsdatastel van 4.57 % kan 'n bietjie misleidend wees as gevolg van die verspreiding van individuele foute. Dit is dus nie so betroubaar soos wat mens sou hoop nie. Die model toon egter 'n redelike betroubare vermoë om die rigting van verandering in die basiese brandstofprys korrek te voorspel. Dit is gedoen in ongeveer 86% van die toetsdatastel waarnemings, en was af met slegs 'n paar sent toe 'n verkeerde rigting voorspel is. Daar word tot die gevolgtrekking gekom dat neurale netwerkmodelle tot 'n mate gebruik kan word om die Suid-Afrikaanse basiese brandstofprys te voorspel. Sulke modelle is sensitief vir die hoeveelheid data wat verskaf word en daarom moet toekomstige werk in hierdie gebied voorkeur gee aan die verkryging van meer data en indien moontlik die insluiting van bykomende databronne.

Acknowledgements

I would like to take some time to thank the following people for their various contributions to the completion of this study:

- My thesis advisor, Dr Willie Brink.
- Fellow masters student, Greg Newman.
- My colleague in finance, Joanna Lambrinos.
- Finally, my family and girlfriend Sonia.

Contents

Chapter 1

Introduction

This first Chapter of the study outlines the effect of the petrol price on an economy, how it is defined and why it can be an important factor to model. The objectives of this study are then discussed and a brief overview is provided to give some context to the content that will be presented in the remaining Chapters.

1.1 Effects of fuel pricing

Changes in fuel pricing can affect an entire nation's economy, but fluctuations in price can be most vividly felt at the consumer level. Increased pump prices leave less funds in the budget for goods and services. In the physical retail space, higher prices could mean that shoppers might drive less for their purchases and retailers are forced to pass on the expenses associated with increased shipping costs to their consumers. Increased pressure on public transportation services and cost of public transportation can also be attributed to an increase at the pump. In some cases high transportation costs have led to businesses and colleges implementing a 4 day week in an attempt to lower expense for employees and students [1]. Due to the wide sweeping effect of the national fuel price, it is often seen as an informal measure of the health of an economy, due to the inverse relationship between consumer confidence and fuel pricing [1].

In South Africa an estimated 50% of people live in poverty [2]. The working poor are vulnerable to increases in the cost of public transportation due to changes in the fuel price [3], as the cost of transportation makes up a large portion of their monthly budget, in some cases up to 20% [4; 5]. With a proposed minimum wage of R3500 per month, even a small increase in necessary travel costs can have a large impact on the quality of life of millions of people [4].

Dramatic changes in the cost of fuel can also have a major effect on small and medium enterprises (SME). In South Africa, SMEs make up 91% of formalised businesses, provide employment to about 60% of the labour force and their total economic output accounts for roughly 34% of gross domestic product (GDP) [6]. Such price changes can present cash flow challenges for their day to day operations [7]. In an attempt to address this some companies and consultants have looked to the forecasting of future fuel prices, using simple linear models to get some insight as to how prices could potentially change. This allows them to put some preparations in place in an attempt to mitigate price shocks [8].

1.2 Petrol fuel price

South African petrol stations have various grades of fuel available to motorists, both unleaded and lead replacement variants. About 1 to 2 percent of vehicles in South Africa require what is referred to as lead replacement fuel [9; 10]. The latest grade of unleaded petrol is known as 95 unleaded, in reference to its octane number [11]. This grade was first made available to the general public in February 1996 [12] and is currently the most popular. Unleaded 95 has two quoted prices, one for the coastal provinces and the other higher price for the inland provinces. The difference between these two is down to the additional transport related costs required to supply inland pumps [13]. The fuel price is adjusted on the first Wednesday of each month following a review period covering the previous month [14].

1.2.1 Fuel price composition

The South African unleaded 95 fuel price is comprised of various components including taxes, levies, fees, Road Accident Fund (RAF) contributions and the Basic Fuel Price (BFP) [15]. At time of writing the BFP makes up approximately 44% of the total pump fuel price [16; 17]. The taxes, levies, fees, contributions etc. are overseen by various South African governmental agencies including the Department of Energy and the ministry of Finance [15]. These are altered as directed by government policy decisions such as those outlined in the national budget speech [18]. The composition of the 95 unleaded fuel price for February 2018 can be seen in Table 1.1.

1.2.2 Basic fuel price (BFP)

The formula to compute the BFP was first used in April 2003, implemented by the then Department of Minerals and Energy (DME). The BFP formula replaced the In Bond Landed Cost (IBLC) method which was based on various refinery gate postings. So called refinery gate prices were found to have a poor

Table 1.1: Levies, taxes and margins (95 unleaded fuel) [19]

Component	Price (RSA c/litre)
BFP	622.17
Fuel tax	315.0
Customs excise	4.0
Equalisation fund levy	0.0
Road accident fund	163.0
Transport cost	41.5
Petroleum products levy	0.33
Wholesale margin	34.0
Secondary storage	18.6
Secondary distribution	15.9
Retail margin	187.2
Slate levy	0.0
Delivery cost	0.0
Demand side management levy	10.0

correlation to international market prices, and hence a suitable replacement was proposed and implemented [14].

The BFP fluctuates daily relative to changes in the price of Brent crude oil in international oil markets, specifically the market price of oil in Singapore and the Arab Gulf [15]. International prices are driven by supply and demand for commodities in a market. Approximately 36% of local demand is met by locally produced synthetic fuels, mostly from coal and natural gas. The remaining 64% is generated by locally refined imported Brent crude oil [20]. Crude oil is the largest input cost for a refinery, so in order to cover costs when the price of crude oil rises the price of local petrol will rise in a similar manner [20]. As the price of Brent crude is quoted in US Dollars, the exchange rate between the South African Rand and US Dollar has an effect on the BFP. Due to this, the BFP can change independently of underlying oil price as the exchange rate changes relative to the perceived health of the economy [21; 14]. The relationship between these elements can be seen in Figure 1.1.

The South African Central Energy Fund (CEF) group is a state-owned energy utility which is comprised of several companies in the local energy sector, including PetroSA and the Strategic Oil Fund (SFF) among others [22]. The CEF group is responsible for providing the daily BFP and associated pump price to the DOE [14]. The BFP takes into account the concept of an over under slate account administered by the CEF [14]. The daily calculated BFP is either higher or lower than the BFP reflected in the fuel price at that time. If the daily BFP is higher than the BFP in the fuel price, an under recovery unit is realised on that day. When the BFP is lower than the BFP in the fuel price, an over recovery unit is realised. An under recovery implies that the

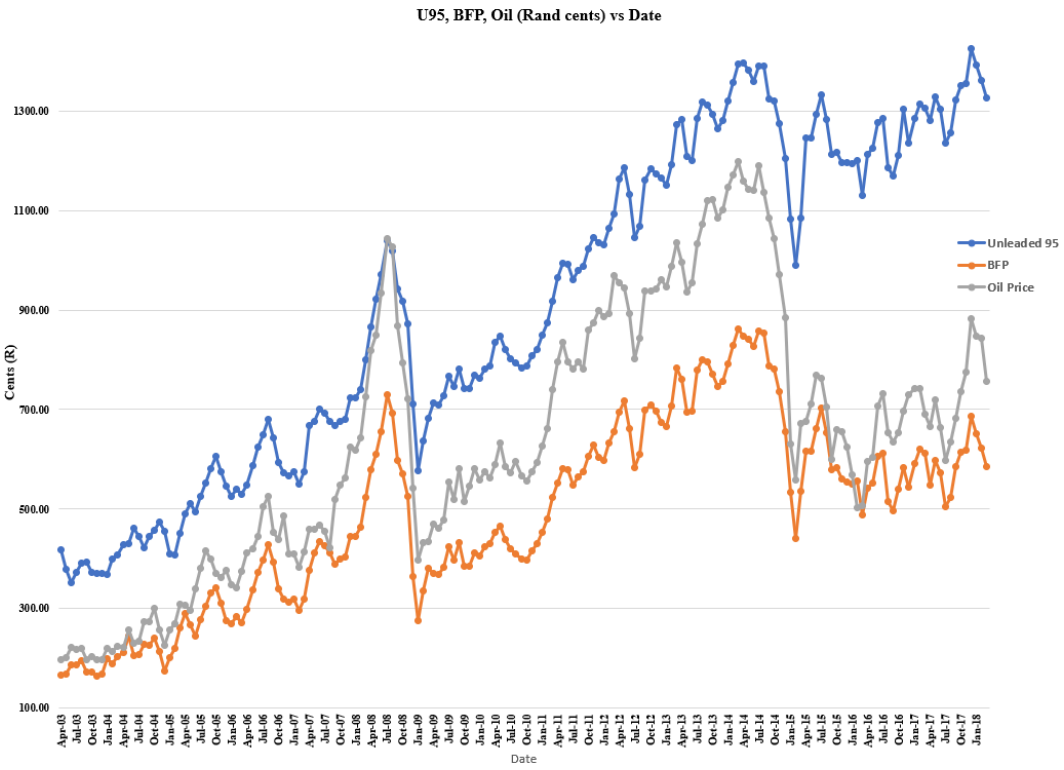


Figure 1.1: Unleaded 95, BFP and barrel of oil in South African Rand cents vs date

consumer is paying too little for product on that day, while an over recovery implies they are paying too much [20; 23]. These calculations are done for each day in the fuel price review period. If needed the average balance of the slate account will have an impact on the following month's revised fuel price [23].

The BFP also includes costs associated with shipping petroleum products to South Africa, and these costs include insurance, storage, and wharfage. These components of the BFP are relatively small and so have little effect when compared to the price of Brent crude oil [15].

1.3 Study objectives

Neural networks trained on time series data have had some success in the forecasting of commodities and stock prices [24]. As oil prices and exchange rates are similar time series financial instruments, they can be used as inputs to models for the prediction of fuel prices. As mentioned in Section 1.1, simple linear models have had some success in this regard, but may not be sophisticated enough for forecasting at the monthly or quarterly time horizons [25]. So the aim of this thesis is to determine the effectiveness of various neural network models in forecasting the South African unleaded 95 BFP. Since the BFP is the single largest and the most volatile component of the overall fuel price, having

an accurate assumption of its future value would be a requirement to forecast the overall fuel price. If useful, such techniques could be helpful to local SMEs in budgeting functions, or government groups looking to understand how the poor will be affected by coming fuel price shocks.

1.4 Overview

The remainder of this thesis will span from Chapter 2 through to Chapter 6. Chapter 2 comprises of a literature study where existing methods of forecasting fuel and commodity prices are compared and assessed. Chapter 3 is an examination of the related theory pertaining to the development of various machine learning models that can be used for time series based forecasting tasks. Chapter 3 also contains some definitions of common financial instruments that are relevant to this study. Chapter 4 is broken down into two main sections, tools and experiments. The tools section highlights the various hardware and software components that were used to create and execute the various machine learning models under review. The list of consulted data sources are also stated under this heading. The experiments section of the Chapter gives an outline of the various experiments conducted on different models and the method of assessing performance for each model, given the experimental task. Chapter 5 presents the results and discussion for the experiments as defined in Chapter 4. Lastly, Chapter 6 provides a conclusion to the question of whether a machine learning approach can be used to effectively forecast the South African BFP. Chapter 6 also provides a brief description of potential future work that can be undertaken in an extension to what is presented here.

Chapter 2

Literature review

Little work has been published on the forecasting of the South African fuel price, especially the basic fuel price using neural networks. Industry professionals have made use of simple regression models in order to gain some rudimentary insight into the direction of the price given the Rand Dollar exchange rate and the price of Brent crude oil. Abroad, there has been some work done on the forecasting of various fuel prices using a neural network approach, for instance Indian petrol prices and US jet fuel prices [26; 27], but again not much. Given this lack of domain specific research it could be worthwhile to look at the broader category of commodity price forecasting and similar, stock price forecasting. Such fields have been an active area of research for many years and continue to attract a lot of attention. They seem to have similar data sources such as exchange rates and spot prices of oil, and most of the data sources in this domain are inherently time series based. The data in these domains also tend to have similar characteristics such as high noise and seasonality. Time series data can be challenging to work with, so the models which operate successfully on such data can be relatively sophisticated and robust in their ability to forecast reliably accurate results. Such qualities would be helpful in the forecasting of the South African BFP given how volatile the South African economic environment can be [28].

This Chapter will first look at what existing non-network methods have been used to forecast stock, commodity and fuel prices. Secondly, it will highlight what network based approaches have been used to perform similar forecasting tasks, and lastly, what hybrid methods have been proposed and evaluated for similar tasks.

2.1 Non-network forecasting

Existing and popular non-network orientated modelling methodologies have had various levels of success when operating on time series data sources, similar to what will be used to forecast local basic fuel prices. A few are described in this Section.

2.1.1 ARIMA model

A popular method of commodity and stock price prediction is the use of autoregressive integrated moving average (ARIMA) models. ARIMA models are a form of statistical analysis that uses time series data to predict future values [29]. The future value output of an ARIMA model is predicated on a linear combination of past values and past errors, which can be written as:

$$Y_t = \phi_0 + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + ... + \phi_p Y_{t-p} + \mathcal{E}_t - \theta_1 \mathcal{E}_{t-1} - \theta_2 \mathcal{E}_{t-2} - ... - \theta_q \mathcal{E}_{t-q} \quad (2.1)$$

where Y_t and $\mathcal{E}_t$ are the value and error at time t. ϕ and θ are the model coefficients and p and q are the auto-regressive and moving average parameters [30]. Adebiyi et al. in [30] found that using published New York Stock Exchange (NYSE) and Nigeria Stock Exchange (NSE) data along with ARIMA models, they were able to reliably forecast short term stock prices for a major electronics company and bank.

2.1.2 Markov model

Another popular method that has been used to forecast commodity and stock prices is the Markov model. It is based on the Markov assumption, which is that the next state of the system is only dependant on its current state. Isah et al. [31] made use of a Markov model to forecast the short term price of crude oil using time series data obtained from the World Trade Institute (WTI). They concluded that the Markov based model was an effective methodology to accurately forecast crude oil prices [31].

2.1.3 Regression model

Multivariate regression models can be used to forecast many quantities including stock index pricing. The general form of the multivariate linear regression model can be expressed as:

$$y = b_0 + b_1 x_1 + b_2 x_2 + ... + b_p x_p \quad (2.2)$$

for p occurrences of input x, and where b is the model coefficients. Cheng et al. [32] looked at using such models to forecast the Hang Seng Index in Hong Kong with market time series data and governmental macro measurements such as

the unemployment rate. Overall they found the model to be too sensitive to changes in the input data, especially the macro variables, and not expressive enough to reliably forecast the direction of the change in price. They concluded that such un-modified models should not be used for real world applications, such as a model used to generate an investment strategy. However, these techniques have the potential to be the base of a more robust model [32]. Using such an approach can be a useful benchmark for the forecasting of the local BFP.

