DETERMINATION OF PARAMETER INFLUENCE ON ANN TRAINING ALGORITHMS FOR TIME SERIES FORECASTING

THESIS

A thesis submitted for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

presents

Ranyart Rodrigo Suárez Ponce de León

Mario Graff Guerrero

Thesis Advisor

Juan J. Flores Romero

Thesis Coadvisor

Universidad Michoacana de San Nicolás de Hidalgo

August 2013

Abstract

Artificial neural networks (ANN) are an effective tool for learning the characteristics of a variety of problems such as: time series forecasting, stock market, speech recognition and robotics, among others. Recently, a number of works have been proposed to design automatically an ANN given a particular problem. However none of these works is centered in the training parameters. This work has two main objectives, the first objective is to determine the influence that the training parameters have on the performance of a trained ANN, this is, how smalls changes in the values for the training parameters affect the behavior and the performance of the ANN. The second objective is to model the expected performance of an ANN, as accurately as possible, in order to tackle the algorithm selection problem. Thanks to this modelling of the performance, new problems can be analyzed and obtain an approximation of the performance of the ANN over these problems without the need of training it over the problems.

The parameter influence was developed for two algorithms, the QUICKPROP and RPROP training algorithms. Both algorithms are improved versions of the traditional Back propagation, and have more parameters than the later making more difficult the appropriate selection of values for the algorithm. With the methodology proposed, this work obtain the parameter influence as a list with the parameters sorted according to their influence on the performance of the ANN. The modelling presents how a real ANN and a modelled ANN had similar performance over a set of problems validating the modelling methodology.

Resumen

Las redes neuronales artificiales (RNA) son herramientas del área de Inteligencia Artificial, las cuales están diseñadas para aprender las carcterísticas de una gran variedad de problemas como por ejemplo: predicción de series de tiempo, predicción de la bolsa de valores, reconocimiento de audio y robótica. Recientemente, algunos trabajos han sido desarrollados enfocados a diseñar automáticamente una RNA. No obstante, ninguno de estos trabajos está enfocado en el rol que juegan los parámetros de los algoritmos de entrenamiento de una RNA. Este trabajo tiene dos objetivos principales, el primer objetivo es determinar la influencia que tienen los parámetros del algoritmo de entrenamiento de una RNA sobre el performance final de la red entrenada, en otras palabras, cómo pequeños cambios en los valores de los parámetros afecta tanto el comportamiento de la red como su performance. El segundo objetivo de este trabajo es modelar el performance de una RNA, lo más exacto posible, para resolver el problema de selección de algoritmos. El hecho de modelar el performance de una RNA permite obtener una aproximación de cómo se comportará una RNA sobre una colección de problemas sin la necesidad de realmente entrenarla sobre dichos problemas.

La influencia de los parámetros ha sido determinada para dos algoritmos de entrenamiento: QUICKPROP y RPROP. Ambos algoritmos representan una versión mejorada del tradicional algoritmo de propagación hacia atrás (Back propagation), en ambos casos los algoritmos cuentan con más parámetros que éste, haciendo más difícil la correcta selección de valores para los parámetros. La principal razón para modelar una RNA es para poder enfrentar el problema de selección de algoritmos. El modelado de una RNA arroja información acerca de cómo se comportará una RNA sobre una colección de problemas en los cuales no se ha entrenado dicha RNA.

Contents

List of Symbols

x_i	i_{th} input of an ANN.
y_i	i_{th} desired output of an ANN.
(X_i, y_i)	i_{th} training sample.
w_i	Connection weight of the i_{th} input of an ANN.
S	Sum of products between inputs and connection weights.
$f(S)$	Activation function of a Neuron.
k	Output of a Neuron.
s	Steepness of an Activation function.
θ	Set of training parameters.
Θ	List containing all the sets of training parameters.
p	Problem instance.
P	List containing all the problem instances considered.
M	Matrix of explanatory variables used by Random Forests.
M_1	Matrix of time series Characteristics.
M_2	Matrix of time series Characteristics.
γ	Performance of an ANN.
$\hat{\gamma}$	Expected performance of an ANN.

List of Acronims

PID	Parameter Influence Determination.
ANN	Artificial Neural Network.
$QUICKPROP$	Quick Back Propagation Algorithm.
$RPROP$	Resilient Back Propagation Algorithm.
$CART$	Classification and Regression Trees.
EA	Evolutionary Algorithm.
GA	Genetic Algorithm.
ES	Evolutionary Strategy.
PSO	Particle Swarm Optimization.
ACO	Ant Colony Optimization.
$ARIMA$	Autoregressive Integrated Moving Average.
MSE	Mean Squared Error.
$sMAPE$	Symmetric Mean Abosolute Percentage Error.
SVM	Support Vector Machines.
RAW	Unmodified Data.
TSA	Trend and Seasonality Adjusted Data.
$M1$	M1 Time Series Competition.

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction

1.1. Introduction

An Artificial Neural Network (ANN) is an accurate model mainly used for classification and regression problems. In machine learning, classification is the process of determining the class of an individual from a set of categories. An ANN classifies individuals based on its features or characteristics, which have to be presented to it as inputs. Another main use for ANNs are regression problems. We face a regression problem when, given a set of independent variables, we determine the value of the dependent variable. One example of this problem is time series forecasting, which will be addressed in this work.

ANNs have been used in many areas: economics ([Udo, 1993], [White, 1988], [Chen et al., 2003], [Herbrich et al., 1999]), medicine ([Rouhani and Soleymani, 2009], [Saritas, 2012], [Tourassi et al., 1995], [Ravdin et al., 1992], [Fogel et al., 1995]), engineering ([Tsoukalas and Uhrig, 1996], [Flood and Kartam, 1994], [Hagan and Demuth, 1999], [Narendra and Parthasarathy, 1990], [Cochocki and Unbehauen, 1993]), and face recognition ([Er et al., 2002], [Rowley et al., 1998], [Lawrence et al., 1997], [Ma and Khorasani, 2004]), among others. Several features of artificial neural networks make them valuable and attractive for a forecasting task: ANN's are data-driven and self-adaptive methods in which there are few *a priori* assumptions about the problems under study. After learning the data presented to them (inputs or samples), ANN's can often correctly infer the unknown part of the samples even if the data contains noise. ANN's are universal function approximators [Hornik et al., 1989], i.e., it has been proven that an ANN can approximate any continuous function to any desired accuracy.

Even though ANNs have these features, their use is still far from being trivial. In fact, most of the time, the use of ANN are left to those persons familiarized with this technique. That is, they are used by experts. There have been a number of procedures (see Section 1.6) that try to overcome the problem of leaving the use of ANNs only to experts. The present contribution fits into those works that help non-experts to choose the most appropriate ANN architecture for their problem. However, there are notorious differences between the works found in the literature and our present procedure.

The majority of works are aimed to help the user to use ANNs successfully; most of them are concerned with the design of the ANN itself. The present work is different from those because it is not focused in exploring the different architectures from ANNs, and then compare which particular architecture presents the best performance. The word performance is a quantitative measure that describes how well the ANN solves the task for which it was designed. This work is centered in the algorithm that makes an ANN able to learn, which is called *training algorithm*. First, this work is interested in understanding the role of each of the training parameters with respect to the performance of the network. That is, identify the training parameters that influence the most the performance of the ANN. To this end, we build a linear equation of the performance and use a feature selection algorithm in this equation. Finally, we complement this study by presenting to the reader those parameters that produce the best networks.

1.2. Problem definition

The problem addressed in this thesis is to determine the influence of the training parameters on the performance of the ANN. More formally, this can be represented in a function Φ that receives the training parameters, a problem instance, and the result is the performance of the ANN. The function defines a relationship between a set of training parameters and a measure of performance, for a given problem. This function is presented in Equation (1.1), where θ is a vector that contains the parameters for the training algorithm and p is a problem instance. The output of this function is a real value that will be called γ, which is the fitness of the ANN, i.e., the performance of the ANN.

$$\Phi : \theta \times p \rightarrow \mathbb{R} \tag{1.1}$$

Once this model Φ has been used to determine the influence of the training pa-

rameters on the performance of an ANN (this is explained in Chapter 3), then this work models an ANN with fixed or static θ training parameters. This modelling can also be seen as a function Δ which receives a problem and its output is the expected performance of the ANN. This function is depicted in Equation (1.2), where p is the problem instance of which the expected performance will be modelled. The output of the function is a real value that represent the expected performance of the ANN and is denoted by $\hat{\gamma}$.

$$\Delta_\theta : p \rightarrow \mathbb{R} \tag{1.2}$$

The goal of the modelling is that, given a fixed set of values for the training parameters, and a problem instance, the resulting expected performance ($\hat{\gamma}$) has to be as close as possible to the real performance of the ANN (γ), this is, $\gamma \approx \hat{\gamma}$. The advantage of modelling an ANN is that, we only have to feed this function with a new collection of problems to obtain an approximation of the performance without the need of training the real ANN and proving it over this new collection.

The influence of the training parameters is determined for the training and validation stages, which will be explained in Section 2.5. The modelling also models ANNs in the training and the validation stages. The performance of the ANN is given by γ_j, where the sub-index j could be T or V. γ_T is the performance of the ANN in the training stage, this is, how well the ANN learned the problem under analysis. γ_V denotes the performance of the ANN in the validation stage, in other words, how well the ANN generalizes unknown data.

1.3. Motivations

ANNs are methods that require a great user effort when used in real world applications. They also require higher computational resources because ANNs have a large number of connections whose values have to be set appropriately. Due to this, the algorithms designed to accomplished this task are much slower than other techniques, where the problem is to find the parameters of a linear equation (like in ARIMA models for example). This means that these computer programs need more CPU power and more memory to run.

As computers become faster and more powerful, making ANNs viable methods, it would be expected that many works will be developed to understand them. As the interest in ANNs has increased, the number of works related to them has also increased. Unfortunately,

a great number of these works are not very practical. Most of these works that are centered in ANNs are focused on the ANN design and not on the way ANNs learn. This work is aimed at producing information about how the training algorithm and its parameters affect the performance of an ANN. This means that all the information resulting from this work can be combined with previous works (see Section 1.6) related to ANNs design in order to improve results.

The aim of this work is to help users who are not familiarized with how to set properly the parameters of an ANN's training algorithm. If the values of the training parameters are well chosen, the resulting ANN will be a good approach. On the other hand, if these values are not good for the training algorithm, the ANN will perform extremely bad. The training algorithms have default parameter values that will normally perform well. Sometimes, these default values are not adequate in some problems, leading to an ANN with deficient performance. If we start to change these default values we will experience many different behaviors of the ANN, even with small changes for the values. Unfortunately, there is no guide upon which we can base our decisions in order to change the parameters values wisely. Based on the conclusions of this work, an unexperienced user will have a starting point in changing the training parameter values.

1.4. Objectives

In this work, ANNs have been used for the Time series prediction problem. This work is not focused in comparing ANNs against other approaches designed to solve the same problem. This work is interested in how the values of the training parameters affect the performance of the ANN. The main objective of this thesis is to identify which training parameters are more correlated with the performance of the ANN. In other words, we want to identify which parameters are the most influential for a given ANN's training algorithm.

The second objective of this work is to model an ANN, based on the most influential parameters. The reason is that there is not a way to know *a priori* what can be the expected performance of a particular ANN without actually test it. In order to overcome this problem, a modelled ANN can be very helpful; it would represent an approximation of the real ANN. ANNs are computationally expensive methods and sometimes, for a given problem, there are simpler models that achieve the same performance that ANNs or they even outperform ANNs using less resources.

1.5. Achievements

Influence of the training parameters on the ANNs performance

The influence of the parameters for the QUICKPROP and RPROP training algorithms has been determined. This influence of parameters is represented by sorting the parameters based on the influence that each parameter has on the performance of the ANN. This was done solving a regression problem using a method called *Random Forests* (this method will be mentioned in Chapter 3). Furthermore, some values were identified that perform better than the default values proposed in the literature and are shown in this work.

Tuning of the training parameters

The results of this thesis show that certain parameters had very little influence on the behavior of the ANN. While, the training algorithm is very sensitive to other parameters. A good idea about tuning the training parameters, is to put more effort tuning those parameters whose influence on the performance is greater. In order to make this tuning of parameters easier, the user can guide their own tuning process based on the parameter influence determined in this work.

Algorithm selection problem

We have modelled an ANN with fixed values of the training parameters with a good percentage of success. This means that the results obtained with the modelling are very close to the ones obtained by the real ANN. The modelling process is specially useful when the user is facing the Algorithm selection problem. The modelling gives information about the expected performance of an ANN given a set of problems. This means that, we could know *a priori* the performance of the ANN over a collection of problems without actually training the ANN.

1.6. Literature Review

In the literature there is a considerable number of works that are aimed at tuning the majority of parameters of an ANN. These works try to find suitable values for ANN's

parameters like: ANN's structure, number of layers, number of neurons per layer, activation functions, steepness of the activation functions etc. One work into this category is [Yao et al., 1996]. In the majority of the cases, this tuning of ANNs parameters is achieved by the use of other algorithms like Genetic Algorithms (GAs), Evolutionary Algorithms (EAs), etc.

One of the approaches that exploits the advantages of both GAs and ANNs is [Zhang and Wang, 2008]. In this work, GAs are used to optimize the initial interconnecting weights of the ANN, and then the traditional Back propagation algorithm is performed to train the network. In other words, the parameters of the ANN are not tuned to increase the performance. Instead, the initial ANNs weight connections are set by some other rule given by GAs; they are not initialized randomly. Another approach that uses GAs to tune some ANNs parameters is [Reinaldo et al., 2009]. Nonetheless, this work differs slightly from others that also use GAs because it is a hybrid method which also uses Machine Learning (ML) and probabilistic techniques.

Algorithms and techniques like GAs are not the only ones used to tackle the problem of tuning the ANNs parameters. Other approaches like EAs are also used for the same purpose. An example of these kind of works, where EAs are used to tune ANN's parameters is [Almeida and Ludermir, 2008]. In this work, EAs are used to evolve different ANN models and try to find certain common parameters values that present good performance. An evolutionary search is performed to tune initial weights, activation functions, architectures, and learning rules of the ANN. Another approach that uses EAs is [Falco et al., 1997], in this work, the EA is used to tune the design of the architecture of the ANN and the best learning method (training algorithm) is also tuned for the non-linear system identification problem.

Besides of the previous works reviewed in this section, where other techniques like GAs and EAs are applied to tune some parameters of an ANN, there are other approaches which perform the opposite way. This is, ANNs are used to tune parameters of other algorithms. One of these kind of approaches is [Dobslaw, 2010], where the author use ANNs to find the best initial configurations for Particle Swarm Optimization (PSO), Evolutionary Algorithms (EA) and, Ant Colony Optimization (ACO). In this work, the ANNs inputs are the problem instance properties and the outputs are the recommended values for that particular problem instance. The former work had gotten our attention because they used a particular ANN model to find the quasi-optimal values for the parameters of PSO, EA

and, ACO.

