Time Series Forecasting Using Neural Networks

Bruno Miguel Paiva Soalheira

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Andreas Miroslaus Wichert

Examination Committee

Chairperson: Prof. José Carlos Martins Delgado
Supervisor: Prof. Andreas Miroslaus Wichert
Member of the committee: Prof. João Carlos Serrenho Dias Pereira

October 2019

Acknowledgments

I would like to acknowledge everyone that played an important role during my academic accomplishments.

First, I want to thank my supervisor, professor Andreas Wichert, for supporting me since the beginning of my thesis. Every time I needed technical advice or even a word of encouragement, he was there to help me.

Secondly, I want to thank my family, who were always there with unconditional love and support.

Finally, my closest friends, who were extremely important just by being there and all the small gestures and words of encouragement.

Thank you all for making this possible.

Abstract

Artificial neural networks are a powerful machine learning technique that is used for several ends; time series forecasting is one of them. Several of these techniques are explored with special attention to recurrent neural networks, which are one of the methods most commonly used in time series forecasting. All these methods are presented with the goal of better understanding recurrent neural networks. There is no predefined better algorithm to solve this type of problems; in this field experimentation is required and only then we can draw conclusions.

In this case, feedforward neural networks and recurrent neural networks will be used to obtain models capable of predicting time series. Two related data sets but with some important differences that will affect the results are used in these experimentations. The main objective is to compare the two types of networks on both data sets, presenting the limitations, drawbacks and advantages of each one of them.

In most cases, recurrent neural networks outperform feedforward neural networks for time series forecasting problems.

Keywords: machine learning; recurrent neural networks; feedforward neural networks; time series forecasting.

Resumo

Redes neuronais artificiais são uma técnica de aprendizagem de máquina que é usada para várias finalidades; previsão de séries temporais é uma delas. Muitas desta técnicas são exploradas com especial atenção às redes neurais recorrentes, que são um dos métodos mais utilizados neste tipo de problemas de previsão de séries temporais. Não existe um método predefinido que seja melhor que os outros para resolver este tipo de problemas; nesta área é necessário experimentação e só depois é que é possível retirar conclusões.

Neste caso, redes neurais feedforward e redes neurais recorrentes serão usadas para obter modelos capazes de fazer a previsão de séries temporais. Dois conjuntos de dados relacionados mas com importantes diferenças que afetarão os resultados serão usados nestas experimentações. O principal objetivo é comparar os dois tipos de redes neurais em ambos os conjuntos de dados, apresentando as limitações, vantagens e desvantagens de cada um deles.

Na maioria dos casos, redes neurais recorrentes têm melhor desempenho do que redes neurais feedforward para problemas de previsão de séries temporais.

Palavras-chave: aprendizagem de máquina; redes neurais recorrentes; redes neurais feedforward; previsão de séries temporais.

Table of Contents

1 Introduction

1.1 Motivation

1.1.1 Time Series Forecasting

Time series forecasting is an area of machine learning that has been fascinating data scientists for the last few decades. The field of machine learning has brought us innumerous advantages, being one of them the capability of being able to predict what is going to happen and the only thing we need is information. No matter how hard and random a data set may seem, these algorithms can find patterns that are not possible to identify using other techniques. Using past and present data we can infer what is going to happen in the future with precision; we must choose the right method and experimentation is the key.

1.1.2 Neural networks

Neural networks are an amazing method using for different purposes; time series forecasting is only one of them. They can also be used in many different other areas like in customer research, data validation and risk management. It is also important to refer that they are not alone; there are other methods capable of accomplishing some things that neural networks can and we will discuss these in the next chapter.

Neural networks models have been around for more than half a century now. Since the publication of the first papers on the matter, scientists knew these models had enormous potential to achieve innumerous goals. Just by thinking of the complexity of the human brain, we can only imagine how far neural networks can go. It is a huge area of study and it is important to refer that only a small fraction of the neural network's capabilities is taken advantage of, nowadays. There is so much more to discover on this matter.

1.2 Objectives

The first part of the thesis is focused on presenting relevant information regarding some of these machine learning techniques with special focus on neural networks.

In the second part of the thesis, two distinct neural networks models will be implemented to two distinct data sets: recurrent neural networks and feedforward

neural networks. We want to compare the results of both these models, and which one is more reliable to solve this type of problems.

1.3 Outline

As previously mentioned, this thesis will be composed of two main parts after this introduction:

- Chapter 2, in which we discuss and present the theory behind this complex area of research. We briefly present some important concepts; prediction methods that are used in this area other than neural networks; we explain in detail neural networks, including its history, definition and structure; then we delve into the specific type of neural networks we are interested in for this project: recurrent neural networks; we describe the different architectures that exist and the advantages, disadvantages and applications for this type of neural networks.
- In Chapters 3, 4 and 5 we will present the proposed architecture, the evaluation method that will be used to analyze the results and the experimentations and conclusions of the thesis, respectively.

2 Related Work

2.1 Important concepts

2.1.1 Forecasting

It consists of predicting what is going to happen, considering the information collected regarding past and present events. This allows people, businesses, companies and other entities to make decisions according to their goals. Although forecasting is, in several cases, a helpful technique it should not be used to exclude or ignore already known information.

Forecasting is a method that helps management in its attempts to cope with the uncertainty of the future.

2.1.2 Time Series

Characterized as a set of data points obtained from existing records, usually a sequence of equally spaced points in time, tabulated or plotted in time order. In a more mathematical way, using the definition suggested by Robert H. Shumway and David S. Stoffer in *Time Series Analysis and Its Applications* [1] *"…*we may consider a time series as a sequence of random variables $x1, x2, x3,$ ..., where the variable $x1$ denotes the value taken by the series at the first time point, the variable $x2$ denotes the value for the second time period, $x3$ denotes the value for the third time period, and so on."

There are several components of time series and some of the most important ones to consider are:

- *Trends.* The linear increasing or decreasing behavior of the series over time.
- *Seasonality.* A seasonal pattern exists when a series is influenced by seasonal factors.
- *Cycles.* A cyclic pattern exists when data exhibits rises and falls that are not of fixed period.
- *Noise.* The variability of the time series observations that cannot be explained by the model.

2.1.3 Time Series Analysis

The primary objective of this analysis is to develop mathematical models in order to find patterns in the data and to extract meaningful statistics and other

characteristics of the data [16]. There are many methods to obtain this type of information and we will delve into some of them later.

A time series can be white noise. It is white noise if the variables are independent and identically distributed with a mean of zero and a finite variance. White noise is important for a main reason:

- *Predictability.* If a time series is white noise it is not possible to reasonably model it and make any type of predictions.

2.1.4 Time Series Forecasting

Joining the previously mentioned concepts we obtain time series forecasting, which is the topic regarding this work [9]. Mainly, it consists of predicting future events by applying models to time series. It is an extremely important area of machine learning because there are several prediction problems involving time components. Every time series problem is very specific and with its own data characteristics, and according to that the most fitting model(s) must be chosen to analyze the data.

2.2 Some Prediction Techniques

In order to have a better understanding on the matter, we will look briefly into some techniques that are used in this type of analysis.

2.2.1 Moving Average

An interesting, yet simple model in which we infer predictions using the mean value of past data. Considering the characteristics of the data, we will choose how further back we want to investigate. Let F_{t+1} be the forecast value for the next period, where t is the current period, n is the forecasting horizon (i.e. how far back we look) and A is the data value from each period:

$$F_{t+1} = \frac{Æ_t + Æ_{t-1} + \ldots + Æ_{t-n}}{n} \tag{2.1}$$

2.2.2 Weighted Moving Average

A variant and clearly more versatile approach of the previously mentioned algorithm. As the name says, it is basically the same idea of the previous algorithm

except the fact that every data value from each period A_t, A_{t-1}, ... , A_{t-n} is multiplied by the importance (weight) we want to give to each period. Considering the same notations from equation (1) and using w as the weight for each period we can obtain the forecast the following way:

$$F_{t+1} = \frac{w_t Æ_t + w_{t-1} Æ_{t-1} + \ldots + w_{t-n} Æ_{t-n}}{n} \quad (2.2)$$

and considering that:

$$w_t + w_{t-1} + \cdots + w_{t-n} = 1 \quad (2.3)$$

Even though MA algorithms are simplistic and effective in some situations they also present several limitations.

The fact that Moving Averages can spread out over any period can be problematic because the general trend can change depending on the time period being analyzed. It is also important to refer that these methods don't work for all companies specially the ones that are heavily influenced by current events. They are only focused on past events and completely ignore current and valuable information such as new competitors, higher or lower demand for products in the industry…

Concluding, careful inspection must be done before applying this method since the results might be misleading if not done following the right terms.

2.2.3 Exponential Smoothing

A different approach that uses past values as dependent variables in a predictive model that gives more weight to more recent observations [2]. Let F_{t-1} be the forecast value for the last period, A_{t-1} be the data value for last period and α be the smoothing constant (expresses how much our forecast will react to observed differences) we can obtain the forecast for the next time period F_t:

$$F_t = F_{t-1} + \alpha(A_{t-1} - F_{t-1}) \quad (2.4)$$

As in the previous method, some concerns arise when dealing with Exponential Smoothing analysis.

It produces forecasts that lag the actual trends, the main reason being that the algorithm neglects the ups and downs associated with the random variations. Also, this method will not work properly if there is a trend in the series, therefore it is more

efficient when used for short-term forecasts and with no cyclical or seasonal variations.

2.2.4 Regression

The main objective is to obtain the relationships between a dependent variable (response) and one or more independent variables (predictors) with the final goal of generating a model that can predict future values for the dependent variables given values for the independent variables [3]. This type of analysis has several subgroups.

2.2.4.1 Linear Regression

It is a type of regression which consists of a linear approach to model the relationship between the dependent and independent variables. There are two types of linear regression:

- *Simple Linear Regression.* This is the simplest case in which only one dependent variable and one independent variable are considered. In other words, only one variable is needed to predict the future values. Considering y to be the response, x the predictor, β_0 and β_1 the regression coefficients and s the error term we obtain:

$$y = \beta_0 + \beta_1 x + s \tag{2.5}$$

where the regression coefficients have the mathematical meanings of interception (β_0) and slope (β_1) of a straight line are to be determined according to the data in analysis and the error term accounts for deviations of the data from the specified straight line.