2.1.4 Thailand fuel price forecast

Tipyan et al. [33] looked at using a number of quantitative models for the forecasting of various grades of Thai vehicle fuel prices. Methods that were assessed include Holt's exponential smoothing, decomposition and regression. They identified a number of important input parameters for fuel price prediction which include world oil demand, US Dollar exchange rate, and the market price of oil in Singapore. Tipyan et al. went on to describe a composite model which was heavily influenced by the regression model. Such a model appeared to be sufficiently capable of forecasting the various grades of Thai fuel prices [33].

2.2 Network based forecasting

Using neural networks to forecast future stock prices has been a field of interest since the 1980s [34], where they were found to be a capable modelling technique for tasks requiring patten recognition and nonlinear forecasting. Since then there has been much research and implementation using neural networks in the financial and economic domain.

2.2.1 Neural networks (NNs)

Lee et al. [35] found that a neural network can be an effective method to forecast returns on the Korean Stock Exchange (KSE) using Korean stock price data. They describe two main benefits of NNs in the capital market domain. First, the models are data driven, which implies they learn from the input data without any additional assumptions. Secondly, NNs are capable of processing large amounts of fuzzy, noisy, and unstructured data [35]. According to Lee et al. the KSE is fairly volatile when compared to more mature markets such as the NYSE [35], so if an NN approach is suitably effective in this environment, it would be beneficial to evaluate its use in the case of forecasting the South African BFP.

Abe et al. [36], working with data from the Japanese stock market to forecast stock returns, found that a deeper NN can be more effective than a shallow

network with the same input data. Their deeper models allowed for an increase in representational power and improvements to prediction accuracy due to repeated nonlinear transformations. A deep network is a network with multiple layers, as discussed in further detail in Chapter 3.

Kulkarni et al. in [37] proposed a deep NN approach to forecasting short term oil prices, using historical oil prices and oil future contracts. They investigated the role of oil future contracts on the ability to yield actionable information about the direction of a move in the spot price of oil. They concluded that, in some cases, using oil futures can provide valuable information in the forecasting of spot prices [37]. They found that an NN can be a useful tool for forecasting commodity prices. Using oil future contracts as input to the models for local BFP pricing could yield in a more accurate result, as a more accurate estimation of a future oil price should produce a more accurate forecast for the BFP.

2.2.2 Recurrent neural networks (RNNs)

Recurrent neural networks seem to be a promising modelling methodology for the task of producing a forecast based off time series data. The input to an RNN is normally some sequence, and the RNN network topology has the ability to detect patterns in input sequences. RNNs are discussed further in Chapter 3.

Saad et al. in their 1998 paper [24] reviewed a number of network topologies with respect to their ability to correctly forecast a number of publicly listed companies' stock price. These companies were drawn from various domains of the economy, including banking, technology and entertainment. Such domains are subject to different levels of price volatility. Saad et al. found all networks yielded comparable results but that recurrent networks seemed to be the most powerful due to their ability to incorporate past observations given their internal recurrence. They noted a downside to the recurrent approach, being the increased implementation complexity with respect to other network topologies such as a more traditional neural network [24].

As mentioned, the South African economic environment is rather volatile, so using a recurrent approach to BFP prediction could be useful in smoothing out prediction errors and potentially yield more stable results. These apparent advantages will need to be weighed against the accuracy and implementation requirements of other models.

Ugurlu et al. [38] looked at various NN techniques including recurrent networks, making use of the Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) architectures to forecast Turkish electricity prices off time series data. These architectures are discussed further in Chapter 3. They found that a recurrent network with GRU cells to be the most effective, and it outper-

formed other NN and statistical methods. They found the recurrent approach to be the most suitable method due to its memory of previous observations in the input series. Also stated was that the GRU seemed to be more performant than the LSTM due to the reduced number of parameters to be learned [38]. Forecasting electricity can be a challenging task, as the price is subject to high volatility, sharp price spikes and seasonality. The success of RNNs in this domain as outlined by Ugurlu et al. bodes well for their implementation in forecasting of the local BFP.

2.2.3 Convolutional neural networks (CNNs)

CNNs are biologically inspired and have seen much success in the realm of image recognition and classification. The building block of these networks is the mathematical convolution operation using learnable filters between the input data and an expected output. This topology is discussed further in Chapter 3. Borovykh et al. in a recent paper [39] presented a convolutional network based method for the forecasting of time series data, including the Standard and Poors (S&P) 500, Chicago Board Options Exchange (CBOE) interest rate and several exchange rates. As noted by Borovykh et al., literature on financial time series forecasting with convolutional architectures is scarce, as the domain is dominated by autoregressive and recurrent models. Nevertheless the ideas behind the CNN are compelling in this use case. Borovykh et al. state that a CNN could be used to learn filters that represent certain repeating patterns in time series data and utilise these to forecast a future value [39]. The model should be able to handle noisy input data by leveraging the patterns that have been identified as meaningful. Their deep convolutional model was inspired by the WaveNet audio model. Borovykh et al. concluded that the CNN approach to time series forecasting was at least comparable to the more common recurrent approach and could be used as a baseline for evaluating forecasting methods. Additionally the CNN proved to be simpler to implement and require less computational and memory resources to produce similar results relative to the recurrent approach [39].

Given that the underlying data used to generate the fuel price is in the format of a time series, it would be worthwhile to investigate the utility of a convolutional approach to BFP forecasting.

2.2.4 US navy jet fuel forecast

In 1995, Kasprzak [26] looked at using a neural network method to accurately forecast the future prices of jet fuel for the Defence Fuel Supply Center (DFSC). The proposed network model was evaluated relative to an existing regression based model. Kasprzak found the proposed model to provide more accurate

predictions, and more robust in the presence of statistical outliers in the input data, when compared to the existing regression based model.

2.3 Hybrid approaches

Attempts have been made by various researchers to combine the best qualities of machine learning and more traditional techniques, such as neural networks and ARIMA models.

2.3.1 Neural network hybrid

Sallehuddin et al. proposed such a method in [40]. Specifically they proposed the GRANN ARIMA model, which integrates the nonlinear Grey Relational Artificial Neural Network (GRANN) and the linear ARIMA model. Grey relational analysis (GRA) is an analysis method introduced by Deng Julong to assess the degree of correlation for different data sequences. The details for the model can be found in [40]. The hybrid model was assessed on several time series data sources including data from the Kuala Lumpur Stock Exchange (KLSE). Sallehuddin et al. found the hybrid suitably effective and a potential alternative tool for forecasting time series data for better forecasting accuracy when compared to more traditional methods.

2.3.2 Support vector machine hybrid

Another attempt to create a hybridisation of two models was made by Pai et al. in [41], combining a Support Vector Machine and an ARIMA model. An SVM is a machine learning technique that can be used to solve nonlinear regression based problems. The hybrid model was evaluated relative to a single SVM and single ARIMA model on 10 stocks of publicly listed companies in the US. Pai et al. found the performance of the hybrid model to be promising, proving to be more effective in its ability to forecast stock prices off time series data than either of the individual models of which it is comprised [41].

2.3.3 Indian fuel price forecast

Thakur et al. [27] made use of a composite Nonlinear Autoregressive Exogenous (NARX) model in an attempt to forecast the petrol prices in India. The composite model consisted of neural network and autoregressive elements. The data upon which it was trained came from the US Energy Information and Administration (EIA) department. Thakur et al. concluded that such a hybrid approach was a robust and highly accurate method for the forecasting of Indian fuel prices.

2.3.4 Summary

This Chapter has highlighted the performance of a number of existing time series based forecasting methods, ranging from the more traditional approaches such as ARIMA model to the relatively speculative abilities of the CNN. As the context for this study is to determine the effectiveness of various neural network models for their ability to forecast the BFP, the upcoming theory Chapter will focus mainly on the NN, RNN and the CNN.

Chapter 3

Theory

This Chapter will lay out some of the theoretical groundwork required to develop various machine learning models for the forecasting of the South African BFP. To start off, a number of machine learning fundamentals are defined. These are followed by a detailed description of feed forward, recurrent and convolutional networks. The reader is directed to the text, *Deep Learning* by Goodfellow, Bengio and Courville for additional overview of these concepts. Lastly a few financial instruments are defined, which are referenced at various points later in the thesis.

3.1 Machine learning principles

The content presented in this Section is intended to given the reader an overview of some core Machine Learning elements. As such, these concepts will not necessarily be directly referenced later in this study.

3.1.1 Learning algorithm

A machine learning algorithm is an algorithm that is able to learn from data [42]. Learning can be defined as follows: *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E* [43].

3.1.1.1 Task (T)

Machine learning allows one to tackle tasks that can be too difficult to address with handcrafted solutions, such as getting a bipedal robot to walk [42]. These tasks are described as to how the system should process the example, where an example is a collection of features that have been measured from some entity the learning system is to operate on. An example can be typically represented

as a vector $x \in \mathbb{R}^n$ where each element x_i represents a feature. Typical tasks include (but are not limited to) the following:

- regression: predict a numerical value given some input;
- classification: classify inputs into predefined classes;
- anomaly detection: evaluate input to determine if abnormal or unusual.

3.1.1.2 Measure (P)

In order to determine the effectiveness of a machine learning approach, a quantitative measure (P) is needed to evaluate the algorithm with respect to the task under review (T). For example, with a classification task, a measure related to its performance would be the error rate, which is related to the proportion of inputs incorrectly categorised relative to the expected output.

Often it is best practice to evaluate algorithms on a test set of data, that is a set of data the algorithm has not been exposed to during training. This allows for a more representative indication of the algorithm's performance when deployed in a real world scenario.

3.1.1.3 Experience (E)

Machine learning experience (E) can be roughly broken down into two broad classes: that of supervised and unsupervised learning. In supervised learning, the algorithm is exposed to input features and is provided with an expected output target or label, often denoted as y. In unsupervised learning there is no expected target y for a given input x. So the goal of an unsupervised approach is to determine the useful properties of features contained in an input data set in an attempt to uncover the probability distribution responsible for its generation.

The definitions for supervised and unsupervised learning are not completely formal, but they do help to categorise some of the tasks that can performed with machine learning algorithms [42].

3.1.2 Data sets and generalisation

A primary goal of a machine learning algorithm is to operate well on inputs not observed during training, which is referred to as a model's ability to generalise. During the training phase, the aim is to maximise the model's performance given the training data set. In addition to a low training error, a comparably low test error on a separate test data set is also desired. The test error is also known as the generalisation error. The generalisation error is taken across different possible inputs drawn from a distribution that would be expected in a

real world scenario. It would be ideal to collect the test data set independently from the training set.

It is considered best practice to split the examples allocated for training into two distinct sets, roughly 80% for standard model training, i.e. learning the models parameters, and the remaining approximately 20% towards a validation set used to estimate the generalisation error, allowing the model's hyperparameters to be tuned as required. Hyperparameters are discussed in Section 3.1.3. Such an approach allows for a more accurate measurement of the model's effectiveness as the test set has not been used to influence the model's topology or parameters.

3.1.2.1 Distribution assumptions

Typically it is assumed that the training and test data sets follow the independent and identically distributed (IID) assumptions. That is, the elements in each set are independent from one another, and both sets are drawn from the same probability distribution. With this, it can be observed that for some model the expected training error is equal to the expected test error [42].

3.1.2.2 Model fitting

In the general case for training a machine learning algorithm, the training set is sampled to help identify the best model parameters by minimising the training set error. Given this, it can be expected that the test error is greater than or equal to the training error. So to optimally train some model, it is best to minimise the training error and the difference between the training error and test error. These two points can lead to the phenomena known as under- and over-fitting. Under-fitting occurs when the model is unable to obtain a sufficiently low error on the training data set, so no acceptable relationship within the data can be reliably identified. Over-fitting occurs when the difference between the training and test errors is large. That is, the model has learnt the relationship present in the training data but cannot adequately generalise it to new inputs.

3.1.2.3 Model capacity

A model's capacity indicates its ability to fit various functions. A model with lower capacity may struggle to sufficiently fit the training set, whereas a model with a higher capacity may over-fit due to memorising the training data. The capacity of a model can be varied by adjusting the hypothesis space, that being the set of functions available to the model to select in an attempt to suitably fit the input data sets [42].

3.1.3 Hyperparameters

Hyperparameters are used to tune the performance of the machine learning algorithm. An example of a hyperparameter could be the model's capacity or the impact of weight decay in a regularisation process. The values of the hyperparameters are not adjusted by the model itself during training, although it is possible to implement a model to discover the optimal hyperparameters for another machine learning model [42].

3.1.4 Regularisation

The "no free lunch" theorem for machine learning states that, averaged over all possible data generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points [44]. So no machine learning algorithm is universally better than any other.

The above implies that a machine learning algorithm should be tailored to perform well on a specific task. This can be achieved by incorporating a set of problem-specific preferences that are geared toward a class of possible solutions. These preferences, such as a preference for lower-order functions, are known as regularisation. Regularisation can be any modification we make to a learning algorithm that is intended to reduce its generalisation error, but not its training error [42]. A few examples of regularisation techniques are presented below.

3.1.4.1 Norm penalties

Many regularisation techniques are based on limiting the model capacity, by adding a parameter norm penalty $\Omega(\theta)$ to the objective function $J(\theta; X, y)$, where θ contains all the model parameters, X is a matrix of inputs and y is a vector of expected outputs. Such a regularised function can be stated as

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta), \tag{3.1}$$

where $\alpha \in [0, \infty)$. In the above α is a weighting that controls the impact of the norm penalty on the function. Typically the norm penalty only affects the weights of the affine transformation at each layer in the neural network and leaves the bias terms un-regularised [42]. These concepts are discussed further in Section 3.2.

L^2 norm: The L^2 parameter norm penalty is one of the simplest and most popular penalties. It is also referred to as weight decay. This approach to regularisation drives the weight vectors in a neural network toward the origin by the addition of the regularisation term $\Omega(\theta) = \frac{1}{2}\|\theta\|_2^2$.

L^1 norm: Another less popular example of a norm penalty, is the L^1 norm penalty, defined as: $\Omega(\theta) = \|\theta\|_1 = \sum_i |\theta|$, which is the sum of absolute values of the individual parameters.

The L^2 norm is often preferred over the L^1 norm as it tends to penalise larger errors more aggressively which can lead to better results.

3.1.4.2 Early stopping

The early stopping regularisation strategy is a simple, effective and non-obtrusive method to improve results. It essentially works by monitoring the validation error. If this error has not improved for some defined number of iterations, then the algorithm returns the model parameters at that point. This may occur before the model training process has completed, and the point at which it returns may not be the local or global minimum of the training error, but hopefully before the model starts to over-fit the training data.

The early stopping approach can have additional cost to it, as periodically running the validation set evaluation can be a resource intensive exercise. Early stopping is a useful technique as it provides regularisation without modifying the model, and potentially limits the number of training iterations.

3.1.4.3 Bagging

Bagging (Bootstrap AGGregating) is another technique for reducing the generalisation error by combining several models [45]. The goal is to independently train several different models, then determine the average of the output for the test examples. This approach is effective as the models will normally not make the same errors on the same input data, bagging is an example of an ensemble method. Bagging allows the same model architecture, algorithm and objective function to be used multiple times.