All the previously mentioned works have something in common: they are approaches that develop methodologies to tune ANNs parameters. However, none of these works is centered only in the ANNs training parameters. Moreover, those approaches are not interested in determining what the influence of the parameters is. Instead, they only produce the values that were identified in models that perform well under some conditions. In the literature, we have found an approach [Hutter et al.,], which tackles the problem of selecting the most influential parameters of an algorithm. Besides, the number of parameters is large, just like in the case of ANNs.

Despite [Hutter et al.,] work is not applied to ANNs and their parameters, the underlying idea in this work is extremely related to the idea on which this work is based. In [Hutter et al.,], they are dealing with algorithms that have a set of parameters that the user can customize. Just like in ANNs, these parameters originate a large combinatorial space, which is hard for humans to handle. It is also hard to identify good configurations and to understand the impact that each parameter has on the algorithm. This work is interested in identifying which parameters the analyzed algorithms are more sensitive to. Forward selection is used to identify the most important parameters for the algorithms. Forward selection adds one variable (parameter) to the model at a time, and assures that only the most significant variables are added to the model.

Our work is extremely related to the latter work because we are also developing a strategy to identify which parameters are the most influential to an algorithm in our case, the parameters of an ANN's training algorithm. Despite of the similarities, our work differs from theirs because they used Forward Selection to identify the parameters and we are using a regression method to perform the selection. Also, the present work is only focused in ANNs instead of reviewing more methods.

1.7. Thesis Outline

Chapter 2 covers a brief but complete introduction to ANNs. It explains the basic aspects of ANNs and discusses all the variables that make ANNs complex models. This number of variables is the reason why ANNs can have a large number of combinations on their structure. Each particular model of ANN exhibits different leaning behavior and therefore some models may deal better with some problems than others. These and more

considerations about ANNs will be covered in the chapter.

Chapter 3 introduces *Parameter Influence Determination (PID)*, a methodology developed to fulfill the objectives set in this work. This chapter explains this work's basic assumptions as well as the characteristics of the problem being solved. An algorithmic view of PID will be presented, which guarantees a better understanding of the steps needed to determine the influence of the training parameters.

After we have determined the influence of the training parameters, Chapter 4 models an ANN based only on the most influential training parameters. That is, algorithm predicts the expected performance of an ANN given the values of the training parameters. Being able to know *a prori* the expected performance of an ANN is specially useful in case we were in position to use other approaches that would perform better for a given problem.

The results obtained from both Chapters 3 and 4 will be presented in different sections in Chapter 5. A brief description of the settings used in the tests will be also covered in this chapter. Finally Chapter 6 presents the conclusions and future work.

Chapter 2

Artificial Neural Networks

The goal of this chapter is to describe the main characteristics of Artificial Neural Networks (ANNs) and the two training algorithms being analyzed in this contribution. The chapter is organized as follows: Section 2.1 presents the basic principles that constitute an ANN, Section 2.2 describes the basic computational unit of an ANN, Section 2.3 presents how an ANN is organized, Section 2.4 shows the activation functions used in ANNs, in Section 2.5 the traditional algorithm to train an ANN as well as other improved versions of it is reviewed, and finally Section 2.6 is the summary of the chapter.

2.1. Fundamentals about ANNs

There is no formal definition of what an Artificial Neural Network is, but a basic description about them is that they are mathematical models inspired in the biological structure of animals' brain cells. Artificial Neural Networks (ANNs) were developed based on the understanding of how the central nervous system manages the information. They have been used to model the relationship between input and output data. An ANN is called a network because it connects simple processing elements together. In this way, the processing elements exhibit a more complex behavior which mimics the way the information is processed in animals' brain. The processing element acts like a real Neuron in the nervous system. Although the complexity of real neurons is highly abstracted when modelling an ANN.

ANNs have attracted interest due to their learning capabilities. This means that they can learn how to solve a particular task. Moreover, if an ANN can solve a particular

task or problem, it can generalize to similar problems and find their solution in some optimal sense. Before an ANN can be used to solve any problem, it has to be trained. ANNs are trained by a *Training algorithm*, which acts like a teacher for the ANN. It presents input data and evaluates the output of the ANN. With this information, the algorithm adjusts the connections between processing elements leading the ANN to learn the desired output for the inputs presented to it. These connections are initialized with small random values. That is why ANNs are not deterministic methods.

As universal approximators, ANNs represent a good approach to find complex relationships between some input and output data, i.e., non-linear relationships. This is why ANNs are good for modelling complex systems, and a pausible alternative to conventional techniques, which are generally restricted by normality, linearity, and variable independence, just to mention a few.

2.2. The Neuron

An ANN is constructed by connecting a number of processing elements called *neurons*. The neurons are connected in such a way that they emulate the inner structure of the animals' brain cells. In the biological structure the neurons are connected by *neuron synapses* which carry information through the neurons. In an ANN, the synapses are simply called *neuron connections*. A neuron receives an input (or inputs if it has more than one connection), processes it, and produces an output. Every neuron connection has a value or *weight* that interacts with the associated input, the way that a neuron processes these inputs is given by Equation (2.1),

$$S = \sum_{i}^{N} w_i x_i \tag{2.1}$$

where x_i is the i-th input and w_i is the weight value of that connection. Then the neuron computes the sum of products between all the inputs and weights, and this value is passed to a function $f(S)$ called *activation function* (Equation (2.2)).

$$k = f(S) \tag{2.2}$$

The result of $f(S)$ Equation (2.2) is finally the output of the neuron. Activation functions are described in Section 2.4. Figure 2.1 shows the model for a neuron.

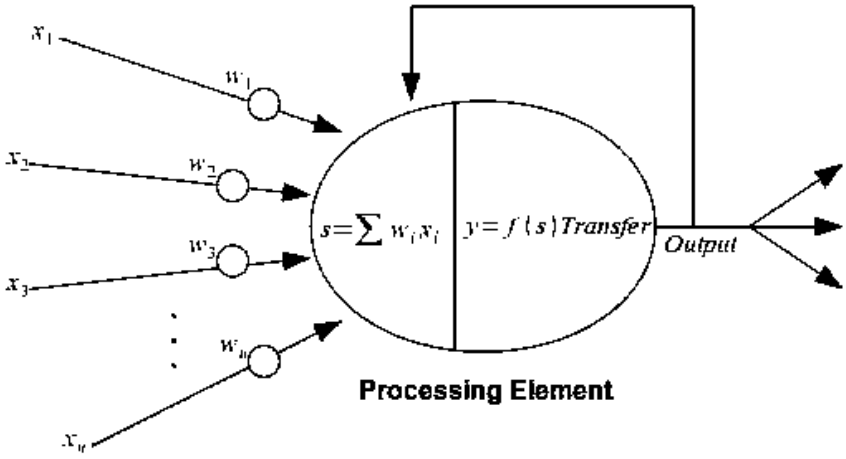

Figure 2.1: Neuron Model

2.3. Structure of an ANN

One neuron by itself cannot do very much (just like an actual brain neuron) when it comes to learn a particular task; however, the computational power of the ANN's does not lie in the capabilities of one element but in the capabilities of a number of interconnected neurons. The neurons in an ANN are arranged in *layers*. That is, two neurons that are in the same layer receive the same inputs. This does not imply that both neurons will return the same output because the connection weights of each neuron are different and they can have different activation functions. One of the most traditional models of an ANN is the *3-layer feed forward ANN*, where the neurons are organized in three layers. This model can be seen in Figure 2.2.

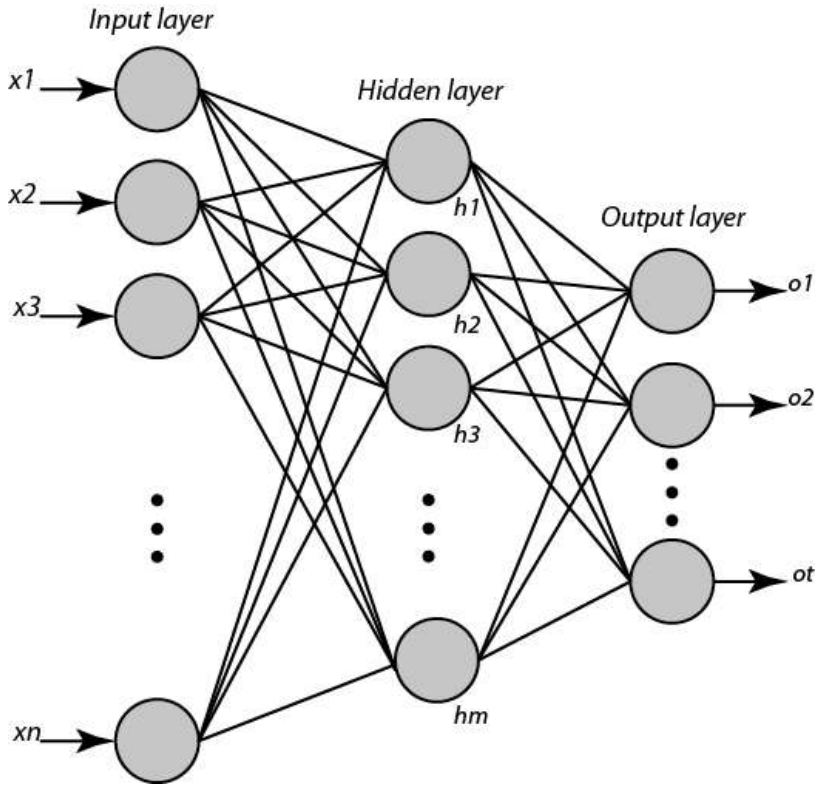

Figure 2.2: Three layer ANN

These three layers are organized as follows: the layer of neurons that directly receives the inputs of the problem is called the *input layer*, the layer that returns the total output of the ANN is called the *output layer*, and the layer that performs the inner calculations is called the *hidden layer*.

The number of neurons placed in the input layer and in the output layer are defined by the problem being solved. For example, in classification problems the number of input neurons is defined by the number of *features* and the number of output neurons is related to the number of *classes*. on the other hand, the number of hidden neurons does not depend on the problem being solved, but on the experience of the user.

If all this process of *tuning* the structure of an ANN was not hard enough, we still have the problem of choosing the number of hidden layers. As we add hidden layers we are adding complexity to the model. Certainly, larger ANNs with several hidden layers could adequate better to some problems that ANNs with less hidden layers. However, ANNs with more hidden layers will require more training time because they have more connections

and this is computationally more expensive. If computational time or cost are not considered in comparing larger ANNs against smaller ANNs, due to the fact that the *learning algorithm* used to train an ANN often finds a local minimum rather than a global one, the solution found by the ANN with more hidden layers is not necessarily better than the solution found by the smaller ANN. It has been proven that an ANN can approximate any continuous function (at some arbitrary precision level), this is demonstrated by the *Cybenko theorem* [Cybenko, 1989]. This is true for any ANN that contains one or more hidden layers. Therefore, an ANN with just one single layer has the same learning capabilities as ANNs with more hidden layers. According to the *parsimony principle*, is always better to choose a simpler model which can achieve good results instead than a complex model with the same performance. This might be the reason why the three layer ANN is widely used.

2.4. Activation functions

All the neurons have an activation function that takes the sum of products between inputs and connection weights as argument. The activation function is responsible for the output of a given neuron. The output is often in the range [0,1] or [-1,1]. Traditionally, if an activation function has a range of [-1,1] is called *symmetric* activation function. There are different types of activation functions, some of them will adequate better to some problems than others. The most utilized activation function is the *Sigmoid* function. However, in this work we are using the *Elliot* function. In this work this particular function was chosen because, in our experiments, it proved to work well with time series data. Other activation functions that are widely used are: Gaussian, Linear, Sine, and Cosine.

Sigmoid function

The sigmoid function is defined as $f(S) = \frac{1}{1+\exp^{-sS}}$ and the symmetric sigmoid as $f(S) = \frac{2}{1+\exp^{-sS}} - 1$, where S is the input and s is a real positive value called *steepness* of the activation function, which is discussed in Subsection 2.4.1. These functions are shown in Figures 2.3 and 2.4.

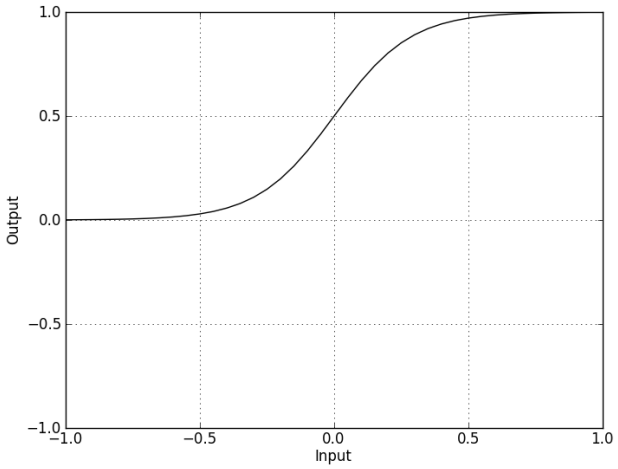

Figure 2.3: Sigmoid function

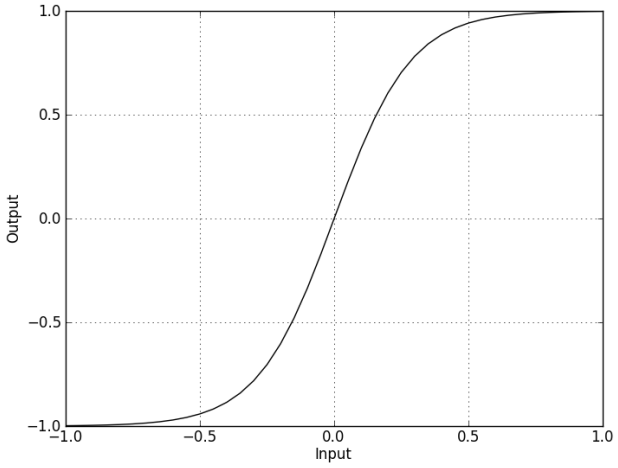

Figure 2.4: Symmetric Sigmoid function

Elliot function

The Elliot function resembles the sigmoid and is defined by $f(S) = \frac{Ss}{2(1+|Ss|)} + 0.5$, and the symmetric Elliot as $f(S) = \frac{Ss}{2(1+|Ss|)}$. Both functions are depicted on Figures 2.5 and 2.6.