- *Multiple Linear Regression.* In some cases, we take into account k predictors:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + s \tag{2.6}$$

and, once again, using the information we possess about the data we choose the values for $\beta_0, \beta_1, \beta_2, \ldots, \beta_k$ according to the importance that we want to give to each predictor.

It may seem obvious, but it is important to refer that linear models are limited to linear relationships and the data must be independent.

Only in a few cases we can find a straight line to model our data and, because of the model being described as a straight line, it is not possible to model cases in which having curves is convenient.

This model is based in the relationships between the mean values of the dependent and independent variables but in some cases can be necessary to look at the extremes of the dependent variable.

Outliers can have huge effects on the predictions because the mean values will be affected by these.

2.2.4.2 Polynomial Regression

It is another subtype of regression analysis and one of the ways to overcome some of the limitations presented by linear regression models, particularly the linearity [23][24]:

$$y = þ_0 + þ_1 x + þ_2 x^2 + \cdots + þ_k x^k + s \tag{2.7}$$

This method still presents some limitations.

As in the previous cases, the presence of outliers in the data can affect the results of a nonlinear analysis and, in addition to this, there are fewer ways to detect outliers in nonlinear regression models than there are for linear regression.

2.2.5 Autoregressive Model

This model uses previous observations as input to a regression equation to predict future values [11]. It is commonly used to describe time-varying processes in different areas, such as, nature, economics, etc. [25][26]. *AR(p)* is the notation used for this type of models and refers to an AR model or order *p* and, taking this into account, we can define the following equation to represent this model:

$$X_t = c + \sum_{i=1}^{p} *_i X_{t-i} + s_t \tag{2.8}$$

where X_t is the value to predict, X_{t-1}, X_{t-2}, ... , X_{t-p} are the *p* previous values used, $*_1$, $*_2$, ... , $*_p$ are the parameters of the model, *c* is a constant and s_t is the white noise error term.

2.2.6 Autoregressive-moving-average model (ARMA)

This name should sound familiar to the reader since this method involves two previously mentioned models working together. As a quick reminder, the AR part consists of predicting a value on its own past values and the MA part involves modeling the error term as a linear combination of error terms occurring contemporaneously and at successive times in the past. *ARMA(p,q)* is used to denote this model and refers to the p order of the AR part and the q order of the MA part [27]. Consider this different way of defining the MA model to better understand the ARMA model:

$$X_t = \mu + s_t + \sum^{q}_{i=1} 8_i \, s_{t-i} \tag{2.9}$$

where $8_1, 8_2, \ldots, 8_q$ are the parameters of the model, μ is the expectation of X_t and s_t and $s_{t-1}, s_{t-2}, \ldots, s_{t-q}$ are white noise error terms.

A new equation to define the ARMA model is obtained:

$$X_t = c + s_t + \sum^{p}_{i=1} *_i X_{t-i} + \sum^{q}_{i=1} 8_i \, s_{t-i} \tag{2.10}$$

2.2.7 Autoregressive integrated moving average (ARIMA)

It is a widely used generalization of the previous model and, once again, we have AR and MA working together with the particularity of the use of an integration component which indicates that the data values have been replaced with the difference between their current values and past values (this difference may occur several times) [28]. A standard notation used is *ARIMA(p,d,q)* where p and q were specified earlier and d refers to the number of times the raw observations are differenced, also known as the degree of differencing. In general, first we apply differencing and after we apply the ARMA model to the differenced series.

ARIMA models requires a stationary series (no trends, no seasonality and a constant autocorrelation).

It also requires expertise and experience if fitting the model manually (it is hard to choose the correct values for p,d,q).

Although this method is excellent for forecasting, it is quite difficult to explain and interpret, therefore if interpretability is very important ARIMA models might not be the best choice.

This section had the main goal of introducing some algorithms used in time series forecasting in order to provide some knowledge on how they work, which will be helpful in understanding the idea behind neural networks, a prediction technique that has the whole next section reserved for itself due to the fact that this will be the method used in this project.

2.3 Neural Networks

2.3.1 History

The study of the human brain has been around for centuries and it was only natural to try to harness this thinking process. In the 40s and 50s some scientists started taking this matter seriously and published several papers.

Warren McCulloch and Walter Pitts were among the first to write about neural networks in 1943 [4].

As computers advanced in the following years, it became possible to model some of these theories. The first few attempts to model neural networks failed but later attempts were successful and from that point on different types of neural network models were developed and used for various purposes.

In 1959, Bernard Widrow and Marcian Hoff developed models that they named ADALINE and MADALINE. MADALINE was the first neural network model to be applied to a real-world problem with success.

Scientists were impressed with the results of these models and they kept pushing forward which caused people to exaggerate the potential of neural networks. Neural networks were powerful methods but the technology limitations in the mid-1950s were too big to overcome. The perceptron, which was proposed by Frank Rosenblatt in 1958, had a major drawback; it could only learn to separate linearly separable classes. Critiques arose regarding neural networks research which, obviously, led to a period of stunted growth till the early 1980s.

In 1982 John Hopfield presented a paper in which he showed how neural networks could work and what they could do.

By 1985 the American Institute of Physics began what has become an annual meeting where innumerous scientists would get together and discuss new ideas about neural networks. This meeting was called Neural Networks for Computing.

Over the last few years, technology has been evolving exponentially, which led to the development of systems capable of handling even more complex artificial neural networks.

Nowadays, neural networks are a subject of constant study; they are discussed everywhere.

In Figure 2.1 some important advances in the area of neural networks are shown.

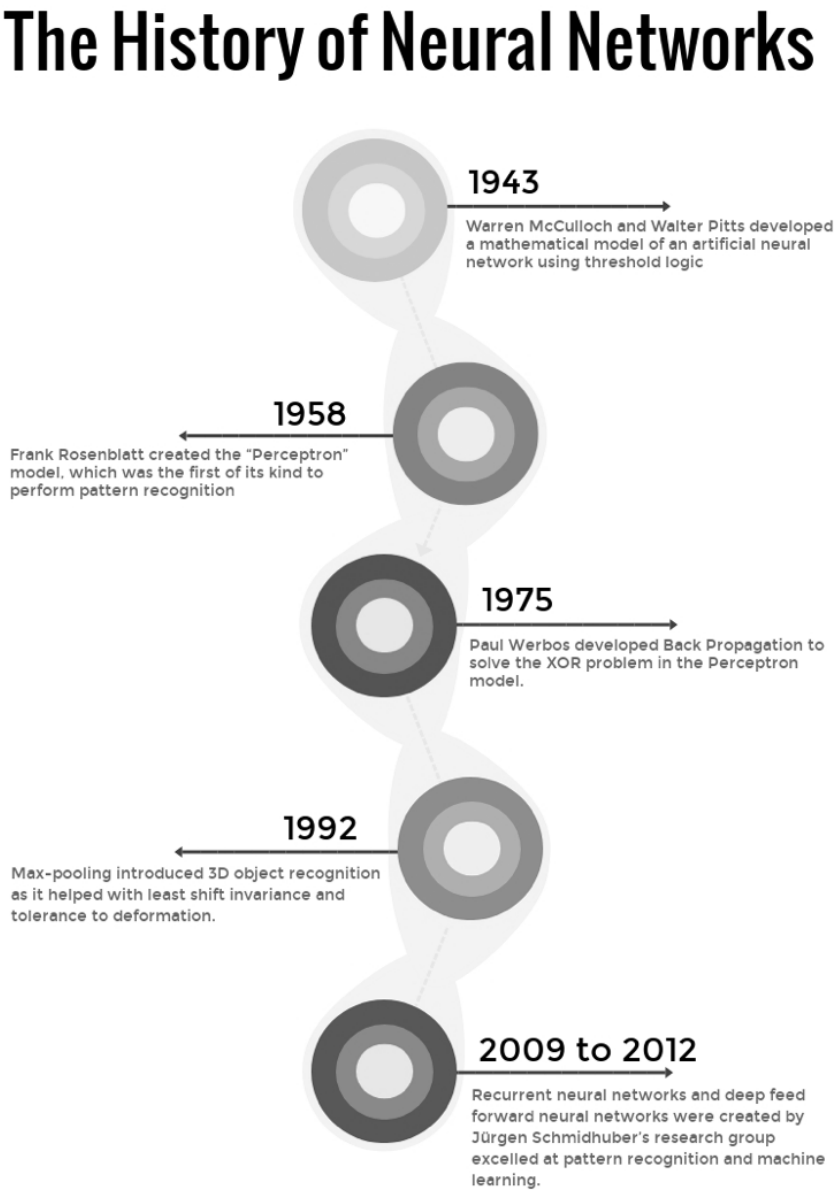

Figure 2.1 Important marks in the history of neural networks

2.3.2 Biology

The biological inspiration is evident therefore it is important to briefly discuss how the brain works [5]. The brain consists of millions of connected elements known as neurons. Considering, for our own convenience, that neurons have four principal components: the dendrites, the axon, the cell body and the synapses. The dendrites

are short and branched and is an extension of the nerve cell, along which impulses received from other cells at synapses are transmitted to the cell body. The axons are the long threadlike parts of the nerve cells, along which impulses are conducted from the cell body to other cells. The cell body is the spherical part of the neuron that contains the nucleus and is connected to both the axon and the dendrites. The junction or point of connection between two nerve cells, more specifically the connection between the axon of one cell and the dendrites of the next cell is called synapse. A simplified visual representation of all these components is represented in Figure 2.2.

Although artificial neural networks were, unarguably, one of the discoveries of the century they still do not approach, by any means, the complexity of the human brain. It is a fact that artificial neural networks have been capable of more and more over the last years, but they are still not able of mimicking the human brain for many complex tasks. It is expected that soon artificial neural networks will be performing tasks closer to human level and perhaps even becoming mathematically and structurally more similar to biological neural networks.