In general, models can be ensembled together in various ways. For instance different model architectures, algorithms or objective functions can be evaluated in unison. This approach can be effective in reducing the test error [42].

3.1.4.4 Dropout

The regularisation method known as "dropout" provides a computationally inexpensive but effective method of regularising various classes of models. At first, it can be thought of as a method of bagging for ensembles of many large networks. Specifically it trains the ensemble comprising of all sub-networks that can be obtained by removing a subset of non-output units from an underlying network. Most networks are based on a series of affine transformations and nonlinearities, so removing a unit can be done by multiplying its output by zero.

Dropout tends to be more effective than other standard computationally inexpensive regularisers, such as weight decay and filtering norm constraints [46]. Dropout may also be combined with other regularisation techniques to yield further improvements.

Dropout does not significantly limit the type of model or training procedure that can be used. It works well with most models that use a distributed representation and can be trained with stochastic gradient descent. Examples of applicable models include feed forward neural networks, probabilistic models such as restricted Boltzmann machines [46], and recurrent neural networks [47; 48].

Although the computational cost of applying dropout to a specific model is low, the cost in a full system can be significant, as dropout reduces the capacity of a model and hence its ability to effectively generalise. A reduction in model capacity can be mitigated by increasing the size of the model and increasing training iterations.

3.1.4.5 Adversary training

An adversarial input is one such that point x' is close to x, but the output of x' is far from the output associated to x, which can lead to large errors. In many cases the difference between x and x' is indistinguishable to a human observer. Training with these adversarially perturbed inputs can be an effective means of regularisation to reduce the test error of the model.

A primary cause for the effectiveness of adversarial examples is excessive linearity [49], as neural networks are built out of primarily linear building blocks. So it can be beneficial to make use of a large function family to allow for the flexibility to capture data trends and resist local data perturbation [42].

3.1.5 Maximum likelihood

The maximum likelihood estimation technique is commonly used to determine the effectiveness of a machine learning algorithm. It can be defined as follows.

With a set of m independent examples $\mathbb{X} = \{x^{(1)}, ..., x^{(m)}\}$ drawn independently from a data generating distribution $p_{data}(X)$, let $p_{model}(X;\theta)$ be a parametric family of probability distributions over the same space indexed by θ. The maximum likelihood estimator for θ can be written as:

$$\theta_{ML} = \arg\max_{\theta} p_{model}(\mathbb{X};\theta) \tag{3.2}$$

$$= \arg\max_{\theta} \prod_{i=1}^{m} p_{model}(x^{(i)};\theta). \tag{3.3}$$

A more convenient but equivalent representation can be obtained by taking the logarithm of the likelihood. This operation does not change the argmax but does transform the product into a sum:

$$\theta_{ML} = \arg\max_{\theta} \sum_{i=1}^{m} \log p_{model}(x^{(i)};\theta), \tag{3.4}$$

which can be expressed as an expectation with respect to the distribution $\hat{p}_{data}$ by dividing through by m:

$$\theta_{ML} = \arg\max_{\theta} \mathbb{E}_{x\sim\hat{p}_{data}} \log p_{model}(x;\theta). \tag{3.5}$$

An interpretation of the maximum likelihood is an attempt to minimise the difference between the data distribution and the model's distribution, with the degree of dissimilarity between the two measured by the Kullback-Leibler (KL) divergence:

$$D_{KL}(\hat{p}_{data}\|p_{model}) = \mathbb{E}_{x\sim\hat{p}_{data}}[\log \hat{p}_{data}(x) - \log p_{model}(x)]. \tag{3.6}$$

The KL divergence is a measure of how one probability distribution differs from an expected distribution. To minimise the KL divergence only the $-\mathbb{E}_{x\sim\hat{p}_{data}}[\log p_{model}(x)]$ term needs to be minimised as the term on the left is a function associated to the data generation process. The minimisation of the KL divergence corresponds to the minimisation of the cross entropy between the distributions [42].

3.1.6 Gradient based optimisation

Many machine learning models involve mathematical optimisation, where optimisation is the minimisation or maximisation of some function $f(x)$ by adjusting the parameter x. This function is referred to as the objective function, error function or cost function.

3.1.6.1 Gradient descent

Suppose there exists a function f such that $y = f(x)$ where $x, y \in \mathbb{R}$ and the derivative is given by $f'(x)$. The derivative can be used to determine the required input that corresponds to a desired output for a function:

$$f(x + \epsilon) \approx f(x) + \epsilon f'(x). \tag{3.7}$$

This property can be used for optimisation, for instance if $f(x - \epsilon \text{sign}(f'(x)))$ is less than $f(x)$ for a sufficiently small ϵ then $f(x)$ can be reduced by moving x with the opposite sign of the derivative. This process is referred to as gradient descent [50; 42].

When $f'(x) = 0$, the derivative provides no information about which direction to move. These points are known as critical points:

- local minimum: $f(x)$ is smaller than all neighbouring points;
- local maximum: $f(x)$ is larger than all neighbouring points;
- saddle point: a critical point that is neither a local maximum nor a local minimum;
- global minimum: $f(x)$ is smaller than function values at all other possible points;
- global maximum: $f(x)$ is larger than function values at all other possible points.

A graphical representation of some of these critical points can be seen in Figure 3.1.

The slope of the function f in direction u is the directional derivative relative to u. With the chain rule, the directional derivative of the function $f(x + \alpha u)$ with respect to α, evaluated at $\alpha = 0$, can be expressed as

$$\frac{\partial}{\partial \alpha} f(x + \alpha u) = u^\intercal \nabla_x f(x). \tag{3.8}$$

The function f can be minimised using the directional vector:

$$\min_{u, u^\intercal u = 1} u^\intercal \nabla_x f(x) = \min_{u, u^\intercal u = 1} \|u\|_2 \|\nabla_x f(x)\|_2 \cos\theta, \tag{3.9}$$

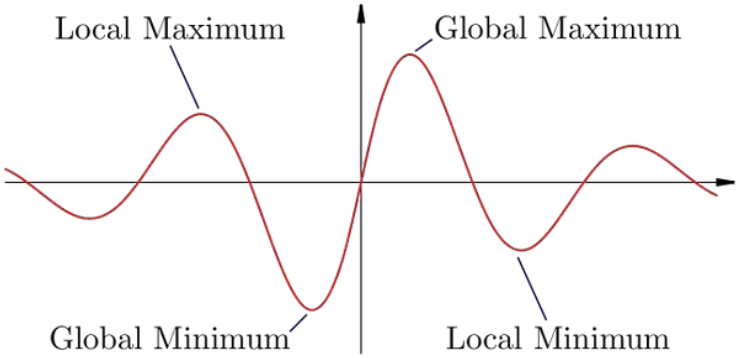


Figure 3.1: Visual representation of described critical points for some function $y = f(x)$.

where θ is the angle between u and the gradient. Setting $\|u\|_2 = 1$ and ignoring terms that do not depend on u, simplifies the expression to $\min_u \cos\theta$. This is minimised when u points in the opposite direction to the gradient, and $f(x)$ can be reduced by moving in that direction, hence the name gradient descent. This process proposes a new point, $x' = x - \epsilon\nabla_x f(x)$ where $\epsilon > 0$ is the learning rate. The learning rate is responsible for the magnitude of the gradient step.

3.1.6.2 Stochastic gradient descent

Stochastic gradient descent (SGD) is an extension to the gradient descent process, and forms the basis for many learning algorithms. It is known that large training sets can be an important requirement for good model generalisation, but they can be a computational burden. The model's cost function often can be represented as the sum over the training examples of a per-example loss function. For instance the negative conditional log-likelihood of training data can be expressed as

$$J(\theta) = \mathbb{E}_{x,y\sim\hat{p}_{data}} L(x, y, \theta) = \frac{1}{m}\sum_{i=1}^{m} L(x^{(i)}, y^{(i)}, \theta), \tag{3.10}$$

where L is the per-example loss: $L(x^{(i)}, y^{(i)}, \theta) = -\log p(y^{(i)}|x^{(i)}; \theta)$. Gradient descent for the per-example cost functions requires

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta L(x^{(i)}, y^{(i)}, \theta). \tag{3.11}$$

The computational cost to compute this gradient is $O(m)$. So the time to compute grows in a linear manner relative to size of the input data set. This can be a problem with a very large number of training examples. The insight from SGD is that the function's gradient can be approximated using a subset of training examples [42].

At each step of the algorithm a mini-batch of examples $\mathbb{B} = \{x^1, ..., x^{m'}\}$ is uniformly randomly selected from the training set. Normally the size of the

mini-batch m' is kept constant and is relatively small when compared to the size of the training set. So the estimate of the gradient g based of the mini-batch m' can be written as

$$g = \frac{1}{m'} \sum_{i=1}^{m'} \nabla_\theta L(x^{(i)}, y^{(i)}, \theta). \tag{3.12}$$

With the estimated gradient the SGD algorithm can move toward a minimum with:

$$\theta \leftarrow \theta - \epsilon g, \tag{3.13}$$

where again ϵ is the specified learning rate of the algorithm.

Gradient descent can be characterised as slow or inconsistent in some cases, but it works well enough to find an acceptably low value for the cost function allowing it to be useful for many learning algorithms, even if the low cost function value is not a minimum of any kind [42].

3.2 Feed forward neural networks (FFNNs)

Feed forward neural networks (FFNNs), often also referred to as multi-layer perceptrons (MLPs) are standard in the realm of machine learning. The goal of an FFNN is to approximate some function $y = f(x)$, with function parameters θ. The model learns the values of θ that will result in the best overall function approximation. FFNN models are named as such due to the nature of how values flow through the network in a single direction, that is from input x, through the intermediate computations that define the function under review f, to the model's expected output denoted as y.

These models are referred to as networks, as they are normally defined as a linear composition of functions. For example three functions f^1, f^2 and f^3 can be composed into the form of $f(x) = f^3(f^2(f^1(x)))$, where f^1 is the first layer, f^2 is the second layer and f^3 is the third layer in the neural network. The length of the chain, or the number of composed functions is referred to as the depth of the neural network. Deep networks have multiple layers. The first layer of the network is known as the input layer and the last is known as the output layer. The layers between the input and output are referred to as the hidden layers of the network.

These networks are called neural networks as they are loosely based on principles of neuroscience. Each layer is vector valued and each element of the vector can be interpreted as a neuron. The neurons in a layer act together in a vector to scalar functional manner. The element is neuron-like as it computes its activation based on inputs from many other neurons. It can be helpful to reason about the operation of such models from a biological and neuroscientific viewpoint, but they do not map one-to-one to actual representations of brain-like activity. These models are merely tools used for statistically generalised function approximation tasks [42]. A simple FFNN can be seen in Figure 3.2.

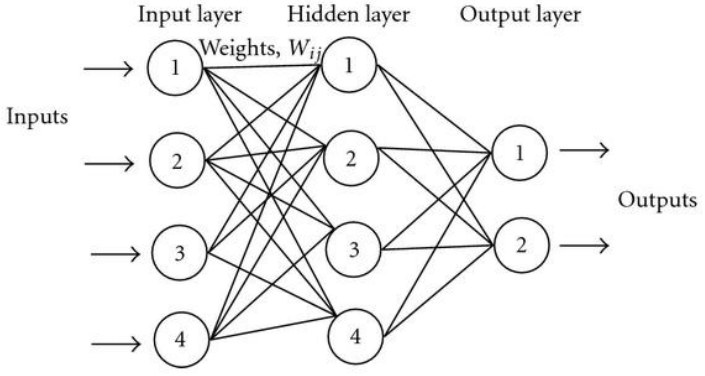


Figure 3.2: Graph of a simple FFNN.

3.2.1 Model architecture

A model's architecture refers to the overall structure of the network. For instance, the number of layers and the number of elements per layer affect the network architecture. A layer is a function of the layer that precedes it. The first layer can be described as:

$$h^{(1)} = g^{(1)}(W^{(1)}x + b^{(1)}). \tag{3.14}$$

The second layer can be described in a similar fashion:

$$h^{(2)} = g^{(2)}(W^{(2)}h^{(1)} + b^{(2)}), \tag{3.15}$$

and so on until all layers in the model have been described. In this representation, g is an activation function. W contains the weights, a set of values learnt by the model in the task to approximate y. The weights of the model are often randomly initialised, to break issues related to symmetry. Lastly parameter b represents the bias, a constant which is used to shift the function.

3.2.1.1 Universal approximation theorem

The universal approximation theorem states that an FFNN with a linear output layer and at least one hidden layer with any squashing activation function can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units. The derivatives of the FFNN can also approximate the derivatives of the function arbitrarily well [42; 51].

This implies that for any function under review, there exists an FFNN that is representative, but there is no certainty that a particular training algorithm can learn the function. The theorem does not state how large this network will be, so in many cases deeper models can reduce the number of units required to represent the function and the associated error [42].

3.2.2 Input layer

The input layer is the first vector valued layer of the network. It serves as the input for the example to the network. The input layer can be viewed as passive as it does not modify the incoming data before relaying it to the next layer in the network.

3.2.3 Hidden layers

The hidden layers of a network are all those which exist between the input and output layer. Most hidden layers can be described as accepting a vector of inputs x, computing the affine transformation $z = Wx + b$ and applying the nonlinear function $g(z)$.

3.2.3.1 Rectified linear units (ReLUs)

The activation function $g(z)$ called the ReLU is $g(z) = \max\{0, z\}$. With this the output is zero across half of its domain, so the derivative is relatively high when it is active. The second derivative is zero, and the derivative of the rectifying operation is 1 everywhere that the unit is active, which makes the gradient direction more useful for learning than what it would be for other activation functions with second-order effects [42]. A graph of the ReLU function can be seen in Figure 3.3.

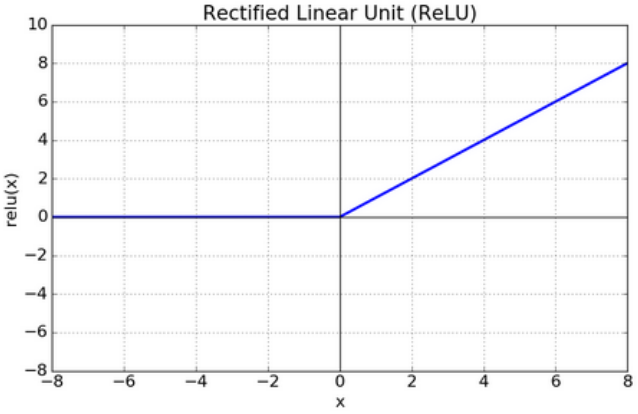


Figure 3.3: ReLU function.

3.2.3.2 Logistic sigmoid units

The logistic sigmoid function is defined as

$$g(z) = \frac{1}{1 + e^{-z}}. \tag{3.16}$$

These sigmoidal units tend to saturate across most of their domain, and are only sensitive to input near 0. These properties of the sigmoidal function can make it difficult for gradient based learning, and due to this it is advised to not to use them as hidden units [42]. A graph of this function can be seen in Figure 3.4.