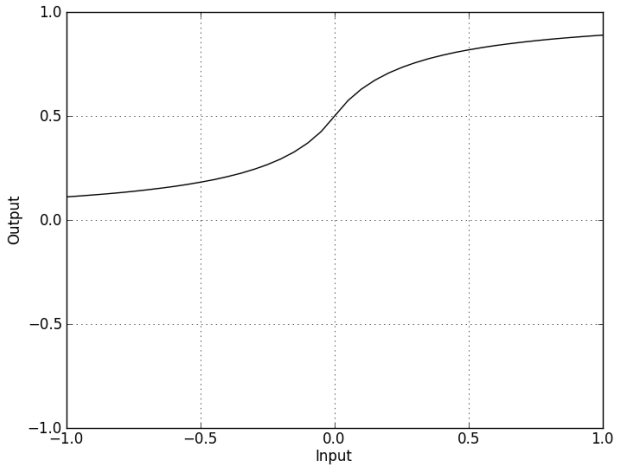

Figure 2.5: Elliot function

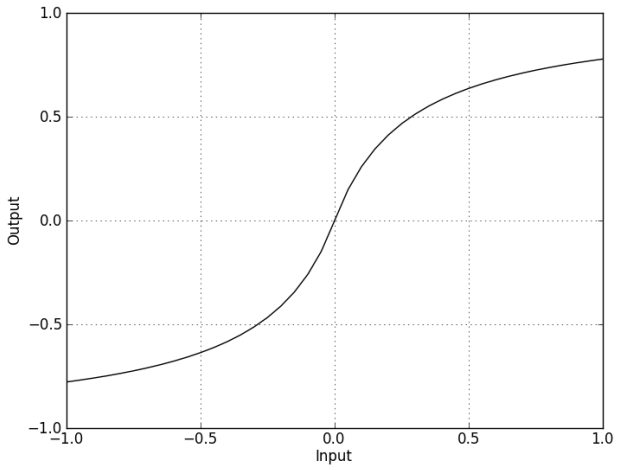

Figure 2.6: Symmetric Elliot function

2.4.1. Steepness of an activation function

All the activation functions mentioned above have a parameter called *steepness*, which acts like a gain in the slope of the function. Lower steepness will reduce the slope of the function, i.e., turning it flatter, while higher values will increase the slope, turning it sharper. The effects of the steepness in the sigmoid function are shown in Figure 2.7

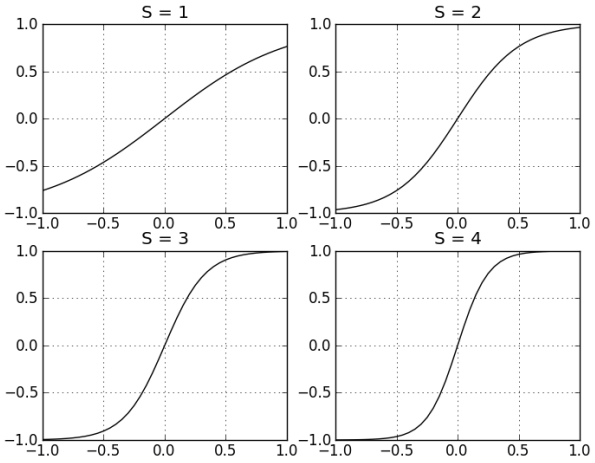

Figure 2.7: Different steepness values for the sigmoid function

2.5. Training an ANN

Before an ANN can be used, it has to be trained. The process of training an ANN is commonly referred as the *training phase*. The ANN's weights are initialized with small random values typically in the range [-0.01,0.01]. The first pattern in the training data is presented to the ANN and the outputs of the input layer is calculated. The resulting values are passed as input to the next layer, and so on. Eventually, the corresponding values are input to the output layer, which performs the last calculation to obtain the output of the whole ANN. Once the output of the ANN is obtained, it is compared to the desired output of that pattern.

The goal is to minimize the difference between the desired output and the actual output for every pattern presented to the ANN. The process when the ANN compares both outputs and then adjusts the weights to minimize the difference is the mentioned *training phase* and the algorithm used to adjust the weights is called the *training algorithm*.

2.5.1. Back propagation Training Algorithm

The Back propagation algorithm was first utilized in training ANNs in 1974 in the book [Werbos, 1974]. The learning process is called back propagation because it propagates the error of the output layer backwards through all the ANN. The Back propagation

algorithm consists of three steps:

1.- Forward Propagation.

2.- Back Propagation.

3.- Weight updates.

Forward propagation means that we present a training input for the ANN and we calculate the outputs of all the neurons, layer by layer until we reach the output layer. At this point, when the forward propagation finishes, we know the output of the ANN. In the back propagation stage we compare the output against the real or desired output, this difference is called the ANN error. This error is used to determine the changes in weight connections of the ANN, first in the output layer, then in the lower layers. Back propagation calculates the first derivative of the error with respect to the weights of the ANN and then, this information is used to update the connection weights. This process starts all over again for the next pattern in the training set. Finally, the algorithm finishes when a stop criteria has been met. Back propagation needs a continuous activation function because it computes the derivative of the error to compute the magnitude of the change in the connection weights.

There are two ways of training an ANN using back propagation: *batch* learning and *incremental* learning. In incremental learning every time we feed the ANN with an input pattern a weight update occurs. In batch learning, we present all the input patterns to the ANN and then the weight changes occur. This form of back propagation needs more memory to store all the weight changes calculated, but in incremental learning more weight changes are produced.

Due to the success of back propagation in training ANNs, a number of algorithms based on it have been developed. This thesis focuses on two of these kind of algorithms: QUICKPROP and RPROP, which are improved versions of the back propagation algorithm. In the next subsections (Subsection 2.5.2 and 2.5.3) these two algorithms are explained.

2.5.2. QUICKPROP Training Algorithm

QUICKPROP stands for *Quick propagation*. It was born as the result of an intense research to understand the factors that govern the learning speed. It tries to change the

ANNs weights as fast as possible but without overshooting the solution. It was developed to overcome some limitations of the Back propagation: it has a slow convergence and it does not scale well on complex problems [Fahlman, 1988].

QUICKPROP can be seen somewhat like a second-order method, due to the fact that the algorithm keeps a copy of the derivative error computed on the previous epoch for every weight. This copy is used to give more information than the one given only by the first derivatives and safely take larger steps in weight space. QUICKPROP is loosely based on Newton's method but it is still more heuristic than formal [Fahlman, 1988]. A simple explanation of what the algorithm does is that the algorithm changes the weights based on what happened during the previous weight updates. For more information regarding QUICKPROP see [Fahlman, 1988].

QUICKPROP has three parameters which are:

1. *Learning Factor (LF)* The learning factor is used to determine how large the step-size should be in the training stage.

2. *Decay Factor (D)* The decay is a small negative valued number which is the factor that the weights should become smaller in each iteration during QUICKPROP training. This is used to make sure that the weights do not become too high during training.

3. μ The mu factor is used to increase and decrease the step-size during QUICKPROP training.

2.5.3. RPROP Training Algorithm

RPROP stands for *Resilient Back Propagation* and was introduced by [Riedmiller and Braun, 1993]. When Back propagation is used, the magnitude of the derivative determines the size of the weights. As a result, neurons far from the output layer learn slower than neurons near of the output layer. This is because the magnitude of the derivative decreases exponentially due to the limiting influence of the slope in the activation function [Riedmiller and Braun, 1993].

RPROP tackles this issue by using only the *sign* of the derivative instead of the magnitude. That way all the weights are updated with values that depend only on the sequence of signs and the information is spread equally all over the entire ANN [Riedmiller

and Braun, 1993]. Because all the layers of the ANN are affected equally with the error information, RPROP provides a faster convergence.

The RPROP algorithm has five parameters:

1. *Decrease Factor (Dec)* The decrease factor is a value, used to decrease the step-size during RPROP training.
2. *Increase Factor (Inc)* The increase factor is a value, used to increase the step-size during RPROP training.
3. *Delta min (Dmin)* A small positive number determining how small the minimum step-size may be.
4. *Delta max (Dmax)* A positive number determining how large the maximum step-size may be.
5. *Delta zero (Dzero)* The initial step-size is a positive number determining the initial step size.

In order to summarize, the parameters of QUICKPROP and RPROP learning algorithms are shown in Table 2.1.

Table 2.1: QUICKPROP and RPROP parameters

QUICKPROP	**RPROP**
Learning factor	Increase factor
Decay factor	Decrease factor
Mu	Delta min
	Delta max
	Delta zero

2.6. Summary

In this chapter the fundaments of Artificial Neural Networks had been presented. ANNs are learning models inspired by nature, specifically in how the brain cells learn. ANNs are good for classification and prediction problems. Under special circumstances, they outperform other approaches for prediction like ARIMA models, but many considerations have to be made in order to design an ANN that actually beats these other approaches. Such considerations include the size of the ANN (number of hidden layers), number of neurons

per layer, activation functions, and the training algorithm. When the design of the ANN is complete, the difficulty of tuning the training parameters remains. This is the problem where this work is aimed to. This kind of problems relative to ANNs have discouraged potential users because there are no guides or formulas that the regular user can use to guide the ANN design. This is the reason why ANNs have been used in real life only by experts in the field.

Chapter 3

Parameter Influence Determination PID

Now that the general parameters of an ANN have been presented, it is important to note that changes in these parameters would lead to entirely different ANN models with different performance. This leads to the algorithm selection problem, first presented in [Rice, 1975], which consists of choosing the best and simpler approach between a collection of algorithms for a given task. This is the same problem with ANNs, the difficulty to know *a prior* which ANN model to use considering only the changes in the training parameters.

Unfortunately, the algorithm selection problem is not simple to solve. One of the objectives of this work is to determine the influence that each training algorithm parameter has over the behavior of the ANN. This objective is related to the algorithm selection problem because by understanding the role of each parameter, we are obtaining valuable information that can be of help to solve this problem.

This chapter presents a methodology for determining the influence of the parameters in an ANNs training algorithm, specifically for QUICKPROP and RPROP training algorithms. This methodology is called *Parameter Influence Determination* or just *PID*. This chapter is organized as follows: Section 3.1 presents the motivations for which PID was developed, Section 3.2 reviews the basic aspects of PID and how the problem of the determination of the parameter influence can be solved using regression. Finally, in Section 3.3 we discuss the lessons learned with PID.

3.1. Introduction

ANNs are powerful tools mainly used in classification and prediction problems, but there are several things to keep in mind when it comes to the selection of a given ANN model (number of layers, number of neurons per layer, selection of the activation functions etc.). Moreover, even when the task of designing the ANN architecture is complete, the problem of choosing which training algorithm is going to be used remains. This training algorithm could be the classical back propagation, or more advanced algorithms like *RPROP* and *QUICKPROP*.

If the back propagation algorithm is chosen, the only parameter to *tune* is called *learning factor*, which is the value of the step size taken by the algorithm in order to accelerate the convergence. Things get a lot more complicated when the RPROP and QUICKPROP algorithms are chosen, because of the number of parameters each algorithm has. The reason for choosing these algorithms is because they represent advanced or improved versions of the back propagation algorithm and with a careful tuning of their parameters they outperform back propagation, in some sense. In particular, they are faster, or find better solutions, or both.

The task of tuning or selecting the parameters values is not trivial, mostly because there is no guide on which we could base our decisions, beyond our own previous experience. Even when *good* values for the parameters of a given algorithm have been found, these values do not guarantee that they will perform well at all in another problem or another task. For example, good values for prediction problems could be not good for classification problems (the same thing occurs with the activation functions, some of them are recommend for classification and some for prediction problems). Therefore, this is why this work is focused only on ANNs used as predictors and specially on the problem of time series forecasting.

The ANN model used in this work corresponds to a three-layer feed-forward ANN, which is the most common ANN architecture. Another consideration that has been made is relative to the activation functions. Different activation functions are not being tested, instead, the *Elliot* activation function is used in all the experiments. This activation function gave good results in preliminary experiments.

An ANN training algorithm (QUICKPROP for example) uses a set of parameters that will determine the way the weights between the layers of the ANN are adjusted (which is an optimization problem itself). If these values are chosen in a proper way, the training

will adjust the weights successfully. This means that the error between the desired output and the real output of the ANN satisfied the stop criteria and now the ANN can be used to forecast the time series. This is why it is extremely important to set the parameters to the appropriate values. This work analyzes the influence that each parameter has over a given training algorithm. In other words, how each parameter affects the training and the performance of the ANN in order to pay special attention on those parameters that determine the performance of the ANN.

3.2. Methodology

PID is an algorithm that sorts the training parameters depending on their influence at the performance. A parameter p_1 is said to be more influential than another parameter p_2 if the training algorithm is more sensible to it. This means that small changes in the parameter p_1 will produce very different behaviors in the training algorithm. Such different behaviors in the training phase will eventually lead to different trained ANNs, with different performances. So, if the value of an important parameter of the chosen training algorithm is not chosen carefully, this will be translated in ANNs that are not well trained and will be useless for the intended purpose.

In order for PID to be aware of what parameters are more sensible to the training algorithm, it has to vary the values of the parameters involved. This variation of values in the parameters must be done in a systematic manner. If the steps taken between one value of the parameter and a different value are very small, the outcome will hardly vary quantitatively. On the other hand, if such steps are taken very large, behaviors between values are not being explored. Therefore, the range of values in which a parameter will vary has to be selected wisely. Another restriction to the steps between values is that if they are very small, there will be a lot of combinations parameter's for the values to proof and this is definitelly computational expensive.

This last consideration takes enormous importance in the time utilized for PID to determine the order of influence. Let us say that the training algorithm being used by PID has 5 parameters (e.g. QUICKPROP). Every single parameter is going to be varied by PID. If PID has 4 possible values for each parameter, this means that there are 4^5 combinations of parameters that PID has to review. Each of these 4^5 combination of parameters represents a different behavior for the training algorithm. Training an ANN is relatively cheap with

modern computers, so for 5 parameters with 4 possible values, PID will have to train $4^5 = 1024$ different networks. This 1024 ANNs does not seem to be much for nowadays computers, but if we raise the possible values for example to 10, the number of ANNs that have to be trained raises to $10^5 = 100,000$.

This means that the number of ANNs to be trained grows exponentially when the number of possible values for each parameter increases. This is $O(n^m)$ where n is the maximum number of possible values for a parameter and m is the number of parameters that the algorithm has. Figure 3.1 shows how this curve looks like in the case of RPROP training algorithm, which has 5 parameters.

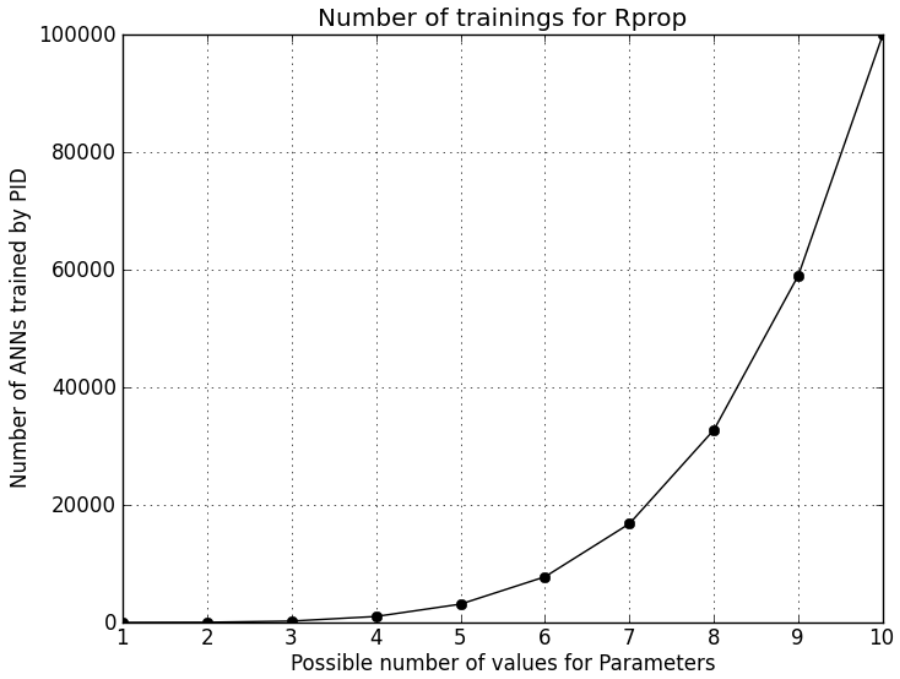

Figure 3.1: Exponential growth of number of ANNs to be trained.