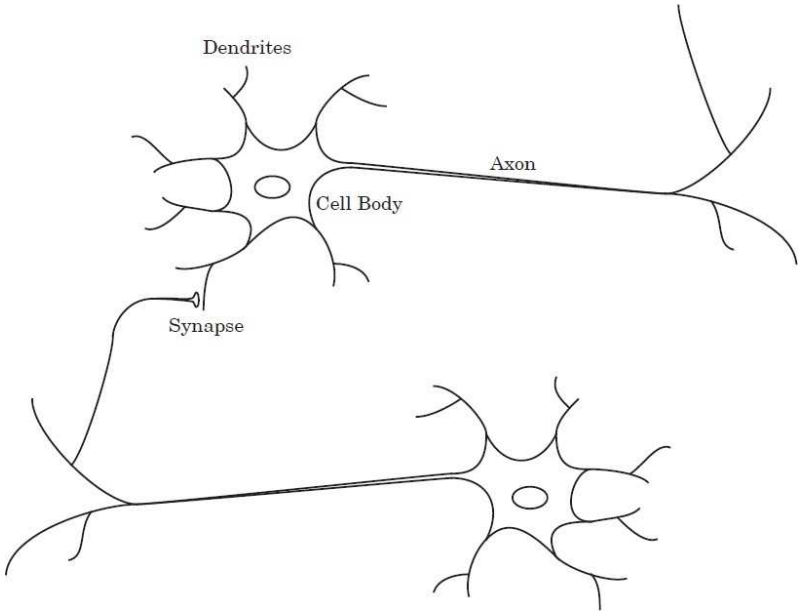

Figure 2.2 The Structure of a Neuron. Adapted from “Neural Network Design” by Hagan, M.T., Demuth, H.B., Beale, M.H. and De Jesús (1996)

2.3.3 Definition and Structure

An artificial neural network, commonly referred as simply neural network is a type of machine learning, just like all the methods presented in the last chapter. In its most general form, a neural network is a machine that is designed to model the

way in which the brain preforms a particular task; the network is usually implemented by using electronic components or is simulated in software on a digital computer [6][8] [10][12][13][14][15][16][17][18][19][20][21][22].

There are multiple types of neural networks and we will look into some of them. The most basic one and easier to understand is the feedforward neural network, in which the information travels in only one direction from input to output.

It is helpful to know the structure of a neural network in order to better understand it. Neural networks are, usually, composed by the following elements:

- *Input layer.* The input layer is the very beginning of the workflow for artificial neural networks. It consists of artificial input neurons and brings the initial data into the system for further processing by subsequent layers of artificial neurons.
- *Hidden layer.* The hidden layer is located between the input layer and output layer, where artificial neurons take in a set of weighted inputs and produce an output using an activation function.
- *Output layer.* The output layer is the last layer of a neural network that produces the outputs for the program.
- *Synapse.* The synapse is the strength of the connection between two artificial neurons.

The following Figure 1.2 shows the typical structure of an artificial neural network. The circles correspond to the neurons of each layer and the links between these neurons are the synapses.

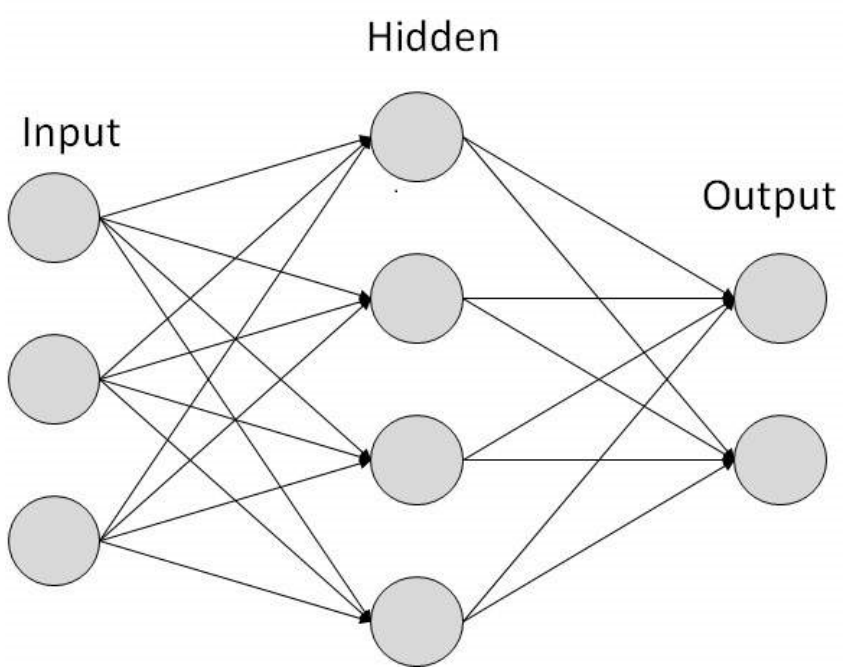

Figure 2.3 A Typical Neural Network

2.3.4 Learning Methods

Neural networks have a learning phase which is crucial for the potential results. There are three methods for learning strategies:

- *Supervised learning*. It is the machine learning task of learning a function that maps an input to output based on example input-output pairs. The example input-output pairs are usually called training data and each of these consists of an input object (usually a vector) and an output value. The new inputs that will be tested are called test data. This algorithm analyses the training data and infer a function that will be used to calculate the output values for the new test data values [12][29].
- *Unsupervised Learning.* This method must be used when there is no training data to learn from. The neural network analyses the data, and then a cost function tells the neural network how far it was from the correct result. Basically, the model learns from the test data and keeps adapting according to this data.
- *Reinforced Learning.* This algorithm reinforces the neural network in the presence of positive results and punishes it for negative results, forcing the neural network to learn over time.

2.3.5 Activation Function

In neural networks, an activation function is what defines the output of a node given an input or set of inputs. This output is used as input of the next node and so on until a desired solution to the original problem is found. It maps the resulting values into a desired range (0 to 1 or -1 to 1, for instance) [33].[1]

There are two types of activation functions:

- *Linear Activation Function.* The function is linear. Therefore, the output of the functions will not be confined between any range. These functions don't really help with the usual problems and data that we analyze using neural networks.

$$f(x) = x \tag{2.11}$$

- *Non-Linear Activation Function.* The function is not linear. This type of functions is commonly used because it can adapt to complex types of data.

[1] *What is an Activation Function* (deepai.org)?

For example, we have the sigmoid activation function, which is used to obtain values between 0 and 1 (commonly used in probability problems), we have tanh activation function, which provides values ranging from -1 to 1 (commonly used for classification between two classes), among many others. The sigmoid function is an example and is defined the following way:

$$\$(z) = \frac{1}{1+e^{-z}} \tag{2.12}$$

2.3.6 Backpropagation

In the beginning, the weights of a neural network are initialized randomly, and these values need to be adjusted until we obtain the values that fit our model the best. One of the most common ways to accomplish this is using the backpropagation technique. It is a method used to calculate a gradient that is needed in the calculation of the weights used in a neural network. It is more used in situations where there are more than a hidden layer in a neural network [34].[2]

Backpropagation requires three things to work properly:

- *A dataset* consisting of input-output pairs.
- *A feedforward neural network*, meaning that there are no connections between nodes in the same layer and that layers are fully connected.
- *An error function,* which defines the error between the desired output and the obtained output.

This method is used in any feedforward networks to learn a training set of inputs and outputs.

The main idea behind backpropagation is to update the weights' values in such a way that the error becomes minimum. First, we figure out if we have whether to increase or decrease the values of the weights. Once we know which way to go, we keep making smaller and smaller adjustments until the network provides the desired results.

Considering η to be the learning rate, C to be the loss function and $\varepsilon(t)$ to be a stochastic term we can update the weights w, using a stochastic gradient descent, the following way:

$$w_{ij}(t+1) = w_{ij}(t) + \eta \frac{\partial C}{\partial w_{ij}} + s(t) \tag{2.13}$$

[2] *What is Backpropagation?* (deepai.org).

2.3.7 Recurrent Neural Networks

Recurrent Neural Networks have been deeply studied since the 1990's. They are commonly used to learn sequential or time varying patterns, which is the matter we are dealing with in this project [7].

There are two types of architectures of recurrent neural networks.

The first ones are named fully connected neural networks (Figure 2.4). Fully connected networks do not have distinct input layers of nodes, and each node has input from all the other nodes.

The second ones are named partially connected neural networks (Figure 2.5). Although some nodes are part of a feedforward structure, other nodes provide the sequential context and receive information from other nodes.

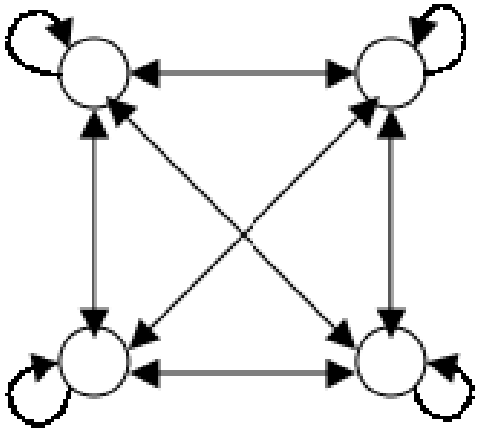

Figure 2.4 A Fully Connected Recurrent Neural Network

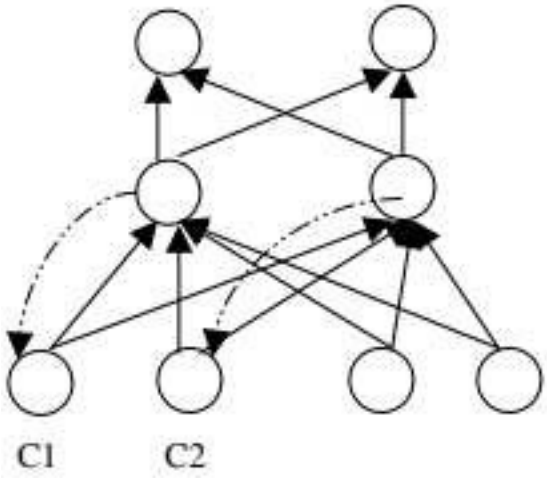

Figure 2.5 A Partially Connected Recurrent Neural Network

2.3.7.1 Elman Network

An Elman network contains three layers and a set of context units. The context units are connected to the hidden layer with the fixed weight of one. At each time step, the input is fed-forward, and a learning rule is applied. The context units save a copy of the previous values of the hidden layer nodes with the goal of maintaining a sort of a state, allowing this model to perform tasks like sequence-prediction [30][31].