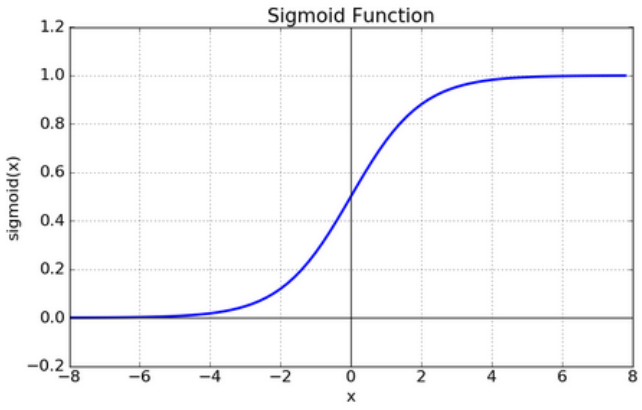


Figure 3.4: Sigmoid function.

3.2.3.3 Hyperbolic tangent units

The hyperbolic tangent activation function is defined as $g(z) = \tanh(z)$. The shape of this function is similar to the logistic sigmoid. A graph of the hyperbolic tangent function can be seen in Figure 3.5.

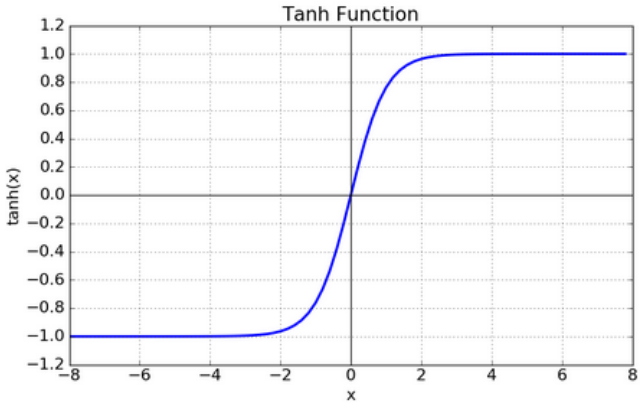


Figure 3.5: Hyperbolic tangent function.

3.2.4 Output layer

The cost function is an important aspect of any machine learning algorithm, and is tightly coupled to the model's output layer. In many cases the cost function leverages off the principle of maximum likelihood, which implies that the cost function is the negative log-likelihood, or the cross-entropy between the training data and the model's distribution. It is used to indicate the effectiveness of the model under review, and can be expressed by

$$J(\theta) = -\mathbb{E}_{x,y\sim\hat{p}_{data}} \log p_{model}(y|x). \tag{3.17}$$

The role of a model's output layer is to provide additional feature transformation in order to complete the learning task at hand. Assuming that an FFNN has a set of hidden features defined by h where $h = f(x;\theta)$ a few output units are described next.

3.2.4.1 Linear units for normal distributions

The linear output is based off an affine transformation with no nonlinearity and given h will produce a vector $\hat{y} = Wh + b$. Often a linear layer is used to generate the mean of a conditional normal distribution $p(y|x) = \mathcal{N}(y;\hat{y},I)$. Maximising the log-likelihood is equivalent to minimising the mean squared error [42].

3.2.4.2 Sigmoid units for Bernoulli distributions

Classification is an important domain within machine learning, where many tasks require the prediction of a binary variable y given x. Using the maximum-likelihood method, a Bernoulli distribution can be defined. The output of a sigmoid unit is described by $\hat{y} = \sigma(Wh + b)$ where σ is the logistic sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. So first the value z is computed as $z = Wh + b$, then z is passed to the sigmoid function to calculate a probability [42].

This can be used to describe an unnormalised probability distribution over y using z denoted as $\tilde{P}(y)$. With appropriate arithmetic a suitable distribution can be obtained. Assuming that the unnormalised log probabilities are linear, exponentiation yields unnormalised probabilities [42]. Normalising this allows for a Bernoulli distribution dependent on the sigmoidal transformation of z.

$$\log \tilde{P}(y) = yz \tag{3.18}$$

$$\tilde{P}(y) = e^{yz} \tag{3.19}$$

$$P(y) = \frac{e^{yz}}{\sum_{y\prime=0}^{1} e^{y\prime z}} \tag{3.20}$$

$$\tilde{P}(y) = \sigma((2y-1)z). \tag{3.21}$$

In this case the z variable, defining a distribution over binary variables is referred to as a logit. The maximum likelihood loss function becomes

$$J(\theta) = -\log P(y|x) \tag{3.22}$$
$$= -\log \sigma((2y-1)z). \tag{3.23}$$

3.2.4.3 Softmax units for multinoulli distributions

The softmax function can be used to represent a discrete variable over n different states. This can be seen as a generalisation of the sigmoid function but instead of a single variable, a vector $\hat{y}$ is needed where $\hat{y} = P(y = i|x)$. In this case each element of $\hat{y}$ is either 0 or 1 and the sum of $\hat{y}$ must equal 1. Similar to the Bernoulli distribution, let $z = Wh+b$ where $z_i = \log \tilde{P}(y = i|x)$. Exponentiation and normalisation of z yields

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}. \tag{3.24}$$

Maximising $\log P(y = i|z) = \log \text{softmax}(z)_i$ yields

$$\log \text{softmax}(z)_i = z_i - \log \sum_j e^{z_j}. \tag{3.25}$$

Maximum likelihood will lead the model to learn parameters which the softmax will use to predict the fraction of occurrences for each outcome in the training set:

$$\text{softmax}(z(x;\theta))_i \approx \frac{\sum_{j=1}^{m} 1_{y^{(j)}=i, x^{(j)}=x}}{\sum_{j=1}^{m} 1_{x^{(j)}=x}}. \tag{3.26}$$

3.2.5 Network propagation

An FFNN accepts an input x at its input layer which then propagates through the hidden layers to the output layer where y is computed, which can be used to calculate the scalar cost $J(\theta)$. This process is called forward propagation. Once the cost has been established, its result needs to be communicated to the weights of the network layers in order for them to be updated accordingly, using gradient descent. This is done by using the back propagation method, or simply backprop [42]. Following Rumelhart [52], assuming there exists a simple FFNN with input, hidden and output layers, the input x_j to unit j is a linear function of the outputs of y_i (the previous layer) of units connected to unit j and the weights between the two defined as w_{ji}:

$$x_j = \sum_i z_i w_{ji}. \tag{3.27}$$

The nonlinear output of unit y_j is

$$y_j = \frac{1}{1+e^{-x_j}}, \tag{3.28}$$

which is the sigmoid function. The goal is to find a set of weights that gives some input x the output of the network y such that it is close to the expected output. The total loss or error L for all actual and expected output is determined as

$$L = \frac{1}{2}\sum_{c}\sum_{j}(y_{j,c} - d_{j,c})^2, \tag{3.29}$$

where c is the index for input-output pairs, j is the actual state of the output unit and d is the expected state. To minimise L using gradient descent the partial derivative of L relative to each weight is required:

$$\frac{\partial L}{\partial y_i} = y_i - d_j. \tag{3.30}$$

Using the chain rule yields

$$\frac{\partial L}{\partial x_j} = \frac{\partial L}{\partial y_j}\frac{dy_j}{dx_j}. \tag{3.31}$$

Differentiating the sigmoid function and substituting accordingly produces

$$\frac{\partial L}{\partial x_j} = \frac{\partial L}{\partial y_j}y_j(1 - y_j). \tag{3.32}$$

The effects of the weights are as follows:

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial x_j}\frac{\partial x_j}{\partial x_{ji}} \tag{3.33}$$

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial x_j}y_i. \tag{3.34}$$

The effect of unit i on j is

$$\frac{\partial L}{\partial x_j}\frac{\partial x_j}{\partial y_i} = \frac{\partial L}{\partial x_j}w_{ji}. \tag{3.35}$$

For unit i:

$$\frac{\partial L}{\partial y_i} = \sum_{j}\frac{\partial L}{\partial x_j}w_{ji}. \tag{3.36}$$

With this the partial derivative of the error relative to the output can be calculated for the last hidden layer, and can be used to update the weights of that layer. A simple use of gradient descent would be to update the weights by a portion of the accumulated partial derivative of error/loss relative to the weight:

$$\Delta w = -\epsilon\frac{\partial L}{\partial w}. \tag{3.37}$$

An alternative method may be

$$\Delta w = -\epsilon \frac{\partial L}{\partial w(t)} + \alpha \Delta w(t-1), \tag{3.38}$$

where t is the count of iterations through the network and α is a decay factor between 0 and 1. This process can be applied successively to compute the weight updates for all the layers of the network.

The second term in the above equation can be referred to as the momentum. It is designed to accelerate the learning process especially in the face of high curvature, small, or noisy gradients. The hyperparameter α is responsible for magnitude of the contribution offered by the previously computed gradients.

3.3 Recurrent neural networks (RNNs)

Recurrent neural networks (RNNs) form a class of networks for processing input data that can be defined as sequential. RNNs are said to operate on an input sequence of vectors $x(t)$ with the time step index t, ranging from 1 to τ. These input sequences can vary in length depending on the network implementation. The ability to share parameters across the model is what makes RNNs possible. This allows for the model to be applied to inputs of different lengths and generalise across those inputs. In the case of a specific parameter per time index it would not be possible to account for input sequence lengths not observed in the training step.

3.3.1 Structure of RNNs

A computational graph formalises the structure of a set of computations. In a computational graph, variables are represented as nodes and operations as edges. A simple example of such a graph can be seen in Figure 3.6.

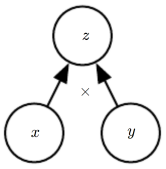


Figure 3.6: A simple graph representation of the function $z = xy$ [42].

Consider the following dynamical system, defined by function f:

$$s^{(t)} = f(s^{(t-1)}, x^{(t)}; \theta), \tag{3.39}$$

where the current state s at time t depends on the previous state at time $t-1$. x is an input at time t and θ is some parameter. This pattern is repeated for the entire sequence. As many functions can be represented as an FFNN, this one can be interpreted as an RNN. So the hidden units of an RNN can be expressed as

$$h^{(t)} = f(s^{(t-1)}, x^{(t)}; \theta). \tag{3.40}$$

With this, an RNN can be trained to predict a value based on a past sequence of inputs up to t. $h^{(t)}$ serves as a summary of the task-relevant properties of the input sequence. It may not capture all aspects of the past, as the sequence can be of arbitrary length and $h^{(t)}$ has a fixed size [42].

The process of unfolding maps the circuit graph to a computational graph. An example of this process can be seen in Figure 3.7. This unfolded graph has a size that is dependent on the length of the sequence, and can be expressed as:

$$h^{(t)} = g^{(t)}(x^{(t)}, x^{(t-1)}, x^{(t-2)}, ..., x^{(2)}, x^{(1)}) \tag{3.41}$$

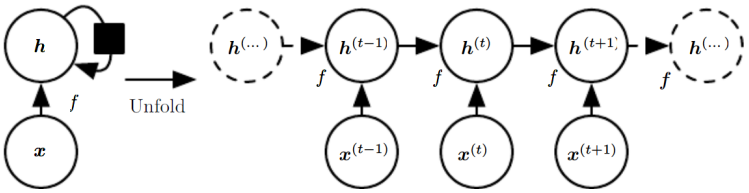


Figure 3.7: Left: A circuit diagram where the square indicates a 1 time-step delay. Right: An unfolded computational graph, where each node is associated with 1 unit of time [42].

Function $g^{(t)}$ takes the sequence as an input and produces the current state. The unfolded recurrent structure allows for the factorisation $g^{(t)}$ into repeated applications of function f which is beneficial as it does not depend on the sequence length and can be applied at each time step. With this it is possible to define a function f that can operate on any sequence length. This allows model generalisation for sequence lengths that were not observed in the training set.

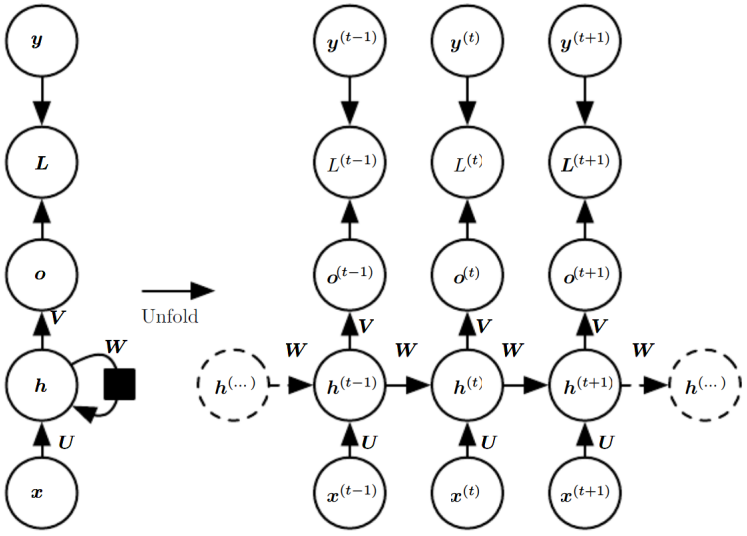


Figure 3.8: Computational graph of an RNN that maps input sequence x to output o. Loss L is the difference between each o and target y. The RNN has input to hidden connections parametrised by weight matrix U, hidden-to-hidden recurrent connections parametrised by weight matrix W, and hidden-to-output connections parametrised by weight matrix V [42].

Using Figure 3.8 as a guide, forward propagation for RNNs can be defined. It begins with a specification of the initial state $h^{(0)}$, and then the update equations:

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)} \tag{3.42}$$

$$h^{(t)} = \tanh(a^{(t)}) \tag{3.43}$$

$$o^{(t)} = c + Vh^{(t)} \tag{3.44}$$

$$\hat{y}^{(t)} = \sigma(o^{(t)}) \tag{3.45}$$

for each time t from 1 to τ, where o is the output, b and c are bias vectors, and U, V and W are weight matrices. In this case the hyperbolic tangent is the activation function. The total loss for an input sequence x relative to an expected sequence y is the sum of the losses over all time steps τ:

$$L(\{x^{(1)}, ..., x^{(\tau)}\}, \{y^{(1)}, ..., y^{(\tau)}\}) = \sum_{\tau} L^{(t)}, \tag{3.46}$$

$$\sum_{\tau} L^{(t)} = -\sum_{\tau} \log p_{model}(y^{(t)} | \{x^{(1)}, ..., x^{(\tau)}\}). \tag{3.47}$$

With this, the gradient of the loss can be computed with respect to the model parameters. Due to the sequential nature of RNNs, their run time is $O(\tau)$ as each time step depends on the last. Memory usage is also $O(\tau)$ as each state computed in the forward pass is stored to be used in the backward pass. Performing back propagation on an RNN is known as back propagation through time (BPTT). BPTT is applied to the unfolded RNN to update the weights accordingly. Although recurrence can make these models very useful, the time and memory requirements can make them difficult to train.

3.3.2 Gated RNNs

A difficulty arises when trying to learn long term dependencies in RNNs, known as the vanishing or exploding gradient problem [53]. A vanishing gradient can cause the weights to go largely unchanged and an exploding one can cause them to fluctuate too erratically for any meaningful learning to occur. This degrading signal is due to the chain rule in back propagation. Multiplying many small numbers together for an update will lead to zero and the inverse occurs with large numbers.