PID will train an ANN with a combination of parameters to obtain a measure of quality. The measure of quality used in this work corresponds to the *Mean Squared Error* or *MSE*. In Statistics, this measure is used to quantify the difference between real and predicted measurements. This measure is presented in Equation (3.1) where Y is the real value and $\hat{Y}$ is the prediction.

$$MSE = \sum_{i}^{N} (Y - \hat{Y})^2 \tag{3.1}$$

Lower MSE scores means better predictions of the ANN. This score is used to distinguish how a small change in the parameters of the algorithm translates to different behaviors in performance. It is important to note that higher MSE scores are not being discarded (high MSE score indicate poor performance), all the MSE scores no matter if they are high or low are included. MSE scores that are related to poor performance could be discarded, however, including these scores will give much more information about the influence of the parameters in the ANN's behavior.

In practice, ANNs are initialized with small random values for all its connection weights. Contrarily, PID always trains an ANN with the same initial weights, the only thing that PID varies are the training parameters. This is done this way because different initial weights will produce different behaviors in the ANN even with the same values for the training parameters, and we would require to perform several runs to measure the performance.

To sum up, PID trains an ANN with the same initial weights in its connections, same architecture (3-layer ANN) but with different combinations for the training parameters. This will produce a different ANN once the training has been completed. Then, this ANN is used for time series prediction, and its behavior is measured with the MSE score. PID establishes a relationship between the MSE score and the combination of parameters seeing the problem as a *Regression* problem, this will be explained in the next section.

3.2.1. Parameter influence as a regression problem

The Regression problem is very simple, it consists of finding the relationship between one or more variables called *explanatory variables* and a *response variable*. Although this seems quite trivial, it is not. Most of the time, the relationship between the explanatory variables and response variable is not easy to find or easy to model accurately. The importance of solving the regression problem is that if the relationship between the variables is known, this will lead to a better understanding of the systems that produce the data being analyzed.

One of the main objectives of this work is to determine the relationship between the parameters of an ANN's training algorithm and the performance of the ANN. It was

possible to achieve this goal using regression techniques. The training parameters are being used as the explanatory variables and the error of the ANN after the training as the response variable.

Linear Regression

There are several ways to solve the regression problem, a widely used approach is *Linear Regression*. Linear Regression assumes that the relationship between explanatory variables and the response variable can be explained with a model whose coefficients are linear. There are two cases for linear regression: if there is only one explanatory variable it is called *simple regression* and for more than one explanatory variable it is called *multiple regression*. An example of simple regression modelling is illustrated in Figure 3.2, where there is one dependent variable and one explanatory variable. The points are the data that is going to be modelled and the line that passes through the points is the model estimated by linear regression approach.

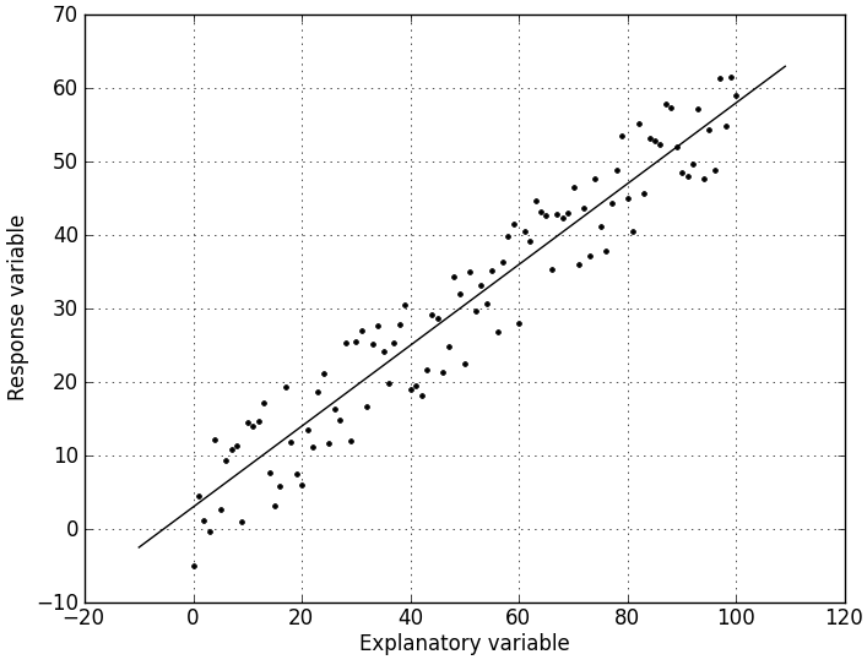

Figure 3.2: Simple Linear Regression

The model for multiple regression has the form $y = M\beta + \epsilon$, where $M \in \mathbb{R}^{n \times m}$

are the m explanatory variables with n observations, $y \in \mathbb{R}^n$ is the response variable with n observations, $\beta \in \mathbb{R}^m$ are the linear parameters estimated and $\epsilon \in \mathbb{R}^n$ are the errors associated with each observation.

Linear regression focuses on finding the parameters β that model the data. There are several techniques or approaches for doing this: Ordinary least squares (OLS), Generalized least squares (GLS) [Kariya and Kurata, 2004], Bayesian linear regression [Minka, 2000], Least-angle regression (LARS) [Efron, 2004], and Regression Shrinkage and Selection (LASSO) [Tibshirani, 1994], among others. In this work, PID uses *Random Forests* to solve the problem of regression. Random forest will be covered in Section 3.2.3.

3.2.2. Decision trees for Classification and Regression problems

Decision trees [Breiman, 1993], are predictive models which are used in several areas like machine learning, statistics and data mining. A decision tree splits the training data according to the data features, so, when a new observation is presented the decision tree fits this new observation into one of its leafs. Decision trees can be used for both classification and regression problems. These methods are often called *Classification and Regression Trees (CART).*

In the case of classification, the tree is called a *Classification tree.* In classification, when the new observation is assigned to a leaf, it will be labeled like the majority of samples in the training set that are also in the same leaf. If a decision tree is used in regression, it is named a *Regressor tree.* In Regression, when the tree has decided what leaf a new observation belongs to, this new sample will take the average value of all the samples that are in that leaf.

To illustrate how this works, let us assume some training data for a classification problem (because it is simpler to illustrate), like the data presented in Figure 3.3. In this example, a collection of objects which belongs to one of two classes (triangles or stars) is presented. Each of the objects has two features, f_1 and f_2.

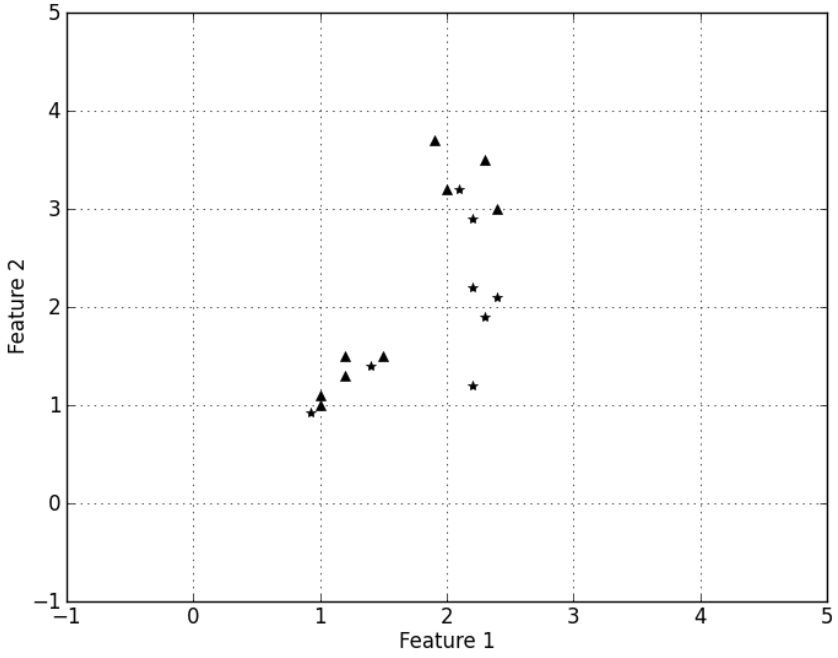

Figure 3.3: Training Data for a classification problem.

A classification tree will split the training data according to its features. In other words, the tree will try to isolate subsets of the training data, whose features values are close together. These splits continue until some criteria is reached, i.e., when the subset generated by the best split contains less than or equal to a fixed number of elements. Two possible splits of the data are shown in Figure 3.4. Although that more than two splits can be considered, only two splits are shown in order to keep the example simple. In the first of these two splits, two subsets are generated but one of them has more elements than the theoretical stopping criteria, so a second split is needed to separate this subset.

When the tree has done all the splits on the data, the data is now represented in a decision tree. The classification tree generated for the splits shown in Figure 3.4 is presented in Figure 3.5. The leafs in a classification tree, tell us how many elements of each of the classes are in that leaf. When a new observation is presented to the tree, it will follow a path trough the tree according to its features values. Eventually, this observation will reach a leaf and it will be classified into the class that has more members in that leaf.

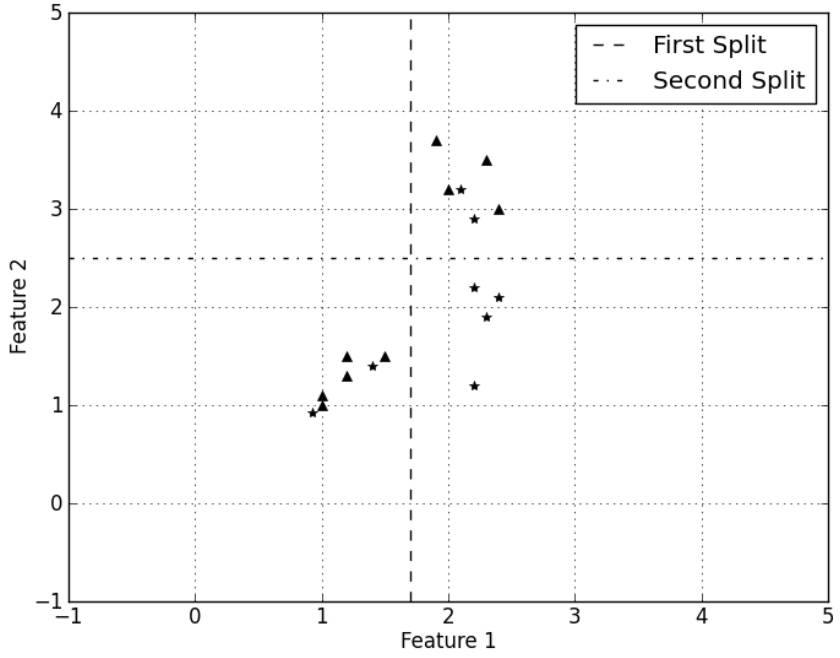

Figure 3.4: Two splits for the training data.

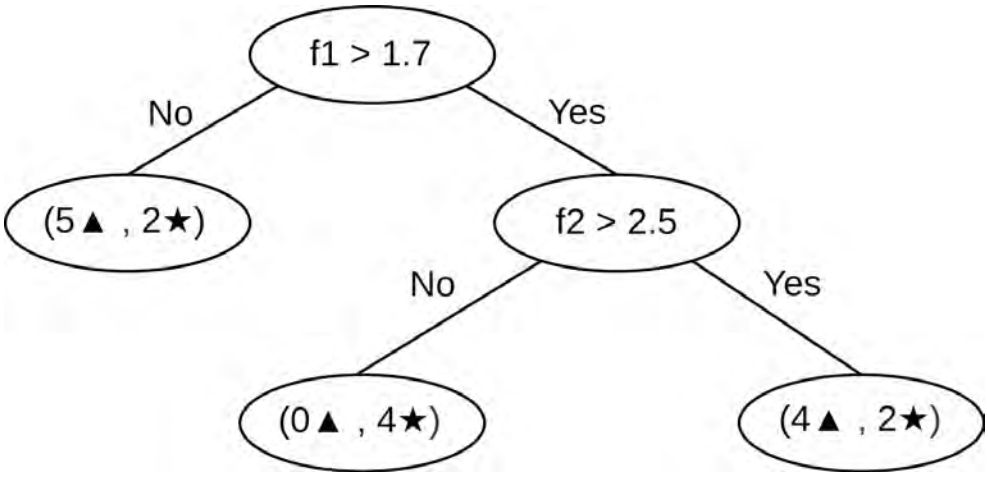

Figure 3.5: Classification tree generated from the splits.

In the last example, a new sample whose feature one is not greater than 1.7, would be labeled as a triangle, because the leaf that will receive the sample has 5 triangles and 2

stars on it. In other case, where the new sample that is presented to the tree has a value of feature one greater than 1.7 but the value of feature two is less than 2.5, it will be labeled as a star, because the leaf has zero triangles and four stars. It is desirable that the number of elements in all the leafs will be odd in order to eliminate draws. In conclusion, a classification tree is an extremely simple structure which arranges the data according to their features or characteristics.

In the case of regression, it is even simpler to fit a new observation to the tree. The tree performs just like in the case of classification, splitting the data and ordering it in the tree. In regression the tree has to predict the outcome of a sample given its features, so there are no classes to decide where the sample belongs. The difference in regression is that, when a sample reaches a leaf, the predicted value of that new sample is calculated as the average of all the training samples that are contained in that leaf.

3.2.3. Random Forests

Random forests can be seen as a generalization of Decision trees. Random forests were introduced by [Breiman, 2001]. They are an ensemble of tree predictors, where the generalization error depends on the strength of individual trees in the forest and the correlation between them [Breiman, 2001]. Random forests are competitive methods against other approaches in supervising learning like Support Vector Machines (SVMs), neural nets, logistic regression, naive bayes, memory based learning, decision trees, bagged trees and, boosted trees [Caruana and Niculescu-Mizil, 2006]. The algorithm for Random Forests is presented in Algorithm 1.