Let h_t be the hidden layer vector, y_t the output vector, x_t the input vector, W, U and b the parameter matrices and vector and o_h and o_y the activation functions. The hidden layer and the output vector are calculated the following way:

$$h_t = o_h(W_h x_t + U_h h_{t-1} + b_h) \tag{2.14}$$

$$y_t = o_y(W_y h_t + b_y) \tag{2.15}$$

2.3.7.2 Jordan Network

Jordan networks are similar to Elman networks with the small difference of the context units containing information about the output layer instead of the hidden layer. Considering the same variable from the previous two formulas, we can obtain the value for the hidden layer and the output vector for the Jordan networks:

$$h_t = o_h(W_h x_t + U_h y_{t-1} + b_h) \tag{2.16}$$

$$y_t = o_y(W_y h_t + b_y) \tag{2.17}$$

2.3.7.2 Hopfield Network

It works by learning a number of binary patterns and then returning the one that is the most similar to a given input.

It consists of only one layer of nodes which are connected to each other but not to itself. It is a feedback network since the outputs are redirected to the inputs. Every node is, at the same time, the input and output of the network, therefore the number of input and output nodes is the same. After a certain number of iterations, the values of the nodes tend to converge.

2.3.7.3 Bidirectional Associative Memory

It is a variant of Hopfield network that stores associative data as a vector.

It consists of two layers of neurons, which are fully connected to each other. Once the weights have been established, the first input layer presents the pattern in the other layer, and vice versa [32].

2.3.7.4 Echo State

The echo state network contains a sparsely connected random hidden layer. The connectivity and weights of the hidden layer neurons are fixed and randomly assigned. The weights of the output neurons can be learned so that the network can produce specific temporal patterns. So, basically the only thing that changes in this network is the weight of the synapses connecting the hidden layer nodes to the output layer nodes.[3]

2.3.7.5 Independently Recurrent Neural Network

In this type of network, each neuron only receives information about its past state and thus neurons are independent of each other's states.

2.3.7.6 Conclusion

These mentioned models are just some of the ones that are used in time series forecasting problems. Only a brief description of each of them was provided in order to give a general idea of how each of them works.

2.3.8 Advantages of neural networks

Artificial neural networks bring some benefits that overcome some of the limitations referred in the algorithms mentioned in the previous chapter. It is apparent that a neural network derives its computing power through its massively parallel distributed structure and its ability to learn and therefore generalize. Generalization consists on the process of the neural network obtaining reasonable results for inputs that were not part of the training [11].

The following points are just some of the benefits and capabilities that neural networks offer us:

- *Nonlinearity.* Artificial neurons can be linear or nonlinear. A neural network, made up of an interconnection of nonlinear neurons, is itself nonlinear. This

[3] Jaeger H., *Echo State Network* (Scholarpedia, 2007).

is an important characteristic because it allows us to deal with a higher variety of problems, particularly nonlinear problems.

- *Input-Output Mapping.* Supervised learning is one of the methods to train neural networks. Each data sample in the training data consists of an input and a desired output. The network is trained by receiving these data samples randomly and then the network's synaptic weights keep being modified in order to minimize the network's result and the desired result. Providing the data samples in a different order also helps training the network and the train keeps going until the network reaches a steady state, where there are no significant changes in the synaptic weights. Thus, the neural network learns from the data contained in the training set by constructing an input-output mapping for the problem at hand.
- *Adaptivity.* Neural networks have the capability of changing synaptic weights according to the goals we have in hands. In some cases, when a neural network is trained to operate in a specific environment it can easily be retrained to deal with minor changes. Moreover, when a neural network is operating in a nonstationary environment it should be trained to deal with constant changes and the synaptic weights must adapt in real time. The ability to adapt a neural network according to the situation we have in hands is indeed a huge advantage, but this will affect the robustness of the same network. As the robustness level increases the adaptive level tends to decrease and vice-versa.
- *Evidential Response.* In the context of pattern classification, a neural network usually provides information about which pattern to select and also the confidence in the decision made. The confidence can be used to reject ambiguous patterns and thereby improve the classification performance of the network.
- *Contextual Information.* Knowledge is represented by the very structure and activation state of a neural network. Every neuron is potentially affected by the global activity of the other neurons in the network. Consequently, neural networks deal with contextual information very naturally.
- *Fault Tolerance.* A neural network, implemented in hardware form, has the potential to be inherently fault tolerant. In other words, in the presence of adverse operating conditions the neural network's performance will degrade gracefully.
- *Uniformity of Analysis and Design.* The same notation is used in different problems and environments where neural networks are used.
- *Neurobiological Analogy.* As previously mentioned, neural networks are inspired by the analogy with the brain. Neurobiologists look to artificial

networks as an interpretation tool to better understand neurobiological phenomena. On the other hand, we have engineers that try to develop more complex models using neurobiology as inspiration.

2.3.9 Disadvantages of neural networks

Although neural networks are extremely advantageous and outperform nearly every other machine learning algorithm, they also have some negative aspects:

- *Black Box.* Neural networks are extremely powerful but also very complex. The complexity of neural networks comes with the disadvantage of being hard to understand what is truly happening when unexpected results are presented. Hence, it is not advised to use neural networks in some domains because it might be hard to explain some results (Figure 2.6).
- *Duration of Development.* There are several libraries that help with the implementation of neural networks but sometimes it is important to be aware of details that might be important.
- *Amount of Data.* Of course, it depends on the problem in hands, but it is frequent that loads of data are required to train neural networks. It needs much more data when compared with other machine learning algorithms. The more data we possess to train our neural network the better it will operate.
- *Computationally Expensive.* The power of neural networks comes with the price of being extremely expensive. The amount of computational power depends how complex and big the neural network is and the amount of data there is to process.

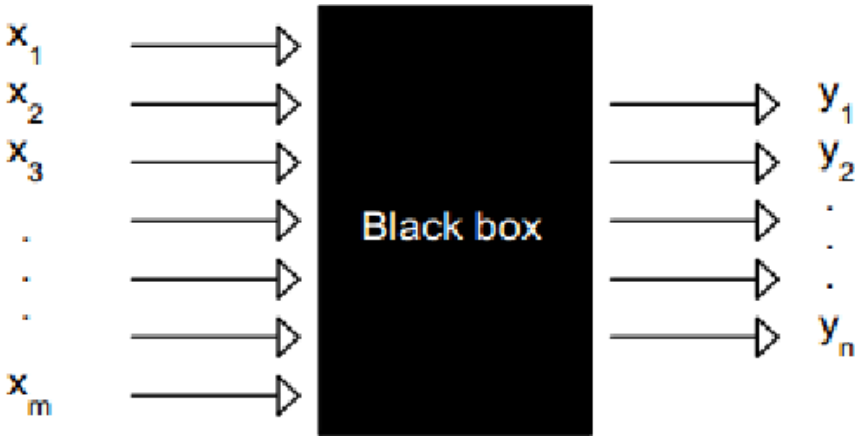

Figure 2.6 A neural network seen as a black box

2.3.10 Applications of neural networks

Neural networks have several applications. Some applications improved its results thanks to neural networks and I'm going to mention two of the most relevant:

- *Image Processing and Character Recognition.* Neural network's capability of receiving several inputs, process them and deal with non-linear relationships is playing a big role in image processing and character recognition. Character recognition is used in several problems like automatic number plates recognition, in airports, for passport recognition and information extraction and converting handwriting in real time to control a computer, for instance. Image processing and recognition are used in facial recognition, cancer detection and satellite imagery for agricultural and defense usage (Figure 2.7).
- *Forecasting.* This is an extremely important application of neural networks and is used in many different contexts. For instance, neural networks are used in weather forecasting, earthquake prediction, mathematical finance, astronomy, among others. All these problems are extremely complex and that's why they rely on neural networks.

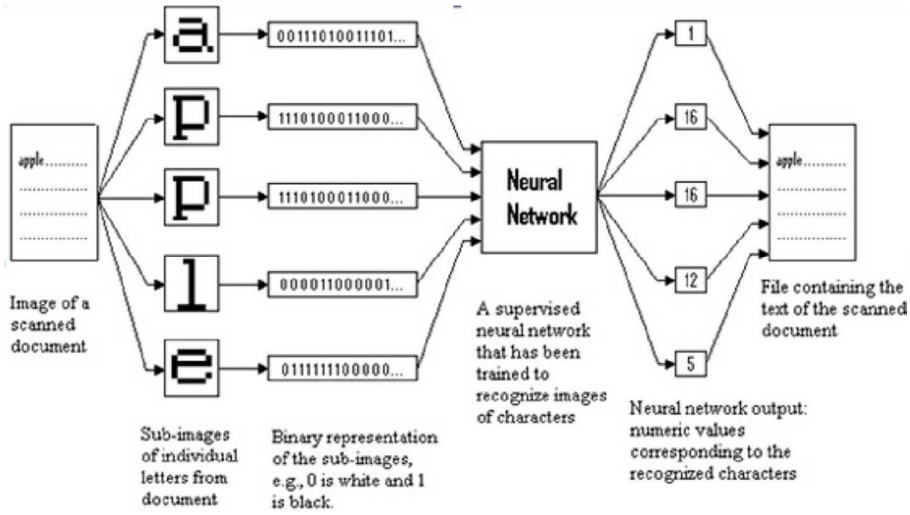

Figure 2.7 An example of how neural networks can be used in the context of Character Recognition problems. Adapted from "Application of Neural Networks to Character Recognition" by Dong Xiao Ni (2007)

2.3.11 Conclusions

Chapter 2 had the main goal of presenting a general overview of neural networks and other related algorithms. It is not possible to go in detail in every single topic because neural networks are a huge matter of research and its capabilities

have been growing over the past decades but the main concepts and ideas were presented so it is possible to understand the idea behind the work of this thesis.