Methods exist to mitigate this issue, such as skip connections where connections are added between the past and present states, and leaky units with linear self-connections that act like a running average for past, observed states by using constant weights. A gated RNN can be interpreted as an evolution of the idea behind leaky units as they allow the weights to change over time steps, which in turn allows for old information to be discarded if not required by the sequence [42].

3.3.2.1 Long short-term memory (LSTM)

The use of a self-connection to allow for the gradient to have a meaningful effect in the LSTM model was made by Hochreiter and Schmidhuber [54]. This was extended by making the self-connection weight context dependent [55] so its time scale can be changed dynamically based on the input sequence. LSTM networks are made up of LSTM cells which contain an internal recurrence in addition to the RNN recurrence. The gating element controls the flow of information thought units. An example of the LSTM cell can be seen in Figure 3.9

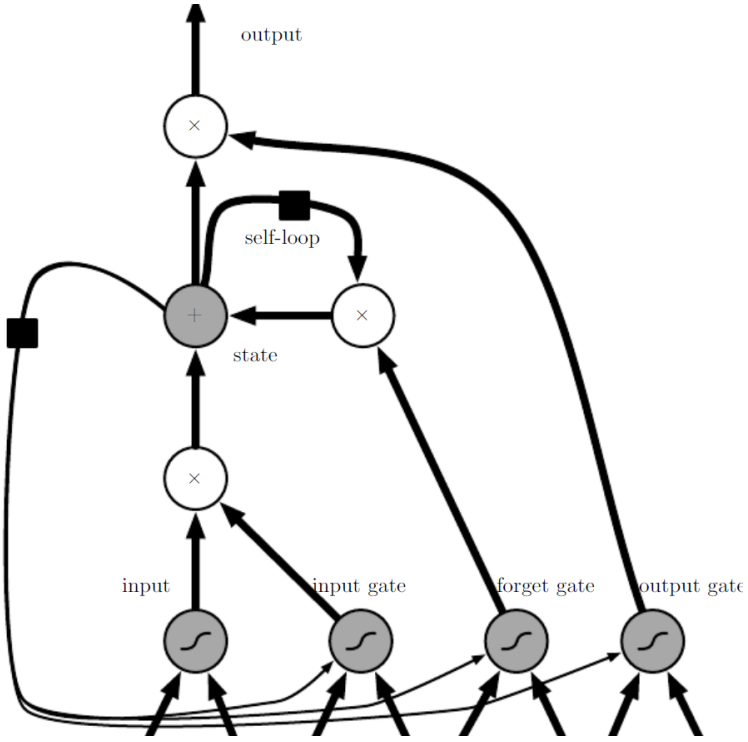


Figure 3.9: LSTM cell. Cells are connected recurrently to each other. An input feature is computed with a normal unit. Its value can be accumulated into the state if the sigmoidal input gate permits it. The state unit has a linear self-loop whose weight is controlled by the forget gate. The output of the cell can be controlled by the output gate. The state unit can also be used as an extra input to the gating units [42].

The state unit $s_i^{(t)}$ has the linear self-loop controlled by a forget gate $f_i^{(t)}$ that sets this weight to a value between 0 and 1 using a sigmoid:

$$f_i^{(t)} = \sigma(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}), \tag{3.48}$$

where t is the time step, i the cell, $x^{(t)}$ is the current input and $h^{(t)}$ is the current hidden layer vector containing the outputs of the cells. b^f, U^f, W^f are

the bias, input and recurrent weights for the forget gates. The internal state is updated with $f_i^{(t)}$:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}), \tag{3.49}$$

where b, U, W are the bias, input and recurrent weights for the LSTM cell. The input gate unit $g_i^{(t)}$ is also computed with a sigmoid:

$$g_i^{(t)} = \sigma(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}), \tag{3.50}$$

The output $h_i^{(t)}$ of the cell can also be controlled via the output gate $q_i^{(t)}$:

$$h_i^{(t)} = \tanh(s_i^{(t)}) q_i^{(t)}, \tag{3.51}$$

$$q_i^{(t)} = \sigma(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}), \tag{3.52}$$

where b^o, U^o, W^o are the bias, input and recurrent weights. LSTMs have been shown to learn long-term dependencies more effectively than the simple RNNs [54; 56].

3.3.2.2 Gated recurrent unit (GRU)

Another option is the GRU [57; 58]. In a GRU a single gating unit is responsible for forget and update factors associated to the state unit. This makes the overall structure of the cell a bit simpler. The update equation is:

$$h_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + (1 - u_i^{(t-1)}) \sigma(b_i + \sum_j U_{i,j} x_j^{(t-1)} + \sum_j W_{i,j} r_j^{(t-1)} h_j^{(t-1)}) \tag{3.53}$$

where u is the update and r is the reset [42]:

$$u_i^{(t)} = \sigma(b_i^u + \sum_j U_i^u x_j^{(t)} + \sum_j W_{i,j}^u h_j^{(t)}), \tag{3.54}$$

$$r_i^{(t)} = \sigma(b_i^r + \sum_j U_i^r x_j^{(t)} + \sum_j W_{i,j}^r h_j^{(t)}). \tag{3.55}$$

3.4 Convolutional neural networks (CNNs)

CNNs are networks designed to operate on input data with a defined structure, such as a two-dimensional image or time series sequence. A convolution is a linear mathematical operation. A convolutional neural network is a network that makes use of such an operation in at least one of its layers. A representation of a CNN with more than one layer can been observed in Figure 3.10.

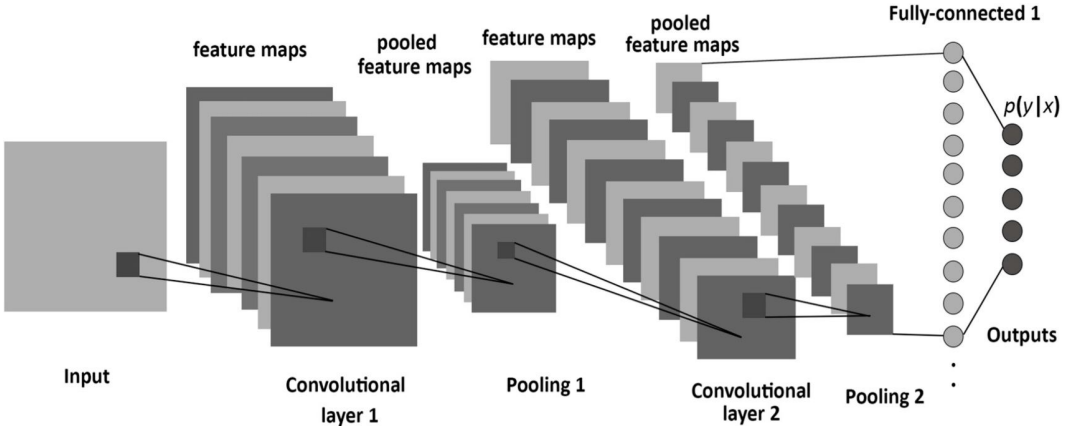


Figure 3.10: Representation of a CNN with convolutional, pooling and fully connected layers [42].

3.4.1 Convolution operation

The convolution operation is an operation on two functions, giving rise to a third function. This third function can be interpreted as an altered version of one of the input functions. To demonstrate this consider two functions, one called $x(t)$ to determine a particle's location at time t and $w(\alpha)$, a weighting function to emphasise recent measurements where α represents the location measurement age. An estimate of the particle's location can be described by

$$s(t) = \int_{-\infty}^{\infty} x(a)w(t-a)da, \tag{3.56}$$

and is a convolution of the two initial functions. The convolution is often denoted with an asterisk:

$$s(t) = (x * w)(t). \tag{3.57}$$

The first argument, function x is referred to as the input, the second, w, is the kernel and the output is called the feature map. Discrete convolution can be defined as

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a). \tag{3.58}$$

In a machine learning setting the input data and kernel are typically multi-dimensional. Convolutions can be done over more than one dimension, for example:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{3.59}$$

The discrete convolution can be interpreted as matrix multiplication. In machine learning, the algorithm will learn the required values of the kernel in order to produce the desired output y given some input x.

In a more traditional FFNN, each input unit interacts with each output unit through various applications of matrix multiplication. In a CNN, the interaction tends to be sparse, by having a smaller kernel than input. A smaller kernel is often sensible as the meaningful characteristics of the input are a fraction of the size of the total input. The CNN leverages the idea of parameter sharing, as the convolution with the kernel uses a single set of weights to evaluate each position of the input. A small kernel with shared parameters can improve the runtime performance of the CNN model when compared to other network architectures with a similar input [42].

3.4.2 Pooling

CCNs often contain a component known as a pooling function which further refines the output of the layer. In this case the layer performs convolutional and nonlinear activations before pooling. In a CNN the nonlinear activations are sometimes referred to as detectors. Pooling can be interpreted as a smoothing operation over some space of outputs. This is done in an attempt to make the model invariant to small input translations. So pooled outputs do not change for small changes in input. Examples of pooling include a simple average of a rectangular neighborhood, or max-pooling, which yields the maximum value of output over some rectangular neighbourhood [59]. As pooling summarises the output over some neighbourhood, it is viable to use fewer pooling units than detector units, which can lead to a more computationally efficient model requiring less run time and memory resources.

3.5 Financial instruments

Financial instruments are tradable assets, allowing for the efficient transfer of global capital. These entities can be real or virtual, representing a legal agreement involving equities, debt or foreign exchange. Generally they can be broken down into two classes: cash and derivative instruments. Cash instruments are easily transferable securities, which value is determined by supply and demand in the market. Derivative instruments are based on underlying components such as the price of an asset or value of an index [60].

3.5.1 Commodity

As a basic interchangeable good used in commerce, it often serves as an input to the production of goods and services. Commodities can be traded on an exchange, but must meet quality standards. The quality of the good can differ slightly from one producer to another but is recognised as homogeneous if it meets the minimum defined standard often rendered to as the basis grade. Some typical commodities include oil, gold and grain. In recent years the definition has expanded to include financial products such as foreign currencies [61].

3.5.2 Exchange rate

An exchange rate is the value of a nation's currency relative to another. An exchange rate has a base currency and a counter currency. In a direct quotation, the foreign currency is the base currency and the domestic currency is the counter currency. In an indirect quotation, the domestic currency is the base currency and the foreign currency is the counter currency [62].

3.5.3 Spot price

Spot price is the current price one can expect to pay for a given asset, for immediate delivery in a marketplace. They are time and place dependent but overall they are relatively uniform globally [63].

3.5.4 Derivative

A derivative is a financial security, where a security is a fungible, negotiable financial instrument that holds some value [64]. The derivative can also be interpreted as a contract between parties where the contract's value is based on an underlying asset or group of assets, and its value changes relative to the changes in the value of the underlying assets. Common assets which are the basis of derivatives include bonds, commodities, currencies, and interest rates [65].

3.5.5 Future contract

Related to the derivative, a future contract is a standardised agreement to buy or sell a particular instrument at a predetermined price and time in the future. The buyer has an obligation to acquire the underlying asset when the future contract expires and the seller is required to produce the asset at the contract expiration date. Future contracts are traded on a futures exchange [66].

3.5.6 Summary

Oil prices, exchange rates and related future contracts serve as input features for examples that will be passed to a neural network model, which is trained with a supervised learning approach to forecast the BFP. The sources of, and format of this input data is discussed further in Section 4.2.

Chapter 4

Methods

This Chapter highlights the various hardware, software and data components used in the development of a number of machine learning models to forecast the South African BFP. The Chapter also describes the experiments devised to evaluate the performance of these models on the forecasting task, given different input criteria and network topologies.

4.1 Tools

This Section is a description of a number of programming tools and libraries used to create the various machine learning models as well as the mechanism for the storage and capture of data sets. The hardware upon which the experimental models were executed is also discussed.

4.1.1 Ruby and Rails

Ruby is an interpreted, dynamically typed, multi-paradigm and object-orientated programming language created by Yukihiro Matsumoto. It first appeared in 1995. Ruby is a high-level language written in C and is the basis for the web programming framework known as Rails [67].

Rails is a popular web framework built on the Ruby programming language which bundles several standard operations such as Hyper Text Markup Language (HTML) templating and database management. It allows for rapid development following a convention over configuration methodology. Some major applications built with Rails include the offerings from Airbnb and SoundCloud [68].

A simple Rails application was created to manage the input and expected output data that would be required by the proposed models. This application allowed data to be uploaded via Character Separated Value (CSV) files and

retrieved via Application Programming Interface (API) requests to endpoints supported by the data providers. Data providers are discussed in Section 4.2.

4.1.2 SQL and SQLite

Structured Query Language (SQL) is an ISO/IEC standard which describes the grammar of a programming language used to operate and retrieve data from a database application [69]. Popular variants of this standard include T-SQL (Microsoft) or PL-SQL (Oracle).

SQLite is a popular self-contained, server-less and transactional SQL database engine [70]. SQLite was used as a non-volatile data store for the data managed by the Rails web application. The supporting code for the various models pulls the required input and expected output data from the SQLite database and provides it to the model under review.

4.1.3 Python libraries

Python is a popular interpreted, dynamically typed and object-orientated programming language created by Guido van Rossum and first released in 1990 [71]. Python is similar to the Ruby programming language, and is also implemented in C. Python is popular in a number of domains including the web (Django, Flask), networking (Twisted, Asyncio) and machine learning (TensorFlow, Torch, Theano).

Statsmodels is an open-source package that provides a wide range of functionality associated to the estimation of many different statistical models. Functionality includes the ability to perform statistical tests and statistical data exploration. It can also provide an extensive list of result statistics for estimators under review [72].

TensorFlow is an open-source Python library designed for a broad range of high performance numerical computational tasks. Originally developed at Google Brain, it comes with strong support for machine and deep learning. It is a popular machine learning package used by established companies such as AMD, Uber and Dropbox [73]. TensorFlow supports modelling operations to be run on CPUs and GPUs.

Keras is a user-friendly and modular, high-level neural network API. It runs on top of computational engines such as TensorFlow and Theano. Keras is focused on providing functionality that allows for rapid prototyping and experimentation [74].

4.1.4 Git and GitHub

Git is an open-source distributed version control system [75] which was used to store all the code associated to the Rails data management application, SQLite database and Python code for all models. GitHub is an on-line development collaboration platform which implements the Git version control system [76]. All code and data is hosted on GitHub.

4.1.5 Digital Ocean

The Rails application to collect and manage data ran on a single Digital Ocean droplet (compute instance). Digital Ocean is a popular Infrastructure As A Service (IAAS) computing platform for hosting and scaling applications [77], similar to Amazon Web Services in their offering.

4.1.6 Hardware

The various models created and reviewed were run on a 2015 Apple Macbook Pro with a 2.2 GHz i7 processor and 16GB of 1600MHz RAM. The models were configured to train and execute on the machine's CPU.

4.2 Data sources

Input and expected output data for training and evaluating the models was obtained from a number of publicly available sources ranging from the South African Department of Energy to an on-line exchange rate service.