Algorithm 1 Random Forest Algorithm

1: Let N be the training samples $[(X_1, y_1),...,(X_n, y_n)]$.
2: Let T be the number of trees in the forest.
3: Let M be the number of features of the samples i.e, X_i is M-dimensional.
4: **for** $i = 1, ..., T$ **do**
5: Choose bootstrap sample N_i from N.
6: Construct T_i tree using N_i such that at each node only a random subset of m features is considered to split the data.
7: **end for**

The Random Forests algorithm constructs multiples trees over the same data, and

its name is due to the fact that it randomly chooses the subset of the model features, done in step 6. Random forest are based on Decision trees, which are extremely simple models and combined with randomized techniques, they become methods that achieve outstanding results. Random Forest have several desired qualities: they are robust to outliers and noise, give useful internal information like error, strength, correlation and, variable importance, and are easily parallelized [Breiman, 2001]. These last two characteristics are the reasons why we focused our attention to Random Forests. Because it is easily parallelized, PID can train a larger number of ANNs in the same amount of time. However, the variable importance computed by Random forest is the main advantage over other approaches for performing the regression.

Variable Importance

This work uses the sklearn implementation of a Random Forest Regressor [Pedregosa et al., 2011]. The variable importance computed by Random forest gives information about which explanatory variables are influencing the response variable by a greater degree. This means that every Random Forests are used to model the relationship between the training parameters and the MSE measure obtained during training, a model that best fits the system and a list of variables sorted by its importance in the outcome are obtained.

Before explaining how Random forests computes the variable importance of the regression problem, the concept of *out-of-bag error* needs to be introduced. When using Random forests the need of using a cross validation or another test to get an unbiased estimate of the error disappears. This error is estimated internally during the run and it is called *out of bag error* or simply *oob error*. As a result each tree is constructed using a bootstrap sample of the original data, one third of these samples are not used to construct the k_{th} tree. When the trees are used, a bootstrap sample that has not been seen by the tree occurs in one third of the all trees. At the end of the run, a class c has been elected as the classification output when it was an oob error case. The proportion of times that this class c is not the real class is the oob error estimate. This has proven to be unbiased in many tests [Breiman, 2001].

This is how Random forests compute variable importance: for every tree grown in the forest, random forests algorithm analyses the oob error cases and counts the number of votes for the correct class. Then, the algorithm randomly permute the values of variable

m in the oob cases and proves it in the tree. The number of votes for the correct class in the variable permuted oob data is subtracted from the number for the correct class in the non-oob data. The average of this number over all the trees is the raw importance score for the variable m.

3.2.4. PID algorithm

So far, the basic aspects of PID have been described. The considerations taken by PID such as the approach used to perform the regression and reasons why we used it have also been covered. Now that the basic idea of what PID does has been explained, it is easier to refer to PID as an algorithm. In this subsection, the steps taken by PID in order to achieve the proposed goals will be explained.

The first thing to do is choosing the training algorithm for which the parameter influence is going to be computed. Then, PID splits the parameters ranges uniformly. Once this is done, a set with different parameters combinations is created. PID tunes the training algorithm with one combination of parameters and trains the ANN for over 3000 epochs. This number may be seem too large, but in this way, we assure that the algorithm has converged.

After the training has been completed, an MSE measure is obtained for the training and validation sets. The combination of parameters used is added as a row of the matrix that is going to be used by Random Forests. The measures obtained in the validation and training sets are added to the vectors y_T and y_V respectively. This process continues until there are no more untested combinations of parameters. At the end of this process PID has the matrix and vectors that are needed for the regression.

PID uses Random Forest to solve the regression in both training and validation sets. PID tests these two sets because it is also important to know if the training parameters affect in different way the training and validation phases. An ANN can be very good in the training set (present low MSE error), and perform poorly in the validation set (high MSE error). If this is the case, the ANN is useless because it can not generalize well. So, if a particular parameter affects the validation results in some deeper way, it would be of great help to identify it. Therefore, the chance of choosing properly, the values for the training parameters increases, leading to an ANN that will have low MSE error in the validation set. An algorithmic representation of PID is presented in Algorithm 2.

Algorithm 2 PID Algorithm

1: Create a set Θ with all the parameters combinations considered.
2: Create empty matrix $M = []$ and empty lists $y_T = []$, $y_V = []$
3: **for** θ in Θ **do**
4: Create empty lists $\Gamma_T = []$, $\Gamma_V = []$
5: **for** p in P **do**
6: Create an ANN with training parameters θ
7: $\gamma_T = \Phi(\theta, p)$
8: $\gamma_V = \Phi(\theta, p)$
9: Add γ_T to Γ_T
10: Add γ_V to Γ_V
11: **end for**
12: Add average of Γ_T to y_T
13: Add average of Γ_V to y_V
14: Add θ as a row of M
15: **end for**
16: Create RadomForest f_T and f_V
17: Solve the regression for M and y_T with random forest f_T
18: Solve the regression for M and y_V with random forest f_V
19: Compute variable importances for f_T and f_V

In line 1 PID creates a set Θ which contains the parameters values combinations, one element of this set is one combination of values for the training parameters being analyzed. Step 2 creates the matrix M where the explanatory variables (i.e. the parameters), vectors y_T and y_V are stored. Steps 6, 7 and 8 create an ANN and train it with one combination of parameter values i.e., θ created in Step 1. Steps 9 and 10 add the performance in the training γ_V and validation γ_T stages to the lists Γ_T and Γ_V. Steps 12 and 13 add the average performance of the ANN with θ training parameters to y_T and y_V. Step 14 add the training parameters as a row of M. Step 16 creates two random forests which will be used for the training and the validation. Steps 17 and 18 solve the regression problem for training and validation. Finally, Step 19 computes the variable importance for both random forests.

It is important to mention that Algorithm 2 is applied to 1001 time series, this

means that the i_{th} element of vectors y_T and y_V is given by equation (3.2). So, every entry of the response vectors are the average performance of the ANN with a fixed combination of values for the training parameters (Θ_i), for 1001 time series. On the other hand, the matrix of explanatory variables is built with all the combinations of parameter values, this matrix has the form given in Equation (3.3), where the sub index indicates the combination of values being used for training. The set Θ contains m elements, so Θ_m is the last combination of training parameters in the set. The super index refers to one parameter in particular, in the case of Equation (3.3) there are 6 parameters (the case of RPROP), for example θ_i^3 refers to the third parameter of the i_{th} element of Θ. Random forests use the matrix M and the response vector y_i to perform the regression.

At the end of the algorithm two lists are presented, containing the parameters influence order, one for the training set and another for the validation set. Such lists, give information about what parameters influenced more the training phase and the validation phase over a collection of time series. This means that, given a set of time series, after running the algorithm, it can be known which parameters is the training algorithm more sensible to.

$$y_i = \frac{1}{1001} \sum_{j=1}^{1001} \Phi(\theta_i, P_j) \tag{3.2}$$

$$M = \begin{pmatrix} \theta_1^{(1)} & \theta_1^{(2)} & \theta_1^{(3)} & \theta_1^{(4)} & \theta_1^{(5)} & \theta_1^{(6)} \\ \theta_2^{(1)} & \theta_2^{(2)} & \theta_2^{(3)} & \theta_2^{(4)} & \theta_2^{(5)} & \theta_2^{(6)} \\ . & . & & & & . \\ . & & . & & & . \\ . & & & . & & . \\ \theta_m^{(1)} & \theta_m^{(2)} & \theta_m^{(3)} & \theta_m^{(4)} & \theta_m^{(5)} & \theta_m^{(6)} \end{pmatrix} \tag{3.3}$$

3.3. Summary

A methodology for obtaining the parameter influence on the QUICKPROP and RPROP training algorithms for ANNs in time series forecasting has been developed. The relationship between the values of the training parameters and the behavior of the ANN have been solved by turning the problem into a Regression problem. Random Forests have been used to solve the regression problem, and based on them, the parameter influence

determination can be determined.

This work can be extended performing the same steps with other training algorithms. For example, the *SaRPROP* learning algorithm, which is a relatively new algorithm and has proved to be very fast (for more information see [**?**]). Also, in order to have more robust results a larger number of time series can be tested.

Chapter 4

Modelling an ANN for time series forecasting

This chapter describes the procedure used to model an ANN and is organized as follows: Section 4.1 explains the reasons behind modelling ANNs, Section 4.2 presents all the aspects related to the modelling process as well as the steps followed to accomplish the task, Section 4.3 describes the time series characteristics used in the process of modelling, and finally, Section 4.4 summarizes the core idea of the chapter.

4.1. Introduction: Reasons to model an ANN

In this work, ANNs are used to predict time series instead of using other approaches like ARIMA models. With a *good* selection of the ANNs training parameters, the ANN can predict with more precision than other tools. Nevertheless, the reality is that even with the help of PID, the task of tuning the parameters is still far from being trivial. Due to this issue, some people prefer to use ARIMA models, which are indeed simpler than an ANN model, obtaining close predictions with the ones obtained by an ANN but without the overwhelming process of tuning the parameters of an ANN.

Considering the algorithm selection problem, it would be of great help to have a model of an ANN and with this model, the possibility to know *a priori* the performance of that ANN and then choose between this particular model or other ANN models. Assuming such ANN model, the algorithm selection problem is trivial, because it will only consists in choosing the ANN based on the expected error predicted by the model.

Obviously, the real performance of an ANN can not be known without training the ANN over the time series. One thing that can be done is obtain a modelled ANN that is a representation of the real ANN that will give information about what will be the performance of the ANN over a collection of problems. This modelled ANN has to be a simpler model than the real ANN. In other case, it would make no sense to obtain a more complex model that only gives an approximation of the real ANN. This modelled ANN has to consider both training and validation stages like the real ANN.

Having said this, it gains a lot of interest to model an ANN and have the possibility to know *a priori*, with some degree of confidence, the expected error of that particular ANN model. This is why modelling an ANN is important to the algorithm selection problem, instead of actually proving all the ANN models available (which is computationally expensive), ANNs have been modelled in such way that given a problem instance, the expected error of the ANN can be known *a priori*, with some degree of confidence.

In this thesis, ANNs are used to solve the time series prediction problem (covered in Section 4.3). So, the problem instances for the modelling are the time series that are going to be used for prediction.

4.2. Methodology

No matter what kind of system is trying to be modelled, it is far from being an easy task. If the modelling considers all possible variables, it could result in a larger system under analysis. Moreover, if the modelling does not take into account some important variables, the final result might not be a good representation of the original. For the case of modelling an ANN, if our approach considers all possible variables (training algorithm, number of hidden layers, etc.), this approach will be more complex that the actual ANN. On the other hand, if we are not carefully enough to include important variables, the final outcome will not model the actual ANN to the desired accuracy.

Given a problem or task and two or more ways to solve it, the simpler method is always the best choice. This is the main idea of the *parsimony principle*, which recommends to choose the simpler solution available. Among all the considerations taken to model an ANN, the idea of the parsimony principle is also included. Unfortunately, it is not easy to solve the *complexity vs. simplicity* dilemma. In order to overcome this, PID could be used; given the results of PID, a starting point for consider for modelling arise.

As we will see in Section 5.1, the most influential parameters for QUICKPROP are the learning factor and the hidden neurons, and for RPROP the increase factor, the decrease factor and the hidden neurons. In order to simplify the modelling of the ANN, two variables (parameters) for each training algorithm are going to be modelled. In the case of RPROP the increase factor and the number of hidden neurons, for QUICKPROP the learning factor and the number of hidden neurons. This will simplify the modelling and also ensure that the selected variables are very important for the performance of the ANN.

The aim of these considerations is to produce an ANN that represents the real ANN, and provides a good approximation of the performance of the ANN. In other words, a model simpler than the real ANN, yields the expected performance of the ANN predicting a given time series. It is important to mention that the ANN model is not going to predict the time series, it will only describe quantitatively the expected performance of the real ANN. That way, if the ANN model indicates that the expected performance would not satisfy some criteria, there is no need to prove the real ANN, which is computational expensive.

Like in the case of PID, this work proposes an algorithm to model an ANN. Also, Random Forests are going to be used to represent the relationship between the ANN to be modelled and their performance over a time series. One thing that always has to be in mind is that the modelled ANN must have the ability to generalize, that is, feed this model with an *unknown* time series and obtain good estimations of the performance of the actual ANN with those time series. This is important, because the modelled ANN is going to be created given the actual performance of the ANN over a set of time series, design this model from this data, and then use this model to obtain an estimation of the performance of the real ANN on unknown time series.

This is why the effort for modelling ANN for time series prediction has to be not over the time series itself but over the *characteristics* (these characteristics will be mentioned in Subsection 4.3.1) of those time series that would create the model. At this stage of modelling, the system used by Random Forest is going to be constructed differentlly. Now, the explanatory variables are the time series characteristics. These characteristics (33 in total), include both unmodified data (RAW) and trend and seasonality adjusted data (TSA). The dependent variable is the performance of the ANN with a particular combination of values chosen for its two parameters considered.

The idea is to use Random Forests to model the relationship between the matrix M_1, built with the time series characteristics and the response vector. Then, with this model

obtained by Random Forest, we present another matrix M_2, which represents an unknown time series and obtain an estimate of the performance of the real ANN in the training and validation sets.

The strategy here is to divide the collection of time series into two different groups, one to create the modelled ANN, and one to validate it. The first group has the same function that the *training set*, i.e., it is used by the model to *learn* the characteristics of the problem under analysis. The second block is used as validation set, to test if the model has the ability to generalize and assure that is not *overtrained*.

First a particular ANN model will be trained, with fixed values for the parameters chosen for the training algorithm. This training is done over the first group of time series, and two response vectors are obtained: the error in the training set and the error in the validation set. Now, with this information, feed the Random Forest and obtain a model. Then, present this model a different set of time series that were not used in the first stage and obtain a response vector. This vector is compared to the actual output of the ANN over that time series to know if the modelled ANN yields good approximations of the actual ANN performance. This steps are described in Algorithm 3.

Algorithm 3 modelling an ANN

1: Split the collection of time series into two groups T_1 and T_2.
2: Extract the characteristics of T_1 and create the M_1 matrix.
3: Extract the characteristics of T_2 and create the M_2 matrix.
4: Create an ANN *net* with training parameters θ.
5: Train *net* with T_1 and store γ_T in y_T and γ_V in y_V.
6: Train *net* with T_2 and store γ_T in Y_T and γ_V in Y_V.
7: Create two Random Forest f_t and f_v.
8: Use f_T for regression with M_1 and y_T
9: Use f_V for regression with M_1 and y_V
10: Obtain $\hat{y_T}$ using f_T to predict the response variable with M_2
11: Obtain $\hat{y_V}$ using f_V to predict the response variable with M_2
12: Compute $\hat{net}_T$ = sMAPE($\hat{y_T}$,Y_T)
13: Compute $\hat{net}_V$ = sMAPE($\hat{y_V}$,Y_V)

Algorithm 3 performs the following way: in step 1 it splits the time series data in order to model the ANN with one group and validate the modelling with the other, steps 2

and 3 extract the time series characteristics of both groups (these characteristics will be the problem instance for the modelling) and create the matrixes M_1 and M_2 that will be used by random forests, steps 4, 5 and 6 select a particular ANN and train it with time series in both groups in order to have the real ANNs errors, steps 7, 8 and 9 performs the regression with random forests with the problem instances created in step 2 and the response variables obtained in step 5, steps 10 and 11 uses the trained random forests created in step 7 to predict the expected output for problem instances in the second group of time series X_2, finally in steps 12 and 13 the sMAPE measure between the real ANNs and modelled ANNs is computed over the second group of time series. The value $\hat{net}_t$ represents the error between the real ANN and the modelled ANN in the training set and $\hat{net}_v$ is the error between the real ANN and the modelled in the validation set. In other words, this two measures represent how correct was the modelling respect to the real ANN.