3 Thesis plan

3.1 Introduction and initial plan

For this project, I always had in mind working with data sets that were related with a big issue that is becoming more and more evident as time goes by; the climate change.

I started researching and looking for data sets that looked interesting to work with and the initial plan was to work with a multivariate data set that aimed to predict if one day is either going to be an ozone day (meaning the ozone level is above a certain predefined threshold) or a normal day. This data set included several attributes including the temperatures at different times and in different atmospheric pressures throughout the day, the wind speed at various times of the day, the sea level pressure, the relative humidity, the total solar radiation for the day, among others. The neural network would learn from these attributes and their respective results (ozone day or not) and would be able to make predictions according to the attributes.

3.2 Second and definitive plan

I ended up deviating from the initial plan because I decided that I wanted to see the differences of how feedforward neural networks and recurrent neural networks deal with seasonal and non-seasonal data sets. I had in mind that feedforward neural networks would perform well on seasonal data sets because there is no need of memory since the pattern is very clear.

Still with the climate change subject in mind, I found two interesting related univariate data sets. The first one was the seasonal data set and it contained information regarding the mean monthly global temperatures for a period of 65 years over the twentieth century. The second one was the non-seasonal data set and it contained information regarding the yearly number of sunspots over a period of 314 years; a factor that greatly affects the temperatures and one that we can't control.

The Maunder Minimum was a period (1645-1715) in which the sunspots became exceedingly rare and this period roughly coincided with the middle part Little Ice Age, during which Europe and North America experienced colder than average temperatures. The volcanic activity is the current best explanation for the cause of this little ice age period, but the Maunder Minimum phenomena is also one of the factors that affected the temperature during this period.

Basically, I want to predict both data sets using feedforward neural networks and recurrent neural networks and then compare the performance of each of the algorithms. During the recurrent neural networks experimentations, I want to

compare Elman networks and Jordan network and check how impactful their small differences in structure affect the results.

I also had in mind performing another experimentation which involved predicting values for one data set while training the neural networks with the other data set. I ended up not doing this experimentation because as we know not only the sunspots affect the temperature; there are many other factors and therefore the results would be more accurate if we would use other data sets that would contain relevant information on these factors.

3.3 Technologies used

At first, I wanted to develop this project in Python, which is a language that I am comfortable with and it is commonly used to solve these types of problems. After a discussion with my supervisor, professor Andreas Wichert, who recommended me to use R, I decided to change and use this R language because it is a language more directed to implement machine learning techniques than Python. I was not as comfortable with R as I was with Python, but the learning process was fine; it didn't take me a long time to get used to the language. I was also familiar with the language from the SAD course (Sistemas de Apoio à Decisão) where I used this language to perform similar activities like data pre-processing and application of several data mining algorithms like SVM (Support Vector Machine), kNN (k-Nearest Neighbors), LVQ (Learning Vector Quantization), Decision Trees and even Neural Networks.

For IDE I chose R Studio, which I had some familiarity with it because I used it during the SAD course.

3.4 Evaluation method

As mentioned before, I plan on comparing the performance of both recurrent neural networks and feedforward neural networks and to accomplish this I will use measures of accuracy. The main measure used will be RMSE (root-mean-squared-error) and, when needed, I will recur to other measures to dissipate the doubts.

The evaluation process will consist of dividing the data sets in two parts: a training data set and a test data set. The training data set will be used to train our neural networks. Then we will compare the predictions from our models against the test data sets. For this, we will compare the results graphically and numerically using the previously mentioned measurement of accuracy and others if needed.

4 Comparing feedforward neural networks with recurrent neural networks

4.1 Choosing the data sets

To start off with my experimentations, I decided to pick two related data sets; a simpler one where the seasonality is very clear and there are not many deviations from the regular values and a more complex one where there are clearly more deviations from the pattern. Both data sets were obtained from datamarket.com and the first one is called “mean-monthly-temperature-1907-1972” and it contains 792 observations of the mean monthly global temperatures in Fahrenheit starting on January 1907 and ending on December 1972 (Figure 4.1). The second data set is called “yearly-mean-total-sunspot-number” and it contains 315 observations of the mean yearly number of sunspots starting on 1700 and ending on 2014 (Figure 4.2). I decided to choose these two data sets because there is a clear relation between them as I explained in the previous chapter. The number of sunspots is a huge factor on the temperatures around the planet. In Figures 4.3 and 4.4 I present two samples of both temperatures and sunspots data sets, respectively.

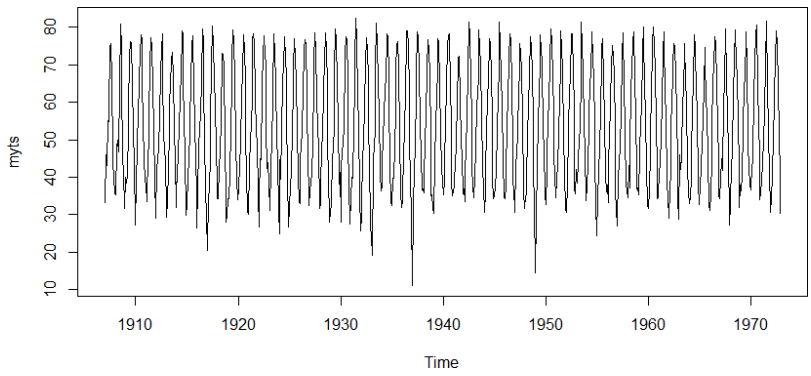

Figure 4.1 Mean monthly temperatures 1907-1972

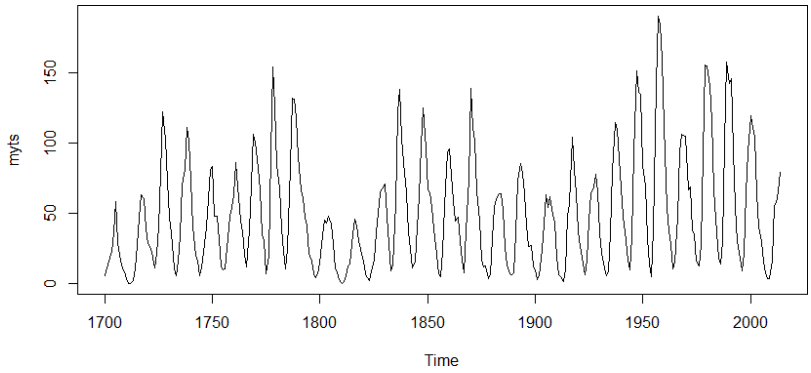

Figure 4.2 Yearly mean number of sunspots 1700-2014

	Month	Mean.monthly.temperature..1907...1972
1	1907-01	33.3
2	1907-02	46.0
3	1907-03	43.0
4	1907-04	55.0
5	1907-05	51.8
6	1907-06	57.5
7	1907-07	74.8
8	1907-08	75.6
9	1907-09	66.3
10	1907-10	46.2
11	1907-11	37.8
12	1907-12	37.6
13	1908-01	35.6
14	1908-02	35.2
15	1908-03	48.1
16	1908-04	50.0
17	1908-05	46.8
18	1908-06	64.6
19	1908-07	81.0
20	1908-08	74.9
21	1908-09	64.9
22	1908-10	45.5
23	1908-11	41.4
24	1908-12	31.6

Showing 1 to 24 of 793 entries

Figure 4.3 Temperatures data sample

	Year	Yearly.mean.total.sunspot.number
1	1700	5.0
2	1701	11.0
3	1702	16.0
4	1703	23.0
5	1704	36.0
6	1705	58.0
7	1706	29.0
8	1707	20.0
9	1708	10.0
10	1709	8.0
11	1710	3.0
12	1711	0.0
13	1712	0.0
14	1713	2.0
15	1714	11.0
16	1715	27.0
17	1716	47.0
18	1717	63.0
19	1718	60.0
20	1719	39.0
21	1720	28.0
22	1721	26.0
23	1722	22.0
24	1723	11.0

Showing 1 to 24 of 316 entries

Figure 4.4 Sunspots data sample

4.2 Pre-processing the data

It is important to mention this process since it can greatly affect the results. If we don't perform a correct pre-processing of the data, it will negatively affect the results. This process is a data mining technique that involves transforming raw data into an understandable format that can be used for several purposes, including analysis. Real world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends; therefore, it is likely to contain many errors. Some of the steps of data pre-processing include checking out for missing values, values normalization/ standardization, checking out the consistency of the data types, splitting the data into training and test data sets, among others.

Now I will describe the pre-processing method I followed before applying the neural network models.

First, I extracted the data sets as csv files from the data source. In R, I set the working directory to the respective directory where I downloaded my csv files to and obtained the files content to the program.

The pre-processing plan presented in the next two sub-sections was followed in every experimentation.

4.2.1 Pre-processing the temperatures data set

- I removed the last row which contained irrelevant information.
- The column containing the month and year was removed.
- The column containing the temperatures was defined as characters, so I had to turn the characters into numeric values.
- I turned the column containing the temperatures into a vector and applied a function that to normalize the data set (values of the temperatures ranging from 0 to 1).
- I split the data set in two parts: a training data set and test data set. I experimented with training data set containing 80% of the observations and the test data set containing 20% of the observations.
- Finally, I turned my training and test data sets into time series objects.

4.2.2 Pre-processing the sunspots data set

- I removed the last row which contained irrelevant information.
- The column containing the year was removed.
- The column containing the number of sunspots was defined as characters, so I had to turn the characters into numeric values.
- I turned the column containing the number of sunspots into a vector and applied the normalization function again.
- I also split the data set in two parts with the same sizes as I did in the first data set.
- I turned the training and test data sets into time series objects which are ready to be trained and analyzed.