4.2.1 South African Department of Energy

The monthly BFP and the full unleaded 95 fuel price was obtained from the South African Department of Energy [78]. The current and historic data is available in a number of formats including HTML, PDF and CSV files. The required data was collated, parsed and then uploaded to the Rails data management application. The fuel data used in the experiments ranged from April 2003 to March 2018. The BFP represents the desired target result of the models, given the supervised learning nature of presented NNs.

4.2.2 Quandl

Quandl is an on-line financial and economic data set aggregation service [79], allowing for easy integration with a simple API. Using the Quandl API, the following data sets where obtained:

- daily European price of Brent crude oil;
- daily Brent crude oil future contract;
- daily US Dollar/South African Rand exchange rate future contract.

Functionality to call this API was integrated into the Rails data management application. The data obtained ranged from the beginning of December 2002 to end of February 2018. The oil price and future contracts daily time series sequences serve as input features to the models for forecasting the BFP.

4.2.3 Open Exchange Rates

Open Exchange Rates (OER) provides a robust and simple JSON (JavaScript Object Notation) API with live and historical foreign exchange rates for over 200 physical and digital currencies. OER blends data algorithmically from multiple sources to provide a fair and unbiased value to requested currencies [80]. This service was used to obtain the daily closing exchange rate between the US Dollar and South African Rand. This was integrated into the Rails data management web application. These exchange rates also served as a model input sequence.

4.3 Experiments

The fuel pricing data collected for this evaluation spans from April 2003 to March 2018. This coincides with the South African governmental financial year, with a total of 180 observed BFP data points. It is noted that this is a relatively small data set. As discussed in Section 1.4, the current BFP formula replaced the In Bond Landed Cost (IBLC) method in 2003. The BFP data is on a monthly level whereas the oil pricing and exchange rates are at a daily level. The daily data is for trading days only, so one month worth of daily data will be represented as 4 work weeks, and 20 days is equivalent to a calendar month. The goal is to forecast the next BFP given some sequence of exchange rate and oil price time series data.

All data is preprocessed by a feature scaling step to help the objective functions to converge to an optimal value. The scaling method used was the min-max normalisation approached defined as:

$$x' = \frac{x - \min(X)}{\max(X) - \min(X)}, \tag{4.1}$$

where X is the set containing all x values and x' is the scaled version of x.

The effectiveness of various models is assessed by determining the mean absolute percentage error (MAPE) of the test set. The MAPE can be described as

$$MAPE = \frac{100}{n} \sum_{t=1}^{n} \left| \frac{A_t - F_t}{A_t} \right|, \tag{4.2}$$

where n is the number of examples in the set, A_t is the actual value and F_t is the forecasted value. An alternative to the MAPE could be the Root Mean Squared Error (RMSE). The MAPE is selected as it can be easy to reason about different model performance using a percentage value. As suggested in Chapter 3, the available data is split into a training, validation and test set. The training set comprises of 62% of all data, with the validation and test set each comprising of 19%. 19% of the total BFP data set is equivalent to 30 months or 2.5 years. These three sets of data are used to assess all the models presented below and were populated by randomly selecting from the total set of 180 data points. These data sets are referred to as the common data sets.

Presented below is a description of the models and experiments undertaken to assess the viability of using machine learning techniques to forecast the South African BFP. The results of these experiments are presented in Chapter 5.

4.3.1 Regression model

In an attempt to replicate what has been done before, using the Statsmodels Python library, a linear regression model was created for various sequence lengths of exchange rate and oil price data. The sequence lengths ranged from 5 to 60 days with increments of 5 days. These various sequences of exchange rate and oil prices where then reduced to single averages. Models are fitted and tested with the common data sets as defined above. The purpose of this is to gain some understanding on how the length of the sequence affects the MAPE of the model and how accurate an existing forecasting method can be.

The literature has highlighted that regression models can provide a good base case to benchmark more advanced methods. More details on this and the format of the regression model are presented in Section 2.1.3.

4.3.2 FFNN models

The FFNN model is discussed in detail in Section 3.2.

4.3.2.1 Improving on the regression model

Using the Keras and TensorFlow Python packages, a deep regulated FFNN was created which accepts the oil and exchange rate time series data as an input to forecast the BFP. From the literature we know that deep networks can be more effective than shallow networks. This experiment is an attempt to improve upon the results of the above regression model by building a deep FFNN and using an optimal sequence length of time series data. This model is fitted, validated and tested using the common data sets.

4.3.2.2 Using futures to reduce error of FFNN

Using the same FFNN as above, the effectiveness of adding the oil and exchange rate future contracts to the input time series data is assessed by its ability to reduce the MAPE of the model.

4.3.2.3 Using lead time to increase forecasting range

In the above cases, it is assumed that all historic data points are available to forecast the BFP. This is not very helpful as the model then provides no forecasting range. In this experiment, the aim is to investigate the effect of lead time on the MAPE by training various FFNNs that operate on lead times increasing in steps of 5 days from 0 to 30 days. The lead time is defined as the number of days before the BFP is expected to change.

4.3.3 RNN models

The RNN model and associated cells are discussed in detail in Section 3.3.

4.3.3.1 Using LSTM and GRU cells

Similar to the FFNN network, an RNN is created with Keras and TensorFlow. From the literature the RNN is well suited for operating on input time series data due to its internal recurrence. The RNN was evaluated with both LSTM and GRU cells for their effectiveness in forecasting the BFP. Also noted in the literature is the complexity of the RNN, which can lead to an increased consumption of resources. These RNNs are again trained, validated and tested using the common data sets.

4.3.3.2 Using less data

Although there are only a total of 180 BFP data points available, in this experiment it will be halved to determine the effect on the RNN's ability to correctly forecast the BFP. The best performing RNN and cell combination from the previous experiment are used for this trial.

4.3.4 CNN models

The CNN model is discussed in detail in Section 3.4.

4.3.4.1 Evaluating performance of CNN

Again, with Keras and TensorFlow a CNN is created to be evaluated for its ability to forecast the BFP. At this time according to the literature, this application of the CNN topology has not been extensively studied but it can provide valuable results due to the structured nature of the input time series data.

4.3.4.2 Testing on latest BFP observations

As discussed, the three common data sets (training, validation, testing), are selected randomly from the total amount of BFP data available from the financial years covering the period between 2003 and 2018. As the goal is to forecast the next BFP, it is potentially more relevant to populate the test set with the latest 2.5 years of BFP data. In this case a new test set is comprised of this latest period of BFP data and a new training and validation set is populated by randomly selecting from the remaining data set. This is done to determine if the MAPE improves when the model is trained and validated on older data, and then assessed on the latest set of observations.

4.3.5 Optimal BFP forecasting model

In this last experiment, the attempt is to create an optimal model that can be used to forecast the BFP. This model is expected to provide the lowest test set MAPE by combining the best results observed in the prior experiments.

Chapter 5

Results

This chapter contains details relating to the results of the various experiments described in chapter 4. The results are presented in the same order as the experiments are described in that chapter. Before presenting each result a brief overview of the experiment will be given.

As described in Section 4.3, model performance in each result is measured in mean absolute percentage error (MAPE). This number is presented as a percentage value ranging between 0 and 100. The lower this value the better.

Chapter 4 highlights that 180 BFP data points represent a small data set which can lead to unstable results. Given this, each experiment ran a number of times and a representative average of model performance was recorded and presented below.

NN hyperparameters were initialised with sensible defaults and altered manually by periodically evaluating model performance on the validation set and updating as required.

5.1 Regression model

The first model created and assessed for its ability to forecast the South African BFP is a simple regression model. This model operates on an input of averaged exchange rate and oil price values. These averaged values are obtained from sequence lengths ranging from 5 to 60 days. This is a rough approximate to what has been done before and provides an initial benchmark for the time series forecasting task given to the various machine learning models. Further details for this initial benchmarking experiment can be found in Section 4.3.1. The findings associated to this experiment can be seen in Figure 5.1.

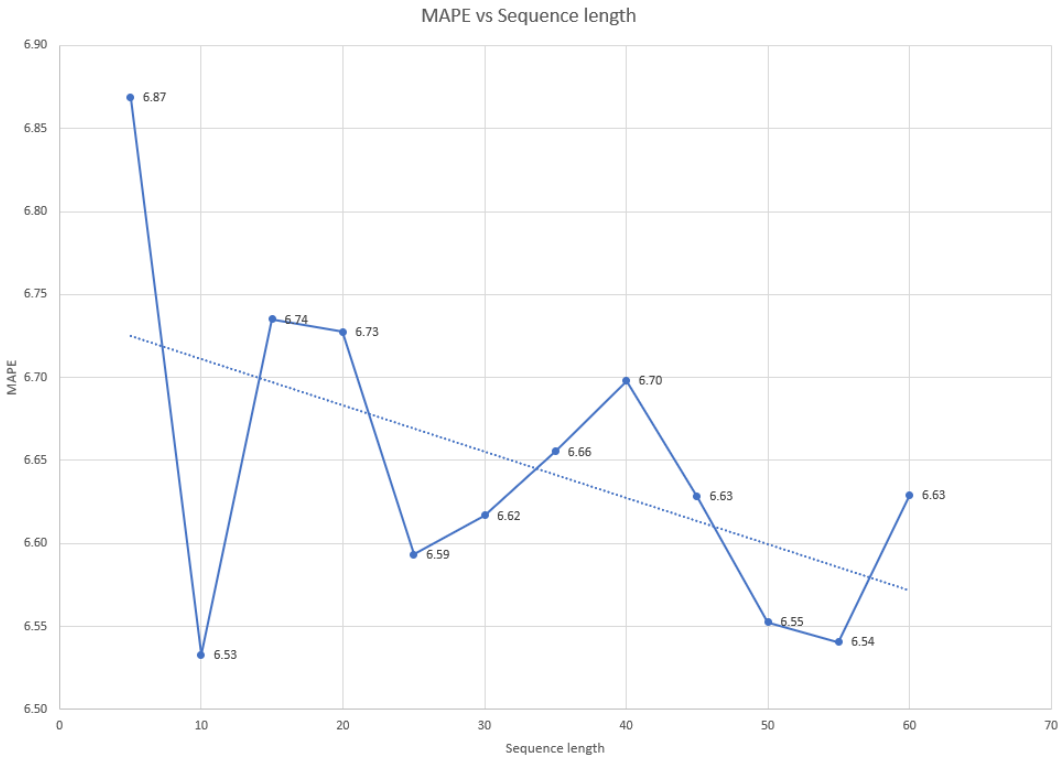


Figure 5.1: MAPE vs sequence length for various regression models fitted for sequences ranging from 5 to 60 days. These sequences represent the days until a change in the BFP.

From Figure 5.1, the latest 10 days of data before the BFP changes seem to perform relatively well, as do the latest 50 to 55 days. It must be noted that the difference in error between points in the sequence is relatively small but the overall trend in the results indicates that generally a longer sequence should yield a lower error when attempting to forecast the BFP. For all models the time to fit was less than 1 second. Knowing that generally the longer the sequence the lower the MAPE, a sequence length of 50 days will be used in the next experiments. 50 days corresponds to roughly 2.5 months.

5.2 FFNN model

The first deep machine learning model developed is an FFNN. The first experiment with this model is to determine whether it can match or improve upon the performance of the regression model as described in Section 5.1. The second experiment is to add the future contracts to the model input, to understand the effect on the model's performance when forecasting the BFP. The last experiment regarding the FFNN is to determine how increasing the lead time in the input sequence affects the model's performance measured by the MAPE. A more detailed description of the FFNN experiments is given in Section 4.3.2.

5.2.1 Improving on the regression model

The created deep regulated FFNN is summarised in Table 5.1, which also includes the hyperparameters chosen by evaluating the MAPE on the common validation data set, described in Section 4.3.

Table 5.1: Description of the FFNN.

(a) Network topology

Layer	Dimension	Activation
Input	130	ReLU
Dropout 1	130	
Hidden dense 1	250	ReLU
Dropout 2	250	
Hidden dense 2	130	ReLU
Dropout 3	130	
Hidden dense 3	60	ReLU
Dropout 4	60	
Hidden dense 4	30	ReLU
Dropout 5	30	
Output	1	Linear

(b) Network hyperparameters

Parameter	Value
Dropout rate	0.04
Epochs	50
Learning rate	0.01
Decay	1.0E-6
Momentum	0.5

The attempt to improve upon the results of the regression model, with a deep FFNN operating on a 50 data point sequence of exchange rate and oil prices was somewhat successful. The FFNN yields a MAPE of 6.31 which is slightly better than the 6.55 observed for the linear regression model with the same sequence length. The FFNN model was trained in approximately 18 seconds. This is rather long when compared to the linear regression model for a marginal increase in forecasting accuracy. It could be possible to reduce the training time of the FFNN by adjusting the hyperparameters, for instance increasing the learning rate and reducing the number of training epochs.

5.2.2 Using futures to reduce error of FFNN

The above FFNN model's ability to reduce the MAPE can be improved by adding the oil and exchange future contracts to the 50 point input sequence. With the same hyperparameters as above this model yields a MAPE of 5.02. This is a marked improvement over just using the oil and exchange rate. The model trained in approximately the same time as above (19 seconds).

5.2.3 Using lead time to increase forecasting range

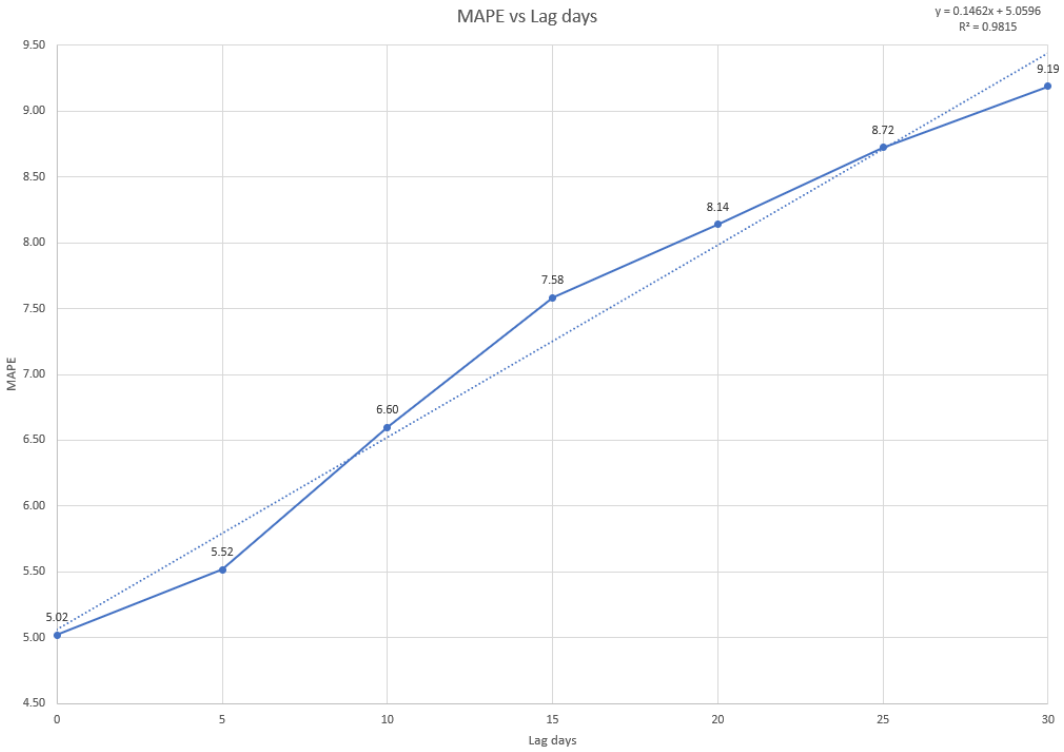


Figure 5.2: MAPE vs lead time in days for various lead lengths ranging from 0 to 30 days.