The function $sMAPE(Y, \hat{Y})$, computes the sMAPE error between y and $\hat{y}$. The term sMAPE stands for *Symmetric Mean Absolute Percentage Error*, the sMAPE measure is defined in Equation (4.1) where y_i is the i_{th} observation of the time series and $\hat{y}_i$ is the prediction for that observation. This work uses sMAPE measure because is an standard when it comes to compare two predictors.

$$sMAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{(y_i + \hat{y}_i)/2} \tag{4.1}$$

4.3. Time Series

A Time Series is a set of equally time-spaced observations for a particular variable during a period of time. They have natural temporal ordering, this means that the time series begins with the observations occurred first in time followed by the ones occurred later on. Time series are important because they represent the behavior of many systems and phenomena. Therefore, the understanding of time series leads to a better understanding of the systems that generate them. Having analyzed the time series, the unknown observations or future values of the time series can be predicted.

Time Series are useful to represent data in different areas such as Statistics, Signal Processing, Weather Forecasting, Pattern Recognition, etc. For demonstration purposes, it is relatively simple to construct a time series, for example a time series constructed by a random integer function is shown in figure 4.1. Certainly, it only represents a random

variable measured 50 times, it does not represent any phenomena of interest but it gives a basic idea of how a time series typically looks like. In practice, time series are not constructed but collected from real measurements of variables in the field of interest.

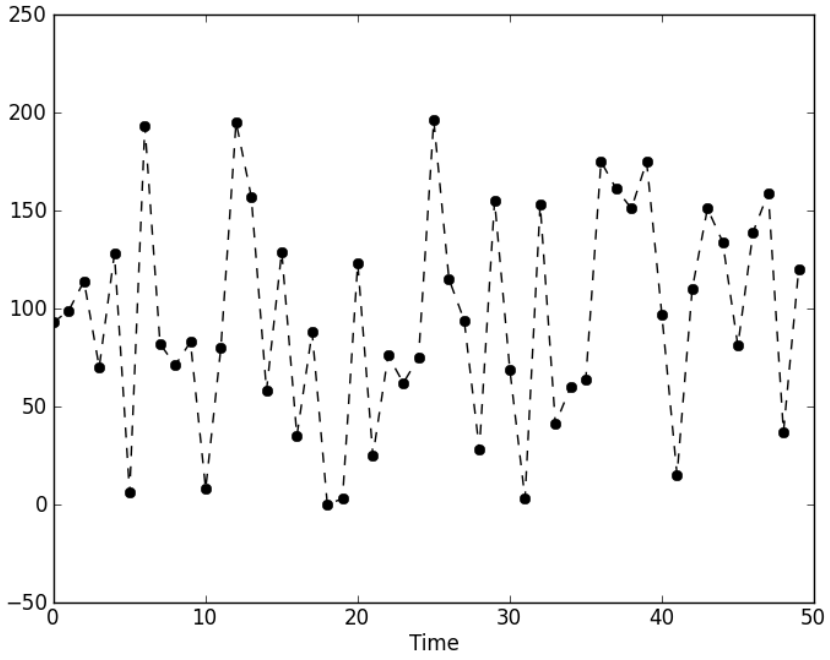

Figure 4.1: Time Series of a random variable.

4.3.1. Time Series Characteristics

When problems like Prediction and Classification are being solved, the more information we have about the problem, the better the chances are to apply the correct method. Additionally, if there are several options to solve those tasks, this additional information can help the method being used to achieve better results. In the specific case of time series, it would be highly appreciated to have any other kind of information that could help. Unfortunately, this is not the case. In order to overcome this, certain *characteristics* of the time series can be extracted. This in fact, will provide this extra information. These time series *characteristics* or features, which have been identified and studied in [Wang et al., 2009], include the traditional statistics such as: *trend, seasonality, periodicity, serial correlation, skewness, and kurtosis.* Other new features have also been included like *non-linearity*

structure, self-similarity, and chaos.

Some of these features are obtained in the flat or *raw* time series (i,e., the original information before filtering or preprocessing) but other features are calculated on the resulting time series after de-trending and de-seasonalizing the original time series. This decomposition is often referred as *Trend and Seasonally Adjusted (TSA) data*. Table 4.1 shows which features are calculated in RAW and TSA data.

Table 4.1: Time Series Characteristics

Feature	**RAW**	**TSA**
Trend	yes	no
Seasonality	yes	no
Serial Correlation	yes	yes
Non-linearity	yes	yes
Skewness	yes	yes
Kurtosis	yes	yes
Self-similarity	yes	no
Chaos	yes	no
Periodicity (frequency)	yes	no

Trend and Seasonality

Trend and seasonality are common features of time series, normally used to describe them. Once trend and seasonality of a time series have been calculated, the time series can be de-trend and de-seasonalized. This process makes it easier to detect other features that can only be calculated on the TSA data. The seasonality of a time series is defined as a pattern that repeats itself over fixed intervals of time [Makridakis and Wheelwright, 1978]. A trend pattern exists when there is a long-term change in the mean level [Makridakis and Wheelwright, 1978]. In [Wang et al., 2009], Wang *et al.* use the model given in Equation (4.2) to describe a time series, where Y_t^* denotes the time series after Box-Cox transformation at time t, T_t is the trend, S_t is the seasonal component, and E_t is the remainder component.

$$Y_t^* = T_t + S_t + E_t \tag{4.2}$$

When the trend and seasonal components (T_t and S_t) have been computed, the de-trended data X_t is $X_t = Y_t^* - T_t$ and the de-seasonalized Z_t is calculated by $Z_t = Y_t^* - S_t$. The remainder is defined as $Y_t^{'} = Y_t^* - T_t - S_t$. Then, a measure of trend is calculated by

$1 - \frac{\Sigma(Y_t^{'})}{Z_t}$ and the measure of seasonality is $1 - \frac{\Sigma(Y_t^{'})}{X_t}$.

Periodicity

Periodicity is intimately related to the seasonality, this is because the seasonality is determined with the help of the periodicity. Therefore, the extraction of periodicity becomes a basic step in the extraction of time series characteristics. Wang *et al.* [Wang et al., 2009] propose an algorithm to measure periodicity because most times, time series do not come with a known frequency parameter, i.e., regular periodicity.

Serial Correlation

The serial correlation has been estimated by Box-Pierce statics [Makridakis and Wheelwright, 1978]. This statistic was designed by Box and Pierce in 1970 and is given by $Q_h = n\sum_{k=1}^{h} r_k^2$ where n is the length of the time series, h is the maximum lag considered (usually $h \approx 20$) and $r_k = Corr(Y_t, Y_{t-k})$.

Non-linearity

Non-linear time series have received attention in recent years due to the reason that the model complex dynamics do not adequately represent linear models [Wang et al., 2009]. Non-linearity is an important feature of time series in order to choose the best approach to predict them. [Wang et al., 2009] use Terasvirta's neural network to extract nonlinearity [Terasvirta et al., 1993].

Skewness

Skewness is a measure of asymmetry of the time series. The dataset is called to be symmetric if we can establish a center point and both sides to the left and right of the center point look the same. A skewness measure is used to quantify the degree of asymmetry of the values around the mean value. The skewness coefficient is given by $S = \frac{1}{n\sigma^3}\sum_{t=1}^{n}(Y_t - \hat{Y})^3$ where $\hat{Y}$ is the mean, σ is the standard deviation and n is the number of data points. The skewness for a normal distribution is zero and any data considered symmetric must have a skewness near zero. If skewness turns out to be positive this indicates that the data are skewed to the right and to the left otherwise.

Kurtosis

Kurtosis is a measure that indicates if the data presents peaks or if it is mostly flat similarly to the peaks and flatness present in a normal distribution. If a set presents high kurtosis this means that the data present one high peak near the mean with fast decline in any other points. Datasets with low kurtosis present a low peak near the mean value rather than a large one. The kurtosis coefficient is given by $K = \frac{1}{n\sigma^4}\sum_{t=1}^{n}(Y_t - \hat{Y})^4 - 3$ for a normal distribution [Wang et al., 2009]. High kurtosis represents peaked data and low kurtosis represent flatted data.

Self-similarity

Self-similarity is not a common feature in time series to extract and it is neglected in time series feature identification. Since processes with long-range dependance are becoming more common in many fields [Wang et al., 2009] have decided to include it in the final set of time series features. The definition that best fits the properties of time series is the self-similarity parameter Hurst exponent H [Adler et al., 1998].

Chaos

For many years the behavior of systems that apparently does not have any feature like the ones described above were called systems or processes with random behavior. Nowadays, the most appropriate term is *chaotic systems*. Nonlinear dynamical systems often exhibit chaos and are normally characterized by sensitive dependance on initial values given by a Lyapunov Exponent (LE) [Wang et al., 2009]. The first algorithm to compute LE for time series was proposed in [Wolf et al., 1985] but in the set of time series features proposed by [Wang et al., 2009] the method demonstrated by Hilborn *et al.* [Hilborn, 2000] was used.

Difficulty indicators

Besides the 9 characteristics (plus 4 calculated in the TSA data) considered above, another 20 characteristics which represent *difficulty indicators* as described in [Mario Graff and Gonzalez, 2013] are also being considered. These characteristics or difficulty indicators are based on the discrete derivative of a function f w.r.t. a variable x, defined in Equation (4.3), where h is desired to be as small as possible.

$$\Delta_h f(x) = \frac{f(x+h) - f(x)}{h} \tag{4.3}$$

In order to use this discrete derivative it has to be converted into a time series domain. In the time series domain f represents a time series and $f(x)$ represents an observation of f at the time x. As said earlier, in the real domain, h is wanted to be as small as possible but in the time series domain there are certain limitations. By definition, the observations of a time series are equally spaced and the smallest value for h is 1 (there are no values between observations). Similarly, the biggest value for h is limited to the number of observations that f (a time series) has. If N is the number of observations, the largest value for h is $N - 1$ because if h becomes bigger, it means that the discrete derivative is trying to obtain a value for an observation that does not even exists.

Equation (4.3) defines how to apply the concept of the discrete derivative how to approximate its values in the time series domain choosing appropriate values for its parameters, this is only for an observation of the time series f. In order to apply the same concept but in the whole time series rather than in a single value, Equation (4.4) must be used, which represents a generalization of Equation (4.3).

$$\delta_i(f) = \frac{1}{|N|} \sum_{x \in N} |\Delta_i f(x)| \tag{4.4}$$

Equation (4.4) calculates the concept of the discrete derivative over a time series but only for a fixed value of h. The last equation shows how to calculate these difficulty indicators using the discrete derivative approach on the time series. Just like in other problems, sometimes the first derivative of a function does not give much information about the problem under study. Having said that, these difficulty indicators can be easily extended to calculate derivatives of higher orders. Calculating derivatives of higher orders could give more information that can be used to comprehend the behavior of the predictors, or in the best case, improve them. Equation (4.5) presents how to calculate higher order derivatives using the discrete derivative approach.

$$\zeta_i(f) = \frac{1}{|N|} \sum_{x \in N} |\Delta_h f^{(i)}(x)| \tag{4.5}$$

This difficulty indicators of time series have been defined for higher order derivatives to provide additional information and can be extended even more. In the previous

formulations, the step size h was set to 1. This is because the difficulty indicators were designed to follow closely the concept of the discrete derivative, where h is desired to be as small as possible. However, following the same reasons why higher order derivatives were calculated, it can be investigated if different values for h result in good indicators. The step size h can be varied from 2 to $N-1$ and, in order to combine this variation with the formulations above, a super index is added to ζ, indicating the value for the step size h. This formulation is presented in Equation (4.6).

$$\zeta_i^h(f) = \frac{1}{|N|} \sum_{x \in N} |\Delta_h f^{(i)}(x)| \tag{4.6}$$

4.4. Summary

This chapter shows how to model an ANN to have an idea of what the performance of a specific ANN over a collection of time series would be. If the performance of the ANN could be known *a priori*, then the possibility of choosing the ANN model that will be expected to perform best increases. The process of modelling is not a trivial task regardless of the system that is intended to be modelled. This is the results obtained by PID are used in order to simplify this task. The results presented in Chapter 5 show that it is actually possible to know the expected performance of an ANN in order to make further decisions related to the selection of the best and simpler approach to use.

Chapter 5

Results

This chapter presents the results obtained by PID and the modelling presented in Chapter 4. In Section 5.1 the parameter order influence for the QUICKPROP and RPROP training algorithms is presented for training and validation phases. Section 5.2 depicts the results obtained by the ANNs modelled. And finally Section 5.3 presents the chapter conclusions.

5.1. Results of PID

This section explains the results obtained by PID. The input for the PID algorithm is a time series, and the output is a list where the parameters are sorted by their importance to the training algorithm being used. This does not tell much about the influence of the parameters in the training algorithm. It only provides the information about what parameters are influencing more the training algorithm for that particular time series.

5.1.1. Test settings

Time series have several characteristics (reviewed in Chapter 4), and therefore, the results obtained by PID are valid just for the time series used. One of the objectives of this work is to determine the order of parameters' influence on the training algorithm for time series prediction. So, in order to accomplish this, PID has to be tested over a number of time series. This collection of time series must include different time series with different characteristics in order to obtain rich information about how the training algorithm behaves when its parameters are varied. A collection of 1001 time series called *M1* [Makridakis et al.,

1982] were used for this purpose. All the time series in this collection are used to benchmark different approaches for time series prediction.

One last consideration about the tests concerns the number of free parameters. Besides the three parameters for QUICKPROP and the five parameters for RPROP an extra parameter has been added, which represents the number of neurons used in the hidden layer. The number of neurons in the hidden layer is well known to be directly related to the way an ANN learns. An ANN with many neurons in the hidden layer can get overtrained and therefore the ANN will not be able to generalize. The main reason to include the number of hidden neurons as a parameter is because this work will demonstrate that a carefully selection of values for the training parameters is equally important than the architecture of the ANN itself, in this case, the size of the hidden layer. This is: an ANN with lees neurons in the hidden layer can outperform another one with more neurons, if the parameters are carefully selected.

5.1.2. PID Test

The tables that present the results are organized as follows: they have four columns numbered from 1 to 4, which describe the order of selection of the parameters, being number 1 the first parameter selected and number 4 the last. Besides the order in which a parameter was selected, a percentage of times where it was selected in a specific order is also presented. This percentage of times is shown below each column, in such way that, every parameter has the columns of order and the percentage of times it was selected in that order below (or example in Table 5.1, parameter decay was chosen 90 % of times as the 4th parameter in importance).

Results for QUICKPROP

The results for the QUICKPROP algorithm are presented for the training stage first and then for the validation stage. Table 5.1 presents the results for the training stage. The learning factor was selected 90 % of the times as the most important, followed by the hidden neurons which were chosen 60 % of the time in second place. The parameter mu was chosen in third place 54 % of the time and the least important parameter was the decay factor, who was chosen in last place the 90 % of the time.