In Figures 4.5 and 4.6 the time-series objects for both data sets I used for my experimentations are shown. Equation 4.1 shows the formula followed to obtain the normalized values.

myts	Time-Series [1:792] from 1907 to 1973: 0.31 0.489 0.447 0.615 0.57 ...
mytsTest	Time-Series [1:158] from 1960 to 1973: 0.437 0.301 0.287 0.357 0.5 ...
mytsTrain	Time-Series [1:634] from 1907 to 1960: 0.31 0.489 0.447 0.615 0.57 ...

Figure 4.5 Time series objects of the temperatures data set used to perform the experimentations

myts	Time-Series [1:315] from 1700 to 2014: 5 11 16 23 36 58 29 20 10 8 ...
mytsTest	Time-Series [1:63] from 1952 to 2014: 31.5 13.9 4.4 38 141.7 ...
mytsTrain	Time-Series [1:252] from 1700 to 1951: 5 11 16 23 36 58 29 20 10 8 ...

Figure 4.6 Time series objects of the sunspots data set used to perform the experimentations

$$X_n = \frac{X - X_{Nin}}{X_{Nax} - X_{Nin}} \quad (4.1)$$

where X_n is the normalized value, X corresponds to the value that we want to normalize, X_{Nin} is the minimum value of our data and X_{Nas} is the maximum value of our data.

For better understanding of the pre-processing involved in these experimentations the code is available for examination in Appendix A.

Since the pre-processing was done, we could now start preparing to deal with the core of our experimentations, which is to train our neural networks.

4.3 Experimentation introduction and notes

The results I'm about to present are in compliance with what I expected in the beginning with exception of one or two points that I will mention later on.

Before I present the results and conclusions I obtained, it is important to mention an important fact about the experimentations on this project; it is known that neural networks have hidden layers and these layers will be the main concern during our experimentations. There are no rules for choosing the number of hidden layers and hidden nodes and the only way to figure them out is by experimenting several values for them and analyzing if the results are improving or not. By default, the nnetar function, has a predefined number of hidden layers and hidden nodes. The hidden layer is one and the hidden nodes depends on the context of the problem. This predefined number of hidden nodes will be my starting point and I will proceed by testing with smaller and higher number of nodes in order to figure out which is the best number to obtain the best results possible. For the Elman and Jordan networks the process will be similar, but I can also choose the number of hidden layers of the network.

4.3.1 Results of the temperatures data set using a feedforward neural network

We trained our data set with a feedforward neural network. For this, we used the R function nnetar which is used for forecasting univariate time series and the architecture of this network is composed by a single hidden layer.

As aforementioned, the data set was divided into the training and test data sets. The training data set contained data from January 1907 to October 1959 while the test data set contained the rest from November 1959 to December 1972.

I called the nnetar function to train the network with the training set and create a model; then I forecasted the respective values for the time period corresponding to the test set using the predict function and compared the obtained results to the test set, graphically and numerically.

I tested the model with several number of hidden nodes. To compare the results, I used the accuracy function which returns several measures of the forecast accuracy. I spoke with my professor and he recommended me to use the RMSE (root-mean-squared-error) which is one of the most frequently used measures in times series forecasting problems. The RMSE value is obtained the following way:

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{i=1}^{N} (f_i - o_i)^2}{N}} \tag{4.2}$$

where f_i are the forecasts, o_i are the observed values (known results) and N is the sample size.

With the default number of hidden nodes (14) I obtained an RMSE of 0.05062419. I tested with higher and lower number of nodes and the best RMSE value was 0.04603968 when the number of hidden nodes was 2. The graphic (Figure 4.7) shows the comparison between the test set data (in red) and the prediction made by our model (in blue). It is important to refer that the graph contains the normalized values of the temperatures.

The results of these experimentations were the expected. Since the data is seasonal which means that it has a periodic, repetitive and generally regular and predictable pattern, the feedforward neural network model performed sufficiently well to make a good prediction.

Figure 4.8 shows the instructions that are used to train the network and to make predictions using our model.

These results were expected since the start because of the seasonality of the data set, which allows the feedforward neural network to perform well even without the existence of a memory.

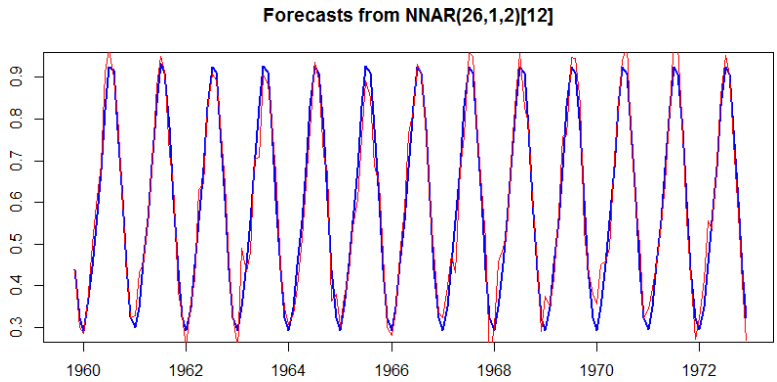

Figure 4.7 Comparison between the temperatures test set and the prediction of the feedforward neural network

```
#train the training time series
fit <- nnetar(mytsTrain, size=2)
#forecast using the trained time series
fcast <- predict(fit, h=158)
```

Figure 4.8 Instructions used to train and make predictions

4.3.2 Results of the sunspots data set using a feedforward neural network

For the number of sunspots data set the approach was very similar.

The data set was divided in two parts. The training data set contained data points from the starting year of 1700 and ending on 1951 while the test data set contained data from 1952 to 2014.

As mentioned, the procedure was similar; we called the nnetar function and it produced a model with its default values. The predict function was used to obtain the predictions of the number of sunspots for the years 1952-2014. After, we compared the results to our test data set.

As in the previous experimentation, we tested with several number of hidden nodes in order to figure out which is the best for this model. The default number of hidden nodes for this model was 5 and provided an RMSE value of 51.668736. I proceeded by testing with lower numbers of hidden nodes and the best I obtained was with 2 hidden nodes the value of 41.01420. For the higher values hidden nodes the best result was found for 7 hidden nodes and the RMSE value of 45.722987. The graphical representations of the models with 2 hidden nodes and 7 hidden nodes

are represented in Figures 4.9 and 4.10, respectively. Again, in blue we have the predictions from our model and in red the actual values.

So, as it is noticeable, for the model with 2 hidden nodes the results are not even remotely close to the desired ones. We can observe that our prediction almost converges to a straight line.

For the second model with 7 hidden nodes even though the results are not the best it is visually clear that they provide a result closer to our goal.

I decided to take into account two more measures that are clearly relevant to the context of this problem. I used the MPE (Mean Percentage Error) and the MAPE (Mean Absolute Percentage Error).

The MPE is the average of percentage errors by which forecasts of a model differ from actual values of the quantity being forecast. The value of MPE is obtained the following way:

$$\mathrm{MPE} = \frac{100\%}{n} \sum_{t=1}^{n} \frac{a_t - f_t}{a_t} \tag{4.3}$$

where a_t is the actual value of the quantity being forecast, f_t is the forecast, and n is the number of times for which the variable is forecast.

The MAPE measures the accuracy as a percentage and can be calculated as the average absolute percent error for each time period minus actual values A_t divided by actual values. Considering A_t to be the actual values, F_t the forecast values and n the sample size we can obtain the MAPE the following way:

$$\mathrm{MAPE} = \frac{1}{n} \sum_{t=1}^{n} \left| \frac{a_t - f_t}{a_t} \right| \tag{4.4}$$

For the 2 hidden nodes model we have an MPE of -136.8492 while MAPE is 166.6492. For the 7 hidden nodes model we have an MPE of -54.22009 and an MAPE of 94.26946.

For this specific case, we can conclude that the second model is overall better than the first one.

The results were not surprising. The performance was poor, as we can see from the accuracy values we obtained. This is since our data set is non-seasonal, which requires our network to have memory in order to perform well.

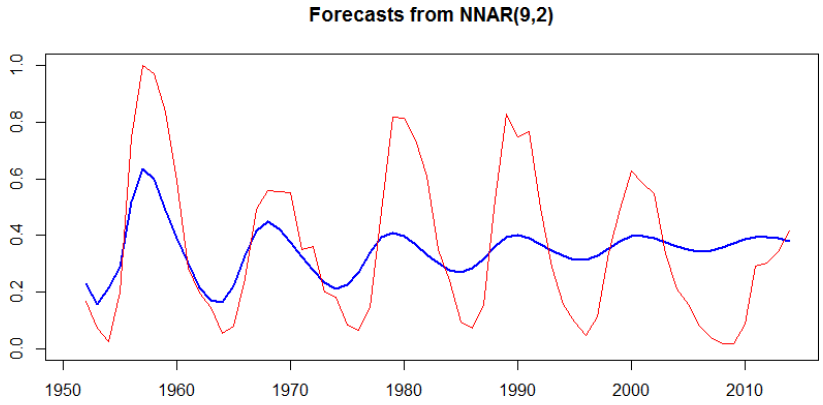

Figure 4.9 Comparison between the sunspots test set and the prediction of the feed-forward neural network with 2 hidden nodes

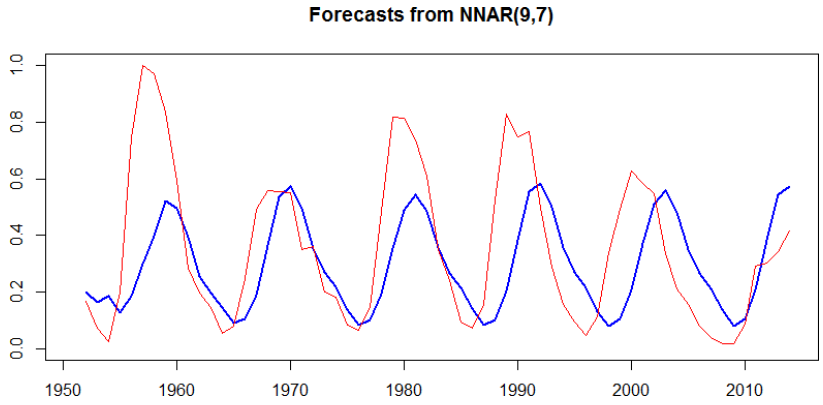

Figure 4.10 Comparison between the sunspots test set and the prediction of the feed-forward neural network with 7 hidden nodes

4.3.3 Results of the temperatures data set using Elman and Jordan neural networks

I will now experiment with the same data sets using recurrent neural networks, more specifically Elman and Jordan neural networks. These networks were designed to implement memory in order to perform better in other type of problems like time varying sequences. For the first problem, I already obtained a pretty decent result with the feedforward neural network mainly because the data is seasonal, but for our second problem, where the data is non-seasonal, I want to improve the results with the following experimentations.