From Figure 5.2, it can be seen that an increase in the lead time has a proportional effect on the increase of the MAPE. 0 days of lead is not very useful as the fuel regulator has most likely set the upcoming price the day before it is expected to change. 5 days of lead has an increase in the MAPE of the common test set from 0.5 to 5.52. 5 days of lead does not allow for much planning to take place for individuals or SMEs, but does allow for the best trade-off between a loss in accuracy and an increase in forecasting range. 10 days of lead yields an increase on the base case MAPE of approximately 1.6, to 6.6. However it is an approximately 3x increase in error for a 2x gain in forecasting range, relative to the 0 and 5 lead day difference. It could be an interesting point as 10 days, which corresponds to two weeks, should give SMEs sufficient time to plan fuel purchasing and logistical decisions. The relationship between MAPE and lead time can be roughly shown by the following equation:

$$y = 0.1462x + 5.0596, \tag{5.1}$$

where y is the MAPE and x is the length of the sequence. The remaining models will be assessed with a 50 day sequence input comprising of spot and future prices, with a 10 day lead time.

5.3 RNN model

The second deep learning model assessed is the RNN. In the first experiment this topology is evaluated with the LSTM and GRU cell. The second experiment explores the effect of a reduced input data set on the model's ability to forecast the BFP. These experiments are described in greater detail in Section 4.3.3.

5.3.1 Using LSTM and GRU cells

Two RNN models are created to forecast the BFP with a 50 day input sequence and a 10 day lead time. The first model is comprised of LSTM cells and the second of GRU cells. A description of the network and their hyperparameters can be seen in Table 5.2, where CELL is a place holder for the LSTM and GRU cell. The hyperparameters for the RNN model are identical to those of the FFNN except for the number of epochs dropped from 50 to 20.

Table 5.2: Description of the RNN.

(a) Network topology

Layer	Dimension	Activation
Input CELL	40, 240	ReLU
Hidden CELL 1	40, 120	ReLU
Hidden CELL 2	60	ReLU
Dropout 1	60	
Hidden dense 1	60	ReLU
Dropout 2	60	
Output	1	Linear

(b) Network hyperparameters

Parameter	Value
Dropout rate	0.04
Epochs	20
Learning rate	0.01
Decay	1.0E-6
Momentum	0.5

The LSTM model yields a MAPE of 6.16 on the common testing set which is approximately 0.5 less than the FFNN for the same task. This is a slight improvement, but the RNN required 367 seconds to train, which is many times more than the training time of the FFNN. This finding is aligned with what is stated in the literature, that is that RNNs can provide better results on time series data but consume more resources due to their relatively more complex nature.

The GRU model provides similar results to the LSTM, with a MAPE of 6.34 on the same task, but required less time at 281 seconds (which is still many times more than the training time of the FFNN). The lower training time can be attributed to the simpler structure of the GRU cell when compared to the LSTM.

Although the LSTM and GRU results are very similar, we shall proceed forward with the LSTM based RNN to determine the sensitivity of the network to a smaller data set. In the next experiment the common training, validation and testing sets are halved to determine the impact of less data on the effectiveness of the model.

5.3.2 Using less data

As discussed in the previous case, the difference in error between the LSTM and GRU based RNN is fairly small, and this experiment will be done with the LSTM based RNN for the same task. With the common data sets halved, the LSTM based RNN yields a MAPE of 11.07 which is almost double the error of the previous LSTM case. This indicates that these models are sensitive to the amount of data available. This finding would indicate that perhaps with more data they would yield more accurate results for the forecasting of the BFP.

5.4 CNN models

The final network topology explored in this study is that of the CNN. In the first experimental case, the effectiveness of the model is assessed with the same input criteria as that of the initial RNN experiment. The next experiment is set up to determine if the composition of the test data set has a meaningful effect on the MAPE. Additional details regarding the CNN are discussed in Section 4.3.4. The hyperparameters of the CNN model are the same as the hyperparameters of the FFNN, and can be seen in Table 5.1.

5.4.1 Evaluating performance of CNN on the same task

A description of the simple CNN developed to forecast the BFP can be seen in Table 5.3.

Table 5.3: Description of the CNN.

Layer	Dimension	Activation
Input Conv2D	4, 40, 1	ReLU
MaxPooling2D	2, 20, 1	
Hidden Conv2D 2	2, 20, 1	ReLU
Hidden MaxPooling2D 2	1, 10, 1	
Hidden Flatten 1	10	
Hidden Dense 1	60	ReLU
Hidden Dropout 1	60	
Output	1	Linear

The CNN produces a test MAPE of 9.8 when operating on the 50 sequence input of prices and futures with a 10 day lead time. This is not as effective as the FFNN or the RNN when operating on the same task but it does train a bit quicker than the FFNN at approximately 16 seconds. This poorer performance could be due to the fact that the input data is not sufficiently large in dimension for the CNN to detect suitable patterns in the structure of the input time series.

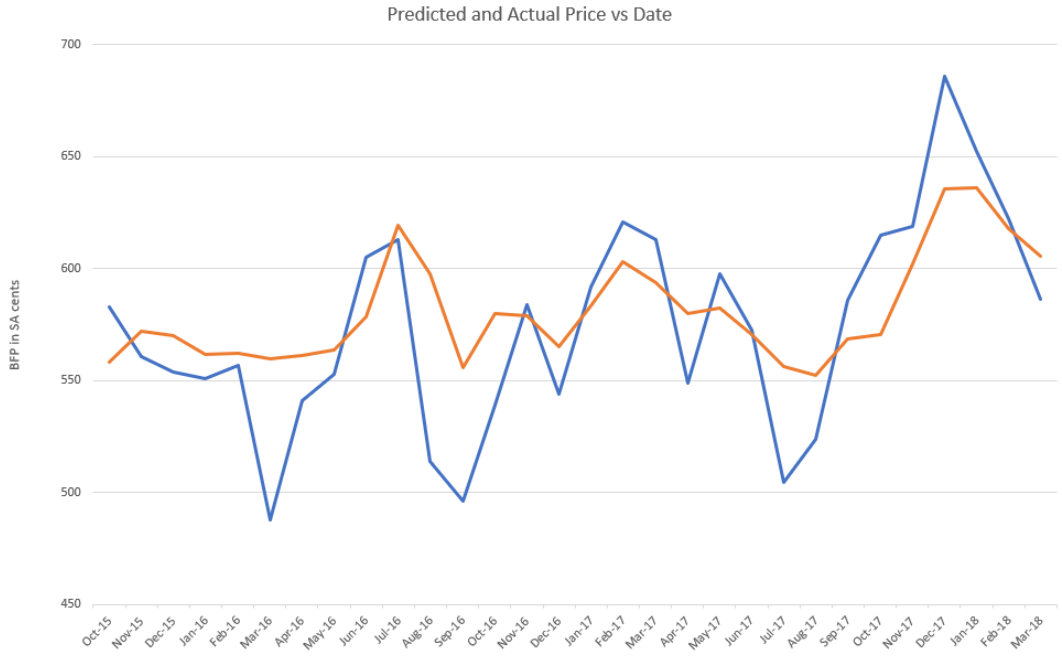
5.4.2 Testing on latest BFP observations

The next experiment uses the same model, hyperparameters and task as above, but evaluates with the latest observed 30 BFP points as the test set. The CNN model yields a MAPE of 5.91, which is a great improvement over the above result. This raises some questions, as such a large improvement was not expected.

It is postulated that such an improvement could be due to the fact that the training set in this experiment contains more data points presenting greater price volatility. For instance it could have more data representing major events such as the 2008 recession. This turns out to be at least somewhat true as the latest 2.5 years worth of data was removed from the total data set to create the new test data set for this experimental case. The last 2.5 years worth of data had relatively small price fluctuations when compared to past major events such as the recession. So the training data set had a higher probability of being populated with major events when randomly sampled from the full BFP data set. Such a result also highlights the need for larger data set sizes to potentially improve performance of these models by reducing the effect of major price swings in the input data.

5.5 Optimal BFP forecasting model

The final experiment is an attempt to create a model that would yield the lowest MAPE for the task of forecasting the BFP. This is done by combining the best results presented in the previous experimental cases. The most optimal model can be defined as an LSTM based RNN operating on a 50 day input sequence comprised of spot and future prices with a 10 day lead time. This model is assessed with the latest observed BFP values in the test set. As expected this model is the best performer in the task of forecasting the BFP with an average MAPE of 4.57. The hyperparameters for this RNN are the same as the parameters described in the first RNN experimental case in Section 5.3. Actual and predicted values for the test set are plotted in Figure 5.3. Although the lowest error is achieved with this setup, it was found that the model could not reliably, in all observations forecast close to the actual price. From Figure 5.3 it can be seen that the model tracks the movements of the actual price changes quite well but as stated cannot at all points reliably



	Oct-15	Nov-15	Dec-15	Jan-16	Feb-16	Mar-16	Apr-16	May-16	Jun-16	Jul-16	Aug-16	Sep-16	Oct-16	Nov-16	Dec-16	Jan-17	Feb-17	Mar-17	Apr-17	May-17	Jun-17	Jul-17	Aug-17	Sep-17	Oct-17	Nov-17	Dec-17	Jan-18	Feb-18	Mar-18
Actual	586.2	622.2	652.2	686.2	619.1	615.1	586.1	523.7	504.7	572.7	597.7	548.7	612.9	620.9	591.9	543.9	584	539	496	514	613	605	553	541	488	557	551	554	560.9	582.9
Predicted	605.6	617.8	636.4	635.7	602.1	570.8	568.5	552.1	556.1	570.7	582.4	580.2	593.6	603	583.3	565.1	578.9	579.9	555.9	597.8	619.2	578.2	563.9	561.3	559.9	562	561.9	569.9	572.1	558.4

Figure 5.3: Actual and predicted BFP vs date.

get close to the actual price. This was due to individual observed absolute percentage errors ranging from less than 1 to over 10, leading to the average of 4.57.

The model's ability to track the general trend is interesting and potentially useful. Of all points displayed, the model correctly forecast the direction (up or down) of the price change, between sequential date pairs, 25 times and miss forecast it 4 times. That equates to a success rate of 86%. The miss forecasted direction of the price change in most cases was only off by a few cents. This is positive, and can potentially be improved upon for even better price direction forecasting of the BFP.

5.6 Overview

This concludes the results Section of this study. An overview of the experiments and their associated MAPE can be observed in Table 5.4. From this Table there are some notable results. The regression model is a relatively unsophisticated model but performs fairly well with a low training time. This model serves as the study benchmark. As highlighted in the literature such methods are well suited for this task. The results of the subsequent experiments are evaluated relative to this initial error value. A single instance of a model performing better than another is not a universal sign of that model's superiority. The results presented indicate a trend of overall improving performance given different model topologies and inputs, and should be interpreted as such. The first FFNN model provides a slight improvement on the regression model but with a greater training time. The MAPE of the FFNN improves with the introduction of the futures but increases with the introduction of a 10 day lead in the input data. In the latter case accuracy is being traded for forecasting utility. Both the LSTM and GRU implementations of the RNN improve upon the MAPE of the FFNN, but at the cost of a much greater training time. As suggested in the literature, these models are sensitive to the amount of data provided during training and this is evident in the last RNN experiment where the MAPE increases substantially when given less training data. The CNN does not fare as well on this data set as the FFNN and RNN, but gives some insight as to how the composition of the test set can have an effect on the MAPE of the model under review. As expected, the optimal model achieves the lowest MAPE as it combines the best elements of the previously observed results into a single model. The regularisation hyperparameters of the models went unchanged through the course of experimentation. This may have been due to the small data set associated to the BFP, or potentially it speaks to the uniformity of the data as they were used in multiple NN topologies.

Table 5.4: MAPE of experiments.

Model	MAPE	Train time
Regression	6.55	1s
FFNN	6.31	18s
FFNN w futures	5.02	19s
FFNN w 10 day lead	6.60	19s
RNN w LSTM	6.16	367s
RNN w GRU	6.34	281s
RNN LSTM w less data	11.70	187s
CNN	9.80	16s
CNN w latest test	5.91	16s
Optimal	4.57	364s

Chapter 6

Conclusion and future work

This last Chapter presents a conclusion to the question posed in Chapter 1, that being whether neural network techniques can be effectively applied to the task of forecasting the South African BFP. This Chapter also covers some additional related topics that could potentially be looked at in a future study.

6.1 Conclusion

The aim of this research was to determine whether neural network techniques would be suitable in forecasting the South African basic fuel price (BFP). A number of network topologies were assessed, including the FFNN, RNN and the CNN. All models were rated on their ability to minimise the mean absolute percentage error (MAPE) on a common test data set. The models were fitted and validated from common training and validation data sets. These three data sets were populated from a set of BFP data spanning from April 2003 to March 2018. This data was published by South African Department of Energy.

The FFNN performed slightly better than the simple linear regression model which has been used in the past for forecasting of the South African fuel price by industry professionals. Both of these models operated on an input consisting of exchange rates and oil prices to forecast the BFP. It must be noted that although the linear regression was slightly less accurate than the FFNN, it could be fitted many times faster than the FFNN. This model fitting time could be a deciding point whether or not to use the simpler method or the relatively more complex FFNN. It was established that adding the exchange rate and oil price future contracts to the input of the FFNN further improved the average error on the test set.

A key aspect of forecasting is how reliably into the future a prediction can be made. It was determined that a lead time of 10 days before the expected BFP change provided the best trade-off between the forecasting range and the

increased model error, as modelling further into the future yielded a greater error.

The next topology to be assessed was that of the RNN. The RNN was the most complex model reviewed and took the longest to fit, but this increased demand on resources gave way to the lowest MAPE yet seen. The RNN training time was many times greater than the FFNN for a marginal improvement in performance over it. Again, this training time constraint could be a deciding factor in model selection. Two RNNs were trained and assessed, one made up of LSTM cells and another comprising of GRU cells. The LSTM based RNN performed slightly better when compared to the RNN comprising of GRUs, but the LSTM based RNN took longer to train. This is most likely due to the increased complexity of the internal workings of the LSTM cell when compared to the streamlined GRU cell implementation.

While assessing the effectiveness of the RNN it was determined that these models can be sensitive to the amount of data provided for fitting. Halving the data yielded an almost doubling in MAPE. This result could also allude to the possibility that providing more data than what the common sets contain, could result in a reduction in the MAPE and hence more reliable forecasting.