Table 5.2 presents the results of the QUICKPROP algorithm for the validation

stage. There are no changes in the parameter order compared to the training stage. The learning factor was again chosen in first place 74 % of the time but mu increased the number of selections in first place. The number of hidden neurons was selected again in second place 46 % of the time but this time, it was also selected 6 % of the time in first place, contrarily to 0 % in the training set. The parameter mu increase the number of times it was chosen in first place from 8 % in the training set to 18 % in the validation set. The parameter order in the validation stage does not change compared to the training, because parameter mu was selected fewer times in second place than the number hidden neurons. The decay factor once again was selected 90 % of the time in last place.

It can be concluded that for the QUICKPROP training algorithm, the learning factor is the most important parameter followed by the number of hidden neurons. Contrarily, the results show that the decay factor is not taken into account very much for time series prediction. The mu parameter exhibited close results to the number of hidden neurons; this means that it is equally important to choose the size of the hidden layer as the values for the parameters, this was mentioned in Subsection 5.1.1, and was the main reason to include the hidden neurons as a parameter.

Table 5.1: QUICKPROP Training results

Parameter	**Parameter Percentages**			
Order of Parameters	**1**	**2**	**3**	**4**
learning factor	**90**	8	2	0
hidden neurons	0	**60**	36	4
mu	8	32	**54**	6
decay factor	2	0	8	**90**

Table 5.2: QUICKPROP Validation results

Parameter	**Parameter Percentages**			
Order of Parameters	**1**	**2**	**3**	**4**
learning factor	**74**	20	4	2
mu	18	32	**48**	2
hidden neurons	6	**46**	42	6
decay factor	2	2	6	**90**

Results for RPROP

The results for the RPROP algorithm are easier to understand and interpret. For the training stage, the results in Table 5.3 show that the 100 % of the time the increase factor is the most important parameter. The second place is for the decrease factor with also the 100 % of the time. The third place is for the hidden neurons with 78 % of the times. The parameters delta max and delta zero were chosen in fourth and fifth place with 44 % and 58 % of the time respectively. The last place was for the parameter delta min, it was chosen in last place all the time.

For RPROP in the validation set, the results presented in Table 5.4, the are no changes in the order. The first three places are for the increase factor, the decrease factor and the hidden neurons, the same order that in the training set. The change in the order of influence for the validation set is for the parameters zero and delta max. Now delta zero was in fourth place 68 % of the times. The parameter delta min is still in last place again, with the 100 % of the times.

What can be inferred from these results is that the RPROP training algorithm is highly sensible to two parameters: the increase and decrease factors. The opposite occurs with two parameters: delta min and delta max, which barely affect the behavior of the algorithm. These results are important, because a final user can spend time tuning the parameters which the training algorithm is sensible to and almost forget about the parameters that do not matter for time series prediction.

Table 5.3: RPROP Training results

Parameter	**Parameter Percentages**					
Order of Parameters	**1**	**2**	**3**	**4**	**5**	**6**
increase factor	**100**	0	0	0	0	0
decrease factor	0	**100**	0	0	0	0
hidden neurons	0	0	**78**	18	4	0
dmax	0	0	18	**44**	38	0
zero	0	0	4	38	**58**	0
dmin	0	0	0	0	0	**100**

Table 5.4: RPROP Validation results

Parameter	**Parameter Percentages**					
Order of Parameters	**1**	**2**	**3**	**4**	**5**	**6**
increase factor	**94**	6	0	0	0	0
decrease factor	6	**94**	0	0	0	0
hidden neurons	0	0	**76**	24	0	0
dmax	0	0	12	8	**80**	0
zero	0	0	12	**68**	20	0
dmin	0	0	0	0	0	**100**

5.1.3. PID Comparisson

Besides the results obtained by PID in both algorithms, values for the training algorithms that present low MSE error (small γ_x) were identified. This values were compared against the default values for the training parameters and the samples were tested with the Wicoxon's paired T test [Wilcoxon, 1945] to assure that the mean of their performance i.e., mean of γ, differs. The Wilcoxon paired T test gave a confidence of 99 % in the case of RPROP and 100 % in the case of QUICKPROP that the mean of the samples are different. This means that the values founded outperform the default values.

Of course, these values are only valid for the tested time series, and it would be risky to assume that these recommendations for parameters values will achieve good results in time series collected from different areas, but for the M1 collection they work well minimizing as much as possible the MSE error. These values are shown in Tables 5.5 and 5.6, which are contrasted with the *default* values proposed for both algorithms. Besides, these values found managed to minimize the MSE error in the order of 1×10^{-3} for the QUICKPROP algorithm and in the order of 1×10^{-2} in the case of the RPROP algorithm. Normally, the desirable MSE error of an ANN is in the order o 1×10^{-1}. This is because very small errors, like errors in the order of 1×10^{-3} will present poor generalization capabilities, i.e., overtraining. The errors obtained by the values found by PID are so small because PID trained the ANN for 3,000 epochs. These errors could be in the desirable range if the ANN is trained for a smaller number of epochs.

Table 5.5: QUICKPROP default values vs. experimental values

Parameter	**Default**	**Experimental**
learning factor	0.7	1.05
decay factor	-0.0001	-5×10^{-5}
mu	1.75	2.05

Table 5.6: RPROP default values vs experimental values

Parameter	**Default**	**Experimental**
increase factor	1.2	1.8
decrease factor	0.5	0.375
dmin	0	0
dmax	50	50
zero	0.1	0.15

Another values for the training parameters that performs better than the default values proposed have been found. This was possible because thanks to the results of PID, this work spend more time tuning the most important training parameters.

5.2. Modelling Results

This section presents the results obtained in Chapter 4, where several ANN models trained with the QUICKPROP and RPROP algorithms were modelled. When an ANN is trained, is common to qualify its performance by the MSE measure. For the modeling results, this work uses the sMAPE measure instead of the MSE measure. The results obtained in this section have to be interpreted as how well the modelled ANN fits the actual performance of the real ANN, not the sMAPE values of the plots. It is true that higher sMAPE values indicate poor performance, but if the real ANN had poor performance the modelling (if it was successful) will indicate poor performance as well.

Each plot represents the modelling of a particular ANN model. The plots consist of two things: a number of points and a curve. Each point represents the performance of the ANN model given a time series, and the curve represents the model obtained with random forests. If these points are very close to the curve, it means that the model obtained with random forests is accurate respect to the real ANN.

This work modelled 225 ANNs trained with RPROP and the same number trained with QUICKPROP. Additionally, the difference between models with the same training

algorithms is only the values of the most important parameter of the training algorithm to be used and the number of neurons in the hidden layer. The reason to vary the number of hidden neurons is because this parameter affects the training capabilities and the ability to generalize well.

As mentioned earlier, each dot in the plot has a particular sMAPE value which is the real performance of the ANN for that time series. At the top of the plot, the additional sMAPE value indicates how close all the points in the plot from the curve are. If this value is small it means that the curve fits well all the points. On the other hand, a large value indicates that the model obtained is not accurate, in other words, it is not representing the performance of the ANN model.

5.2.1. Modelling ANNs trained with QUICKPROP algorithm

Beginning with the QUICKPROP training algorithm Figures 5.1 and 5.2 present a modelled ANN that performs well in both training and validation sets in the majority of the time series. For the training stage it has an error of 7.32 % and in the validation stage the error is 5.44 %. This means that in the training stage the modelling has an accuracy around 93 % and around 95 % for the validation stage.

In Figures 5.3 and 5.4 the accuracy of the modelling is also around 93 % in the training but in the validation stage the accuracy is around 55 %. This means that this modelling can not be trusted to provide a good approximation of how the real ANN will perform in the validation set.

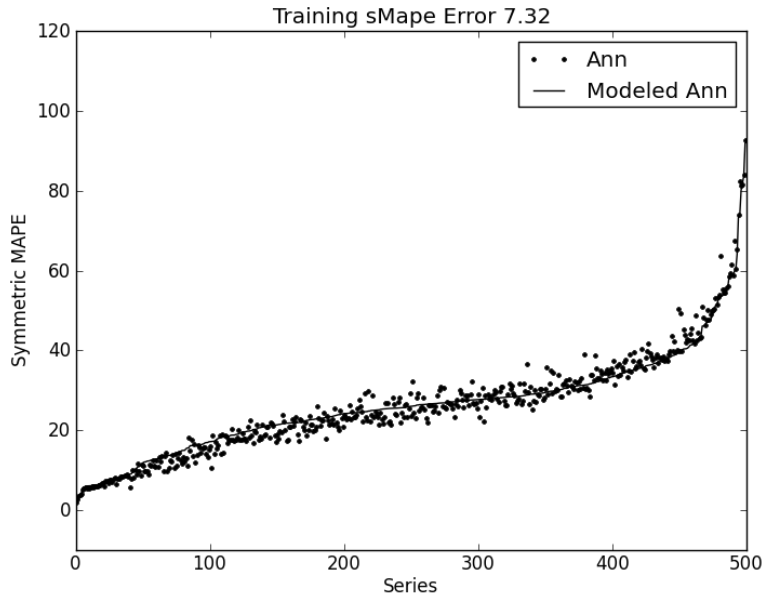

Figure 5.1: modelling in training stage with known time series (QUICKPROP)

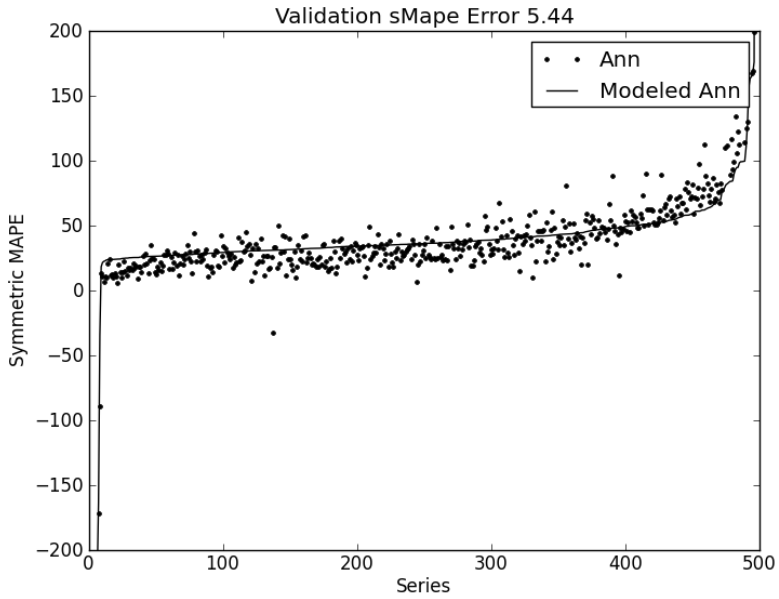

Figure 5.2: modelling in validation stage with known time series (QUICKPROP)

Figure 5.3: modelling in training stage with known time series (QUICKPROP)

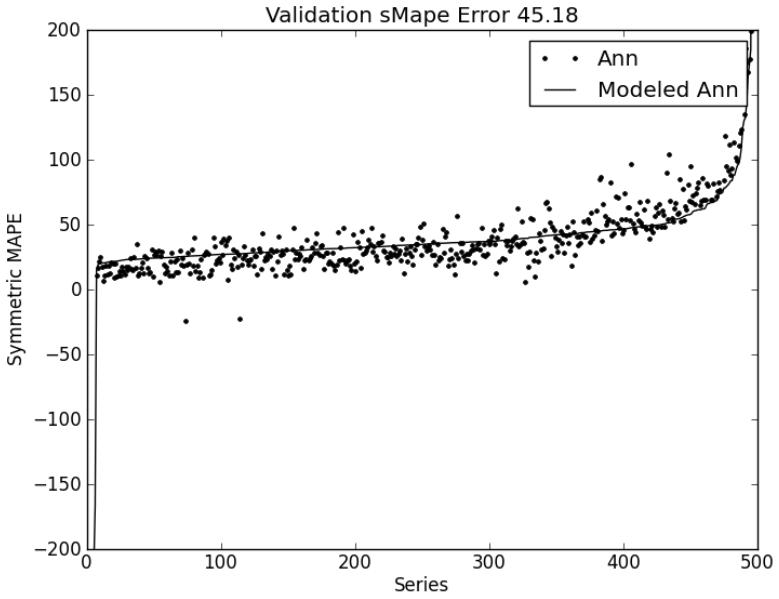

Figure 5.4: modelling in validation stage with known time series (QUICKPROP)

The results previously depicted correspond to a modelling where once the modelled

ANN is created, the same data used to model, is now presented again in order to measure how close the modelling is to the real ANN. By doing this, the model is being fed with *known* time series, this is, series that the model already knows. If the algorithm selection problem is being tackled, the modelling has to be tested with data that had not been presented to it in order to prove its real capabilities, this is feeding it with *unknown* time series.

The results of QUICKPROP with unknown time series are shown in Figures 5.5 and 5.6. The accuracy is now around 93 % in the training stage and an accuracy around 74 % in the validation stage. This means that the modelling manages to approximate well the performance in the validation stage of a given ANN, and most importantly, with data that was not presented to it. These results showed that it is possible to model an ANN with random forests with some known data, and then use this modelling to know, *a priori* (with some degree of confidence), what the expected performance of this given ANN with new data is.

Thanks to this, we have come with a way to tackle the algorithm selection problem. We can have several ANNs modelated with random forests and prove it with new data, if the results shown that ANNs are expected to perform well, we can consider using them to solve the problem under analysis. In the case that the results show the opposite way, we are in position of consider another techniques.