To achieve better results, I trained my model with lagged versions of the original time series. In this case I used 12 lagged versions which corresponds to the 12 months of the year. This method is commonly used in times series because of a phenomenon called autocorrelation, which is a tendency for the values in a time series to be related to previous copies of itself. This is important to identify patterns which will help determining the seasonality of the series.

I started experimenting with a single hidden layer. I called the elman function with the different numbers of hidden nodes and then the predict function. In the end, the respective RMSE values were calculated. The best value for the RMSE I obtained was 0.04949789 for 6 hidden nodes. After this, I started experimenting with 2 hidden layers. For 2 hidden layers, I decided to automate the process with a cycle since it would consume a lot of time otherwise. By the end of the loop, my program would return the best RMSE value and the corresponding number of hidden nodes of each hidden layer. The best results obtained were for 3 hidden nodes for both of the layers. The corresponding RMSE value was 0.04879495. It is important to mention the parameters that I used to call the elman and jordan functions; the “inputs[train]” contains the values corresponding to the training component (80%) of all the lagged versions of the time series, the “outputs[train]” contains the values corresponding to the training component (80%) of the original time series, “size=c(i,j)” corresponds to the number of hidden nodes in the corresponding hidden layers, “learnFuncParams=c(0.1)” is the parameter used for the learning function and “maxit=5000” corresponds to the number of iterations that we use to train our model. The following Figures 4.11, 4.12 and 4.13 show the comparison between the prediction and test set, the iterative error plot, and the instruction used to train the neural network, respectively. The first graph has the normalized values of the temperatures and the bottom values are the years of the prediction. The iterator error plot shows us how the network error evolved along the training iterations.

The results here were fine as expected. The prediction of both the recurrent neural networks were fine with good accuracy values. The unexpected part is that our feedforward model outperformed the recurrent models.

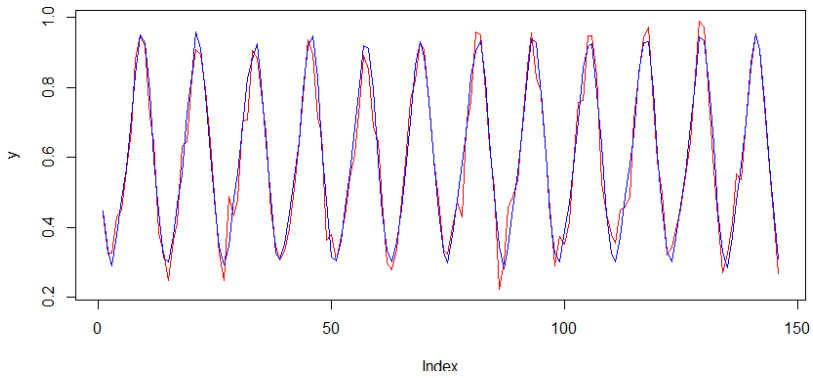

Figure 4.11 Comparison between the temperature test set and the prediction of the Elman neural network

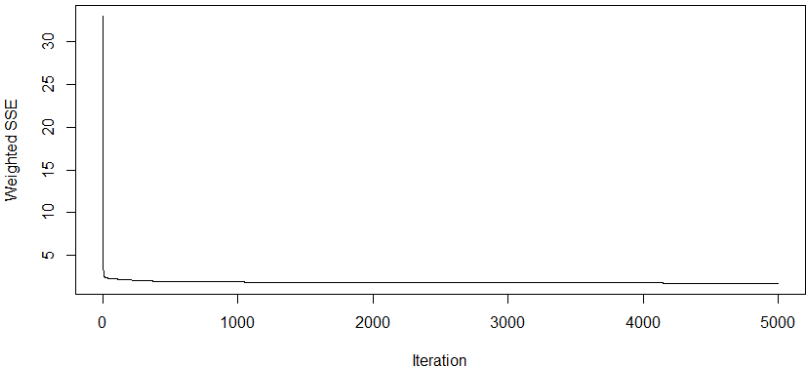

Figure 4.12 Iterative error plot for the Elman network on the temperatures data set

```
#elman network
fit<-elman(inputs[train],
           outputs[train],
           size=c(i,j),
           learnFuncParams=c(0.1),
           maxit=5000)
```

Figure 4.13 Instruction used to call the elman function

Now it is time to train our data with the Jordan network. The setup was the same of the Elman network. The only difference is that I called the jordan function instead. The explanation of the parameters is analogous to the elman network and the parameters used in the function call were also the same. The best result achieved for this network was with an RMSE value of 0.0495885 for a

single hidden layer with 4 nodes. As it is not possible to have more than a hidden layer in Jordan networks, we will compare the results between the best values of RMSE between the two networks with one hidden layer. In this case the Elman network performed slightly better; the difference is minor though. The respective prediction is in Figure 4.14 and the instruction is in Figure 4.15.

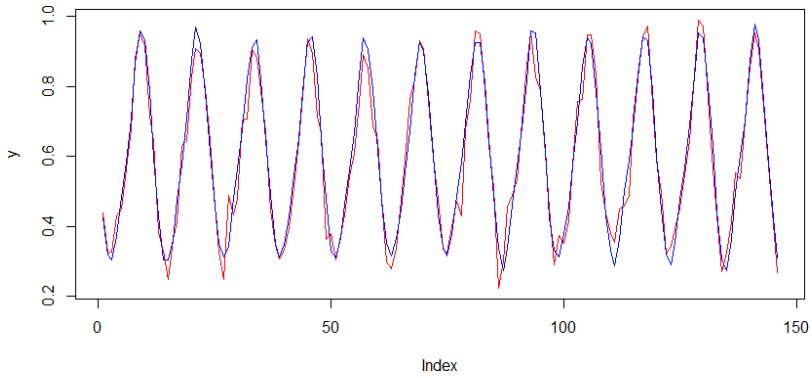

Figure 4.14 Comparison between the temperature test set and the prediction of the Jordan neural network

```
#jordan network
fit<-jordan(inputs[train],
            outputs[train],
            size=c(3),
            learnFuncParams=c(0.1),
            maxit=5000)
```

Figure 4.15 Instruction used to call the jordan function

4.3.4 Results of the sunspots data set using Elman and Jordan neural networks

The procedure for this experimentation was the same presented in the previous section 6.3.3.

For this non-seasonal data set, the best results I obtained for the Elman network were an RMSE value of 0.06965678 with two hidden layers containing 5 and 2 hidden nodes. For a single hidden layer, the best RMSE value was 0.08226485.

For the Jordan network the best value for the RMSE was 0.08479622 corresponding to a single layer with 7 hidden nodes.

In this case, the Elman network outperformed the Jordan network again.

The following Figure 4.16 and 4.17 show the comparisons for the Elman and Jordan networks, respectively.

The results obtained were expected. The recurrent neural networks performed well on this non-seasonal data set and the values of the accuracy for the predictions are very good. The existence of a memory in our models, for non-seasonal time series forecasting problems, is crucial for the positive results.

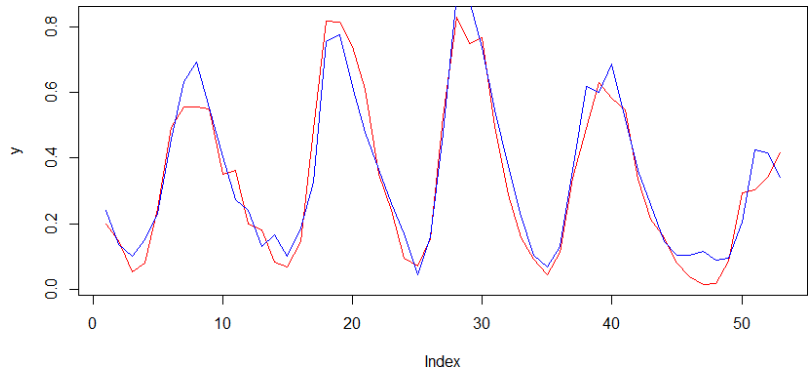

Figure 4.16 Comparison between the sunspots test set and the prediction of the Elman neural network

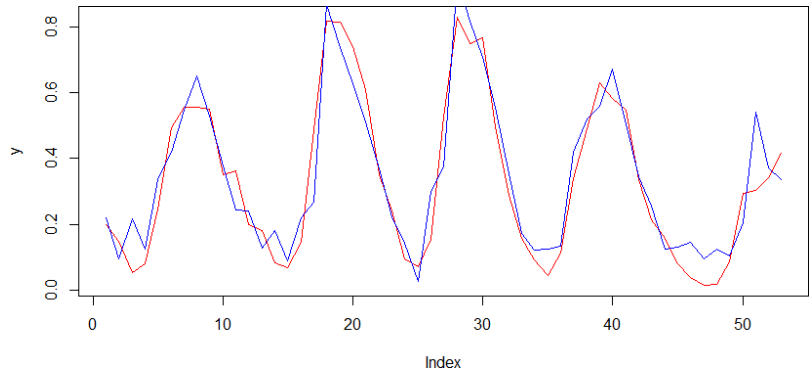

Figure 4.17 Comparison between the sunspots test set and the prediction of the Jordan neural network

4.4 Conclusions of the experimentations

From these experimentations, I can conclude that for the seasonal dataset the feedforward neural network performed sufficiently well. On the other hand, for the non-seasonal dataset the results were not even close to satisfactory. Using the recurrent neural networks, either Jordan network or Elman network, the results were positive for both the seasonal and non-seasonal dataset.