The CNN was the last unique topology assessed. Until recently, the CNN has not normally been used when time series data is the format of the model input. The intuition around the use of such a model is that the convolutional operation upon which the model is based could be used to detect patterns in the input time series data much akin to the matrix like input associated to images, which these models often operate on. The results of the CNN when forecasting the BFP were not very promising when compared to the FFNN or RNN. It is postulated that the reason for this is that the dimension of the input is not sufficiently large for the model to reliably detect a pattern or relationship between the input and the output.

While assessing the CNN, another experiment was performed. In this case the test set was populated with the last 2.5 years worth of observed BFP data and the training and validation sets randomly populated with the remaining data. Considering the nature of the forecasting task, it would be worthwhile back-testing with such a data set. In this case the CNN performed very well.

A final experiment was performed by combining the best elements of each experimental case in an attempt to create a model with the greatest forecasting power measured by its ability to reduce the MAPE of the test data set. In this case the test set was the last 2.5 years. The model topology selected was the RNN operating on a 50 day input sequence consisting of spot and future prices with a 10 day lead. As speculated this combination provided the lowest average error. Unfortunately errors in individual observations ranged widely so the forecasting of discrete BFP values can still be somewhat unreliable, but the model managed to correctly forecast the direction of the price change quite

well. It must be noted that the model was trained to forecast individual basic fuel prices. It could potentially be more capable at price direction forecasting, if explicitly trained to do so.

With these results it can be claimed that neural networks, specifically RNNs can be used to forecast the direction of a change in the South African BFP with a relatively low error rate. Such forecasting ability could be valuable to SMEs when it comes to making fuel purchasing or logistical decisions.

6.2 Future work

Two potential areas that could benefit from additional investigation include the modelling methodology and the size and content of the data sets.

6.2.1 Methodology

As stated previously, the aim of this research was to determine the capability of various machine learning techniques to forecast the South African BFP. The BFP is one of many elements that make up the total fuel price. It was determined that an LSTM based RNN could be an effective method to forecast the price change direction of the BFP per month. The BFP has a large impact on the total price, due to its percentage contribution and its volatility, hence making it a good candidate to model first. The proposed forecasting model could potentially be extended to include additional elements such as the fuel levy and transport costs in an attempt to model the total fuel price.

Although the main focus was on neural network techniques, other methods such as ARIMA or hybrid models could be a potentially interesting avenue to explore regarding their forecasting ability on the presented data sets. The performance of these hybrid models could be benchmarked relative to the performance of the discussed methods.

6.2.2 Data sets

As stated in the results, the implemented methods are sensitive to the amount of data available for training. So obtaining more BFP training data should be useful in reducing the overall error. An investigation into more supplementary data sets as input to the models could yield beneficial results and further increase a model's forecasting capability. This data could potentially include:

- spot and future prices of gold;
- volatility indexes such as the RAIN and VIX;
- US Federal funds rate.

References

[1] Jean Folger: How gas prices affect the economy. 2011.
Available at: https://www.investopedia.com/financial-edge/0511/how-gas-prices-affect-the-economy.aspx

[2] Lameez Omarjee: More than 50 percent of SA population is living in poverty. 2017.
Available at: https://www.fin24.com/Economy/more-than-50-of-sas-population-is-living-in-poverty-20170822

[3] Kabelo Khumalo: Transport costs a worry for commuters. 2017.
Available at: https://www.iol.co.za/personal-finance/transport-costs-a-worry-for-commuters-11239314

[4] BusinessTech - Staff Writer: Why 30 million South Africans are trapped in poverty - even with a national minimum wage. 2017.
Available at: https://businesstech.co.za/news/wealth/196966/why-30-million-south-africans-are-trapped-in-poverty-even-with-a-national-minimum-wage/

[5] Siseko Njobeni: Poor don't benefit from SA transport. 2016.
Available at: https://www.iol.co.za/business-report/economy/poor-dont-benefit-from-sa-transport-2041686

[6] The Banking Association of South Africa: SME Enterprise. 2013.
Available at: http://www.banking.org.za/what-we-do/sme/sme-enterprise

[7] Fin24 - Economy: Fuel price hike to hit SMEs the hardest. 2018.
Available at: https://www.fin24.com/Economy/fuel-price-hike-to-hit-small-enterprises-the-hardest-20180403

[8] Hein du Plessis: Fuel price trends and forecast. Tech. Rep., Eqstra fleet consulting, January 2014.

[9] NAAMSA: National association of automobile manufacturers of South Africa, unleaded FAQ. 2006.
Available at: http://www.naamsa.co.za/unleaded/faq.htm

[10] Sherryn de Vos: Should I use the 93 or 95 version of unleaded petrol? 2017.
Available at: http://compareguru.co.za/news/should-i-use-the-93-or-95-version-of-unleaded-petrol/

[11] FuelEconomy, US Department of Energy: Selecting the right octane fuel. 2015.
Available at: https://www.fueleconomy.gov/feg/octane.shtml

[12] Department of Energy: History of petrol prices. 2011.
Available at: http://www.energy.gov.za/files/esources/petroleum/history_petrol_price.html

[13] The Automobile Association: Petrol price breakdown. 2018.
Available at: https://www.aa.co.za/insights/petrol-price-breakdown

[14] South African Petrol Industry Association: Overview - fuel price. 2003.
Available at: http://www.sapia.org.za/overview/fuel-price

[15] Department of Energy: Fuel price structure - overview. 2006.
Available at: http://www.energy.gov.za/files/esources/petroleum/petroleum_pricestructure.html

[16] Department of Energy: Basic fuel price - February 2018. 2018.
Available at: http://www.energy.gov.za/files/esources/petroleum/February2018/Basic-Fuel-Price.pdf

[17] Department of Energy: South African fuel price history. 2018.
Available at: http://www.energy.gov.za/files/esources/petroleum/February2018/Fuel-Price-History.pdf

[18] Malusi Gigaba, Minister of Finance : Budget speech 2018. 2018.
Available at: http://www.treasury.gov.za/documents/national\%20budget/2018/speech/speech.pdf

[19] Department of Energy: Petrol levies, taxes and margins 95 octane (unleaded petrol). 2018.
Available at: http://www.energy.gov.za/files/esources/petroleum/February2018/Petrol-margins.pdf

[20] Department of Energy: Petroleum sources - overview. 2006.
Available at: http://www.energy.gov.za/files/petroleum_frame.html

[21] Jason van Bergen: 6 factors that influence exchange rates. 2018.
Available at: https://www.investopedia.com/articles/basics/04/050704.asp

[22] Central Energy Fund: CEF - about us. 2015.
Available at: http://www.cefgroup.co.za/about-us/

[23] Department of Energy: Self adjusting slate levy mechanism. 2008.
Available at: http://www.energy.gov.za/files/esources/pdfs/energy/liquidfuels/slate\%20levy.pdf

[24] Emad Saad, Danil Prokhorov (Member) and Donald Wunsch II: Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks. *IEEE Transactions on Neural Networks*, vol. 9, no. 6, 1998.

[25] Lin Zhao: Neural networks in business time series forecasting: benefits and problems. *Review of Business Information Systems*, vol. 13, no. 3, 2009.

[26] Mary Kasprzak: Forecasting jet fuel prices using artificial neural networks. Tech. Rep., Naval Postgraduate School Monterey CA, 1995.

[27] Anita Thakur, Aishwarya Tiwari, Saswat Kumar, Aditya Jain and Jagjot Singh: NARX based forecasting of petrol prices. In: *2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*, pp. 610–614. IEEE, 2016.

[28] Nasha Mavee and Axel Schimmel-pfennig: What makes the rand so volatile: global or home-made factors? 2017.
Available at: `http://www.econ3x3.org/article/what-makes-rand-so-volatile-global-or-home-made-factors`

[29] Investopedia: Autoregressive Integrated Moving Average - ARIMA. 2010.
Available at: `https://www.investopedia.com/terms/a/autoregressive-integrated-moving-average-arima.asp`

[30] Adebiyi Ariyo, Adewumi Adewumi, and Charles Ayo: Stock price prediction using the ARIMA model. In: *2014 UKSim-AMSS 16th international conference on computer modelling and simulation (UKSim)*, pp. 106–112. IEEE, 2014.

[31] Nuhu Isah and Abdul Talib Bon: Application of Markov model in crude oil price forecasting. *Path of Science*, vol. 3, no. 8, 2017.

[32] T.C.E. Cheng, Y.K. Lo and K.W. Ma: Forecasting stock price index by multiple regression. *Managerial Finance*, vol. 16, no. 1, pp. 27–31, 1990.

[33] Jenjira Tipyan and Chartchai Leenawong: Quantitative models for forecasting vehicle fuel prices in Thailand. In: *2009 WRI World Congress on Computer Science and Information Engineering*, vol. 2, pp. 317–321. IEEE, 2009.

[34] Halbert White: Economic prediction using neural networks: the case of IBM daily stock returns. In: *IEEE 1988 international conference on neural networks*, pp. 451–458 vol.2. July 1988.

[35] Kyungjoo Lee, Sehwan Yoo and John Jongdae Jin: Neural network model vs SARIMA model in forecasting Korean Stock Price Index (KOSPI). *Issues in Information Systems*, vol. 8, no. 2, pp. 372–378, 2007.

[36] Masaya Abe and Hideki Nakayama: Deep learning for forecasting stock returns in the cross-section. *arXiv preprint arXiv:1801.01777*, 2018.

[37] Siddhivinayak Kulkarni and Imad Haidar: Forecasting model for crude oil price using artificial neural networks and commodity futures prices. *arXiv preprint arXiv:0906.4838*, 2009.

[38] Umut Ugurlu, Ilkay Oksuz, Oktay Tas and others: Electricity price forecasting using recurrent neural networks. *Energies*, vol. 11, no. 5, pp. 1–23, 2018.

[39] Anastasia Borovykh, Sander Bohte and Cornelis Oosterlee: Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691v4*, 2018.

[40] Roselina Sallehuddin, Shamsuddin Hj and Mariyam Siti : Hybrid grey relational artificial neural network and auto regressive integrated moving average model for forecasting time-series data. *Applied Artificial Intelligence*, vol. 23, no. 5, pp. 443–486, 2009.

[41] Ping-Feng Pai and Chih-Sheng Lin: A hybrid ARIMA and support vector machines model in stock price forecasting. *Omega*, vol. 33, no. 6, pp. 497–505, 2005.

[42] Ian Goodfellow, Yoshua Bengio and Aaron Courville: *Deep Learning*. MIT Press, 2016. ISBN 9780262035613.

[43] Tom Mitchell: *Machine Learning*. McGraw Hill, 1997. ISBN 0070428077.

[44] David Wolpert and William Macready: No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[45] Leo Breiman: Bagging predictors. *Machine Learning*, vol. 24, pp. 123–140, 1996.

[46] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov: Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[47] Justin Bayer and Christian Osendorfer: Learning stochastic recurrent networks. International Conference on Learning Representations (ICLR). Google Inc., Mountain View, CA, 2015.

[48] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho and Yoshua Bengio: How to construct deep recurrent neural networks. International Conference on Learning Representations (ICLR). 2014.

[49] Ian Goodfellow, Jonathon Shlens and Christian Szegedy: Explaining and harnessing adversarial examples. International Conference on Learning Representations (ICLR). Google Inc., Mountain View, CA, 2015.

[50] Louis Augustin Cauchy: Methode generale pour la resolution de systemes dequations simultanees. *Compte Rendu a lAcad emie des Sciences*, pp. 536–538, 1847.

[51] Kurt Hornik, Maxwell Stinchcombe and Halbert White: Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, vol. 3, pp. 551–560, 1990.

[52] David Rumelhart, Geoffrey Hinton and Ronald Williams: Learning internal representations by error propagation. *Nature*, vol. 323, pp. 533–536, 1986.

[53] Josef Hochreiter: *Untersuchungen zu dynamischen neuronalen Netzen*. Master's thesis, Institut fur Informatik Technische Universitat Munchen, Arcisstr 21, 8000 Munchen 2, Germany, 6 1993.

[54] Sepp Hochreiter and Jurgen Schmidhuber: Long Short-Term Memory. *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[55] Felix Gers, Jurgen Schmidhuber and Fred Cummins: Learning to forget: Continual prediction with LSTM. 1999.

[56] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jurgen Schmidhuber and others: Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. 2001.

[57] Kyunghyun Cho, Bart Van Merrienboer, Dzmitry Bahdanau and Yoshua Bengio: On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[58] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho and Yoshua Bengio: Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[59] Yi-Tong Zhou and Rama Chellappa: Computation of optical flow using a neural network. In: *IEEE International Conference on Neural Networks*, vol. 1998, pp. 71–78. 1988.

[60] Investopedia: Financial instrument. 2016.
Available at: `https://www.investopedia.com/terms/f/financialinstrument.asp`

[61] Investopedia: Commodity. 2016.
Available at: `https://www.investopedia.com/terms/c/commodity.asp`

[62] Investopedia: Exchange rate. 2017.
Available at: `https://www.investopedia.com/terms/e/exchangerate.asp`

[63] Investopedia: Spot price. 2016.
Available at: `https://www.investopedia.com/terms/s/spotprice.asp`

[64] Investopedia: Security. 2017.
Available at: `https://www.investopedia.com/terms/s/security.asp`

[65] Investopedia: Derivative. 2015.
Available at: `https://www.investopedia.com/terms/d/derivative.asp`

[66] Investopedia: Futures contract. 2018.
Available at: `https://www.investopedia.com/terms/f/futurescontract.asp`

[67] Ruby-lang: About Ruby. 2006.
Available at: `https://www.ruby-lang.org/en/about/`

[68] Rubyonrails: Rails. 2016.
Available at: `https://rubyonrails.org/`

[69] International Organization for Standardization: ISOIEC 9075-1:2016. 2016.
Available at: `https://www.iso.org/standard/63555.html`

[70] SQLite: About SQLite. 2007.
Available at: `https://sqlite.org/about.html`

[71] Malcom Smith: Python Beginners Guide. 2017.
Available at: `https://wiki.python.org/moin/BeginnersGuide/Overview`

[72] Skipper Seabold and Josef Perktold: Statsmodels: Econometric and statistical modeling with python. In: *9th Python in Science Conference*. 2010.

[73] TensorFlow: About TensorFlow. 2018.
Available at: `https://www.tensorflow.org/`

[74] Keras: Keras: The Python Deep Learning library. 2015.
Available at: `https://keras.io/`

[75] Git: Git. 2012.
Available at: `https://git-scm.com/`

[76] Github: Github. 2008.
Available at: `https://github.com/`

[77] DigitalOcean: DigitalOcean. 2018.
Available at: `https://www.digitalocean.com/`

[78] Department of Energy: Petroleum sources - petrol price archive. 2006.
Available at: `http://www.energy.gov.za/files/petroleum_frame.html`

[79] Quandl: Quandl. 2018.
Available at: `https://www.quandl.com/`

[80] Open Exchange Rates: About the Open Exchange Rates API. 2012.
Available at: `https://openexchangerates.org/about`