The main conclusion we can take is that recurrent neural networks are a better approach when we are dealing with time series.

5 Conclusion

5.1 Results

To sum up the obtained results I present them in the following tables of Figures 5.1 and 5.2. In the first table we have the RMSE values corresponding to the networks with one hidden layer.

	feedforward neural network	elman neural network	jordan neural network
temperatures data set	0.04603968	0.04949789	0.0495885
sunspots data set	51.668736	0.08226485	0.08479622

Figure 5.1 Comparison between all the networks with one hidden layer

	feedforward neural network	elman neural network	jordan neural network
temperatures data set		0.04879495	
sunspots data set		0.06965678	

Figure 5.2 Results from the Elman network with two hidden layers

For the temperatures data set, we can conclude that the feedforward neural network was the one that performed the best. This is a surprising result for me, I was expecting it to perform well but not to beat the performance of both the recurrent neural networks; as mentioned before, the good result is explained by the seasonality of the first data. Between the Jordan and Elman networks the results are very similar even though the Elman network value is slightly better.

Regarding the sunspots and the feedforward neural network the expected happened; the network performance was extremely poor in the non-seasonal data set, which can be explained by the fact that this type of network doesn't have a type of memory like the recurrent neural networks do. Between the Jordan and Elman networks the value obtained from the Elman network is again slightly better.

In the second table, where we have the values for two hidden layers, we obtained better values. This happened because the network is deeper; an extra layer will allow the network to boost its performance when dealing with more complex, non-linear problems.

So, the main conclusion we can infer from these results are:

- Feedforward neural networks are worth considering we are dealing with seasonal time series data sets; they might even outperform recurrent neural networks or other techniques.
- Elman neural networks slightly outperformed Jordan neural networks but the difference is not significant, so it is worth taking both into account when dealing with these problems.
- The deeper the neural network is the better it will perform with complex problems. It is important to note that increasing layers will also increase the cost and the time consumed, and sometimes this tradeoff is not worth because the results might not be that much better or even, in some cases, they might get worse.

5.2 Improvements

Even though we chose to work with neural networks, other algorithms (some of them described in Chapter 2) would be able to solve this type of problems and maybe with even better results. Regarding the neural networks, we could try different activation functions.

We used training and test data sets containing 80% and 20% of the data, respectively. We could improve the results by trying different sizes for these partitions. We could also use different parts of the data for these partitions; for example the training data set being the last years of the data set and the test data set being the beginning or even split the test data set and make part of the prediction in the beginning and part of the prediction in the end.

5.3 Difficulties

As previously mentioned, it took me some time to get used to the R language.

The pre-processing was also a part that was time consuming; it was hard to figure out some things like, for instance, that some variable were not defined as the type they should be and that I had to turn my vectors into time series objects.

During the model training part, the hardest parts were figuring out what were the best accuracy measures to compare the results and researching on the neural networks function and the meaning of their parameters.

Certainly, I faced more problems, but these are the ones that come to mind and the ones that I struggled the most with.

I am satisfied with the way I overcame these difficulties and managed to complete my work.

5.4 End notes

Through the realization of this work, I learned a lot about neural networks, which caught my attention and had been intriguing me since the SAD course. I learned about different types of neural networks as well as other algorithms that can be used to accomplish the same goals. During the realization of the experimentations, I gained a deeper view on how some of these machine learning techniques work.

Overall, it was an enriching experience that allowed me to learn more and improve in one of the areas that has been captivating me.

6 Bibliography

1. Shumway, R.H. and Stoffer, D.S. (2000). *Time Series Analysis and Its Applications*. Springer International Publishing.
2. Cowpertwait, P.S.P and Metcalfe A.V. (2009). *Introductory Time Series with R*. Springer-Verlag New York.
3. Montgomery, D.C. , Jennings, C.L. and Kulahci, M. (2008). *Introduction to Time Series Analysis and Forecasting.* Wiley
4. McCulloch, W.S. and Pitts, W.H. (1943). *A Logical Calculus of the Ideas Immanent in Nervous Activity.*
5. Hagan, M.T. , Demuth, H.B. , Beale, M.H. and De Jesús, O. (1996). *Neural Network Design.* Martin Hagan.
6. Haykin, S.S. (1993). *Neural Networks and Learning Machines.* Pearson.
7. Medsker, L.R. and Jain, L.C. (1999, Dec 20). *Recurrent Neural Networks: Design and Applications.* CRC Press.
8. Skapura, D.M. (1996). *Building neural networks.* Addison-Wesley Professional.
9. Casdagli, M. And Eubank, S. (1992, Jun 20). *Nonlinear Modeling and Forecasting*. CRC Press.
10. Gerven, M. V. and Bohte, S. (2017, Dec 19). *Artificial Neural Networks as Models of Neural Information Processing.* Frontiers in Computional Neuroscience.
11. Shmueli, G. and Lichtendahl Jr., K.C. (2015, Jul 17). *Pratical Time Series Forecasting with R: A Hands-On Guide.* Axelrod Schnall Publishers.
12. Russel, S.J. and Norving, P (2010). *Artificial Intelligence: A Modern Approach.* Prentice Hall.
13. Kleene, S.C. (1956). *Representation of Events in Nerve Nets and Finite Automata.* Annals of Mathematics Studies.
14. Donald, H. (1949). *The Organization Of Behaviour.* New York: Wiley.
15. Werbos, P.J. (1975). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.*
16. Schmidhuber, J. (2015). *Deep Learning in Neural Networks: An Overview.*
17. Ivakhnenko, A.G. and Grigor'evich Lapa, V. (1967). *Cybernetics and forecasting techniques.* American Elsevier Pub. Co.
18. Behnke, S. (2003). *Hierarchical Neural Networks for Image Interpretation.* Lecture Notes in Computer Science. Springer.
19. Graves, A. and Schmidhuber, J. (2009). *Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks.*

20. Ciresan, D.; Giusti, A.; Gambardella, L.M. and Schmidhuber, J. (2012). *Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images.* Curran Associates, Inc.
21. Graupe, D. (2013. *Principles of Artificial Neural Networks.* World Scientific.
22. Crick, F. (1989). *The recent excitement about neural networks.* Nature.
23. Yin-Wen, C.; Cho-Jui, H.; Kai-Wei, C.;Ringgaard, M.; Chih-Jen, L. (2010). *Training and testing low-degree polynomial data mapping via linear SVM.* Journal of Machine Learning Research.
24. Fan, J. (1996). *Local Polynomial Modelling and Its Applications: From linear regression to nonlinear regression.* Chapman & Hall/CRC.
25. Yule, G.U. (1927). *On a Method of Investigating Periodicities in Disturbed Series with Special Reference to Wolfer's Sunspot Numbers.* Philosophical Transactions of the Royal Society of London.
26. Theodoridis, S. (2015). *Machine Learning: A Bayesian and Optimization Prespective.* Academic Press.
27. Box, G.; Jenkins, G.M.; Reinsel, G.C. (1994). *Time Series Analysis: Forecasting and Control.* Prentice-Hall.
28. *Notation for ARIMA Models. Time Series Forecasting System.* SAS Institute.
29. Geman, S.; Bienenstock, E. and Doursat, R. (1992). *Neural networks and the bias/variance dilemma.* Neural Computation.
30. Sak, H.; Senior, A. and Beaufays, F. (2014). *Long Short-Term Memory recurrent neural network architectures for large scale acoustic modeling.*
31. Milos, M. (2012). *Comparative analysis of Recurrent and Finite Impulse Response Neural Networks in Time Series Prediction.* Indian Journal of Computer and Engineering.
32. Kosko, B. (1988). *Bidirectional Associative Memories.* IEEE Transactions on Systems, Man, and Cybernetics.
33. Hinkelmann, K. *Neural Networks.* University of Applied Science Northwestern Switzerland.
34. Nielsen, M.A. (2015). Chapter 6. *Neural Networks and Deep Learning.*

7 Appendix A

Here we can find the pre-processing code for both data sets used in our experimentations (first for the temperatures data set and secondly the sunspots data set).

Temperatures data set:

```
#normalization function (n)
range01 <- function(x){(x-min(x))/(max(x)-min(x))}

#read data
MyData <- read.csv(file="mean-monthly-temperature-1907-1972.csv", header=TRUE, sep=",")
#remove last irrelevant row
MyData <- MyData[-c(793),]
#remove month column
MyData$Month <- NULL
#from char to num
MyData$Mean.monthly.temperature..1907...1972 <-
as.numeric(as.character(MyData$Mean.monthly.temperature..1907...1972))
#turn into a vector
MyDataVector <- as.vector(unlist(MyData))
#normalize vector (0,1)
MyDataVector <- range01(MyDataVector)
#train vector
MyDataVectorTrain <- MyDataVector[1:634]
#test vector
MyDataVectorTest <- MyDataVector[635:792]
#turn into a time series
myts <- ts(MyDataVector, start=c(1907, 1), end=c(1972, 12), frequency=12)
#train time series
mytsTrain <- ts(MyDataVectorTrain, start=c(1907, 1), end=c(1959, 10), frequency=12)
#test time series
mytsTest <- ts(MyDataVectorTest, start=c(1959, 11), end=c(1972, 12), frequency=12)
```

Sunspots data set:

```
range01 <- function(x){(x-min(x))/(max(x)-min(x))}

MyData <- read.csv(file="yearly-mean-total-sunspot-number.csv", header=TRUE, sep=",")

MyData <- MyData[-c(316),]

MyData$Year <- NULL

MyData$Yearly.mean.total.sunspot.number <-
as.numeric(as.character(MyData$Yearly.mean.total.sunspot.number))

MyDataVector <- as.vector(unlist(MyData))

MyDataVector <- range01(MyDataVector)

MyDataVectorTrain <- MyDataVector[1:252]

MyDataVectorTest <- MyDataVector[253:315]

myts <- ts(MyDataVector, start=c(1700), end=c(2014), frequency=1)

mytsTrain <- ts(MyDataVectorTrain, start=c(1700), end=c(1951), frequency=1)

mytsTest <- ts(MyDataVectorTest, start=c(1952), end=c(2014), frequency=1)
```