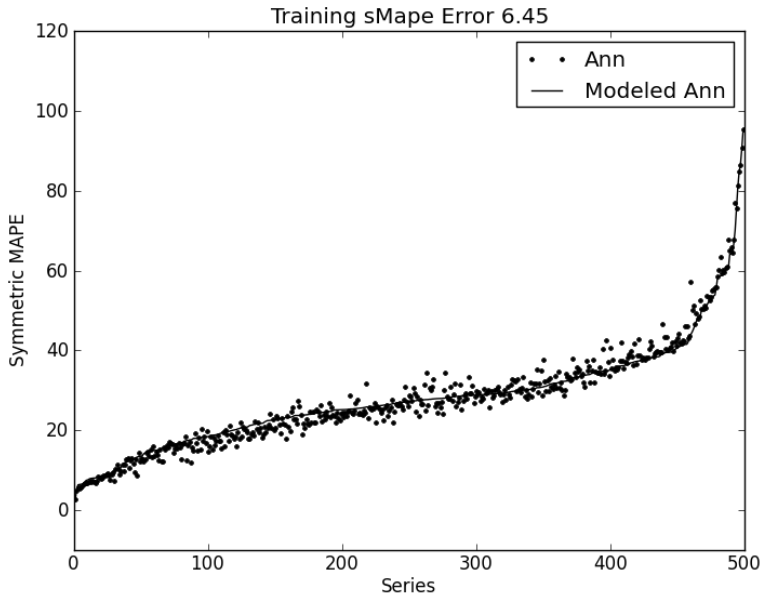

Figure 5.5: modelling in training stage with unknown time series (QUICKPROP)

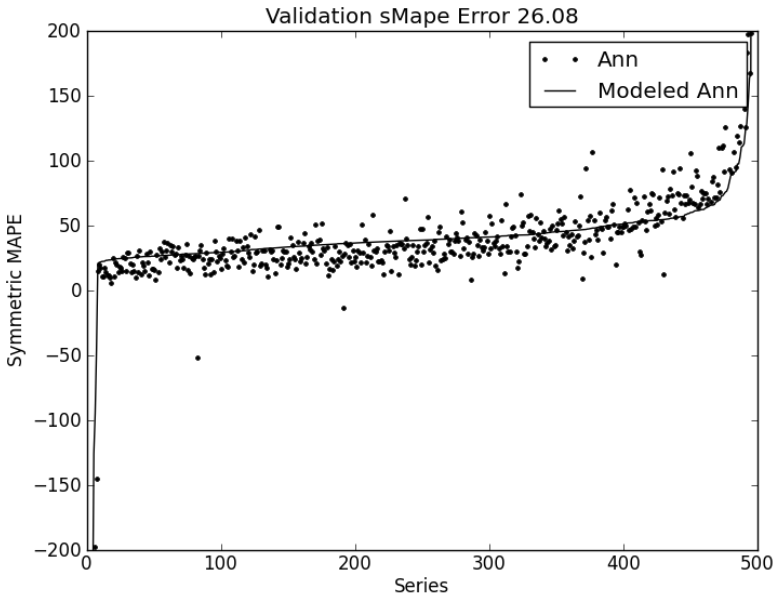

Figure 5.6: modelling in validation stage with unknown time series (QUICKPROP)

5.2.2. Modelling ANNs trained with RPROP algorithm

For the RPROP algorithm the results are presented in the same way: first the modelling results with known time series and then with unknown time series. Figures 5.7 and 5.8 show the modelling with known time series. The accuracy is around 94 % in the training stage and around 80 % in the validation stage. These results are good but they correspond to data that the model already know. The interesting results are the ones obtained with time series new to the model. Figures 5.9 and 5.10 presents this case. In these figures we can see an accuracy around 93 % in the training stage and around 80 % in the validation stage.

These results are important because it proves that random forests could be used for modelling ANNs with accuracy. In order to solve the algorithm selection problem, ANNs can be modelled with random forests and then decide wether ANNs are suitable techniques for the problem under analysis or not. This modelling results can be improved if more time is spend adjusting all the parameters that random forests have. Unfortunately, the tuning of the random forests parameters is not a trivial task (as the tuning of ANN parameters). Random forests have also *default* values for the parameters that are the ones used in this thesis. If this tuning of random forests' parameters is taken more consciously it is not an error to think that the modelling results will improve significantly from the ones shown in this work.

Figure 5.7: modelling in training stage with known time series (RPROP)

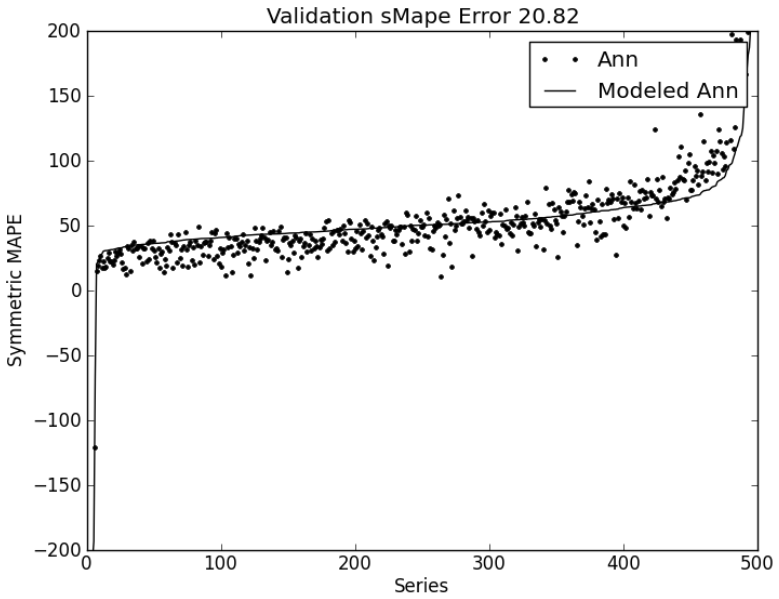

Figure 5.8: modelling in validation stage with known time series (RPROP)

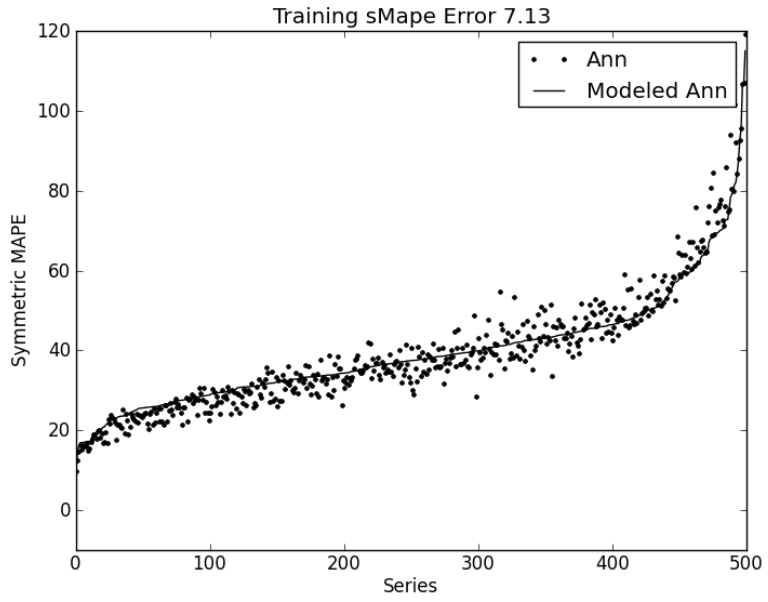

Figure 5.9: modelling in training stage with unknown time series (RPROP)

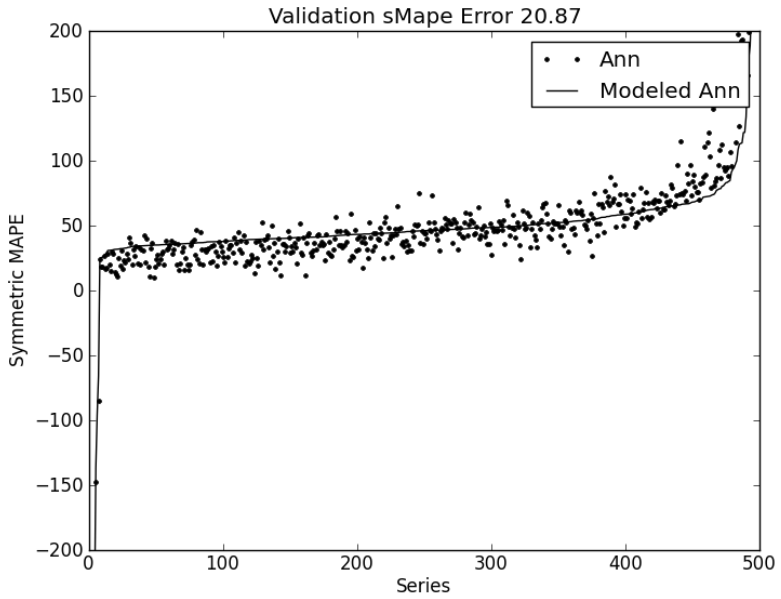

Figure 5.10: modelling in validation stage with unknown time series (RPROP)

5.3. Summary

The results of modelling an ANN using random forests have been presented. The reason why this thesis includes modelling is two fold: to prove that an ANN can be modelled only varying the parameters that PID selected as the most influential for the training algorithms, and to tackle the algorithm selection problem. The models obtained in this chapter are trained with a set of time series and proved with this same set to validate the model and then a new set of time series is presented to it. This different set of unknown time series represents the hypothetical case where new data is obtained and is mandatory to know *a priori* the expected performance of a given ANN. This is important because is pointless to model an ANN and prove it in data where we already know the real performance of an ANN. Random forests have proved that they can be a good technique to model ANNs and with a little more effort in the tuning of their parameters, based on the results, the user can achieve outstanding results.

Chapter 6

Conclusions

This chapter presents the final conclusions that have been collected along this thesis. Section 6.1 discusses the achievements of this work. Section 6.2 explains the limitations of this work. Finally, Section 6.3, presents ideas for possible future works.

The main objective of this thesis was to determine the influence of the parameters of an ANN training algorithm. This analysis was done for two training algorithms: QUICKPROP and RPROP. Both algorithms represent improved versions of the traditional Back propagation algorithm. The ANNs with different values in the training parameters were used to solve the Time Series prediction problem. This thesis was aimed in determining the influence of parameters in the training algorithms (3 parameters for QUICKPROP and 5 Parameters for RPROP).

6.1. Achievements

We have determined the parameter influence because, if we know which parameters are more important to the algorithm we can pay more attention in the selection of their values or tuning, and came up with an ANN model that has better possibilities of solving the task well. If the user knows what parameter is more influential to the training algorithm, he can spend more time finding good values for the parameter. Furthermore, if one parameter has little influence there is no need to pay much attention to it. There is not much information about what parameters are the most influential, the literature only gives *default* values for the parameters that have achieved good results for the majority of problems. But if the user wants to explore different values that could achieve better results,

he is on his own. We hope that the results obtained in this work could help for tackling this issue, even when they are only valid for the time series prediction problem.

As mentioned in Chapter 1 Section 1.3, ANNs are complex models, generally more expensive, computationally speaking, than other approaches. There are problems where ANNs are not as good as other techniques, even with a carefully selection not only of the training parameters but on the design of the ANN itself. When a situation like this occurs, it would be preferable to use some simpler method that outperform ANNs. The basic idea of the algorithm selection problem is that given one problem and two or more different algorithms to solve it, it is always preferable to choose the simplest and best method.

In order to solve the algorithm selection problem, we have to decide which of the available algorithms is the best approach for the problem we are trying to solve. This is far from being a trivial task, because the no free lunch theorem [Wolpert and Macready, Apr], states that there is no algorithm that would perform the best for all the problems. This means that, for all algorithms there exists one problem instance for which a particular algorithm is the best solution to that problem. The trivial solution for this problem is to test all the algorithms in the given problem, and then choose the one with the best performance. It is obvious that this strategy is far from being the optimal solution, because the resources and time needed to accomplish this is the most expensive solution.

This is the reason why we also modelled an ANN. Because this modelled ANN is actually much simpler than the real ANN and it would be an approximation of how well the ANN will perform given some problem. In that way, we would know *a priori* if the ANN is a suitable approach or if we have to select a simpler and better technique. ANNs are not simple models, since modelling an ANN involves so many variables in order to have good results. We have managed to model ANNs based only in the results of PID; that is, we modelled an ANN based only in the most influential parameters for the training algorithm. The final result is that we would know how a combination of values for the parameters would perform in the ANN without actually proving the real ANN, which is more expensive computationally.

PID and the process of modelling an ANN were achieved turning both problems into a regression problem. We used a *Random forests* to solve the regression problem representation of PID and the modelling. Random forests are powerful tools for classification and regression that are parallelized in order to improve efficiency.

6.2. Limitations

The limitations of the results obtained by PID is that they are only valid for the Time series prediction problem. The most notorious improvement for this work lies on changing the application or the problem being solved by the ANNs. Instead of the Time series prediction problem, this work could be applied on any problem that ANNs handle, whether they are classification or regression problems. It would be an enormous error to suppose that the influence of the parameters obtained will remain equal for all the problems that an ANN can handle. Also, the results presented in this work hold for the ANN architecture used in this work (3-layer feed-forward network). Different architectures may result in different behaviors of the ANN and different parameter orderings.

For the modelling, the limitations are that if we choose an ANN with bad values on its training parameters and we want to model it, we will encounter that the output of the modelled ANN and the real ANN are totally different. This is certainly not an error, because we are trying to model an ANN whose performance is very poor. Also, with a fine tuning of the random forests parameters used in the modelling, the results will improve greatly.

Another issue of PID is that it is slow. This is because PID trains a large number of ANNs for each training algorithm. Furthermore, every single ANN model is trained for 3,000 epochs. This could be a little bit exaggerated but we assure that in the later epochs the ANN has converged into a good solution during training phase.

6.3. Future Work

The parameter influence on the training algorithms QUICKPROP and RPROP had been determined, but these are not the only algorithms available for ANN training. One of the algorithms that have exhibited a good performance is SaRPROP [**?**], which is based on Simulated Annealing optimization [Kirkpatrick et al., 1983]. Because of this, one way to expand this work is to determine the parameter influence for SaRPROP.

As for the modelling of ANNs is concerned, a way to improve it is to fine tune the parameters of the random forests, the main idea of PID can be applied to determine the influence of the random forests parameters and diminish complexity of the task.

References

[Adler et al., 1998] Adler, R., Feldman, R., a Taqqu, M. (1998). *A Practical Guide to Heavy Tails: Statistical Techniques and Applications*. Birkhäuser Boston.

[Almeida and Ludermir, 2008] Almeida, L. M. a Ludermir, T. B. (2008). Tuning artificial neural networks parameters using an evolutionary algorithm. *International Conference on Hybrid Intelligent Systems.*, 0:927–930.

[Breiman, 1993] Breiman, L. (1993). *Classification and regression trees*. CRC press.

[Breiman, 2001] Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

[Caruana and Niculescu-Mizil, 2006] Caruana, R. a Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 161–168, New York, NY, USA. ACM.

[Chen et al., 2003] Chen, A.-S., Leung, M. T., a Daouk, H. (2003). Application of neural networks to an emerging financial market: forecasting and trading the taiwan stock index. *Computers & Operations Research*, 30(6):901 – 923. ¡ce:title¿Operation Research in Emerging Economics¡/ce:title¿.

[Cochocki and Unbehauen, 1993] Cochocki, A. a Unbehauen, R. (1993). *Neural Networks for Optimization and Signal Processing*. John Wiley & Sons, Inc., New York, NY, USA, 1st edición.

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314.

[Dobslaw, 2010] Dobslaw, F. (2010). A parameter tuning framework for metaheuristics

based on design of experiments and artificial neural [elektronisk resurs]... *Proceeding of the International Conference on Computer Mathematics and Natural Computing 2010*.

[Efron, 2004] Efron, B. (2004). Least angle regression. *The Annals of Statistics*, 32(2):407–499.

[Er et al., 2002] Er, M. J., Wu, S., Lu, J., a Toh, H. L. (2002). Face recognition with radial basis function (rbf) neural networks. *Neural Networks, IEEE Transactions on*, 13(3):697 –710.

[Fahlman, 1988] Fahlman, S. (1988). *An Empirical Study of Learning Speed in Back-propagation Networks*. Number v. 88-162 in Research paper. Carnegie Mellon University, Computer Science Department.

[Falco et al., 1997] Falco, I. D., Cioppa, A. D., Natale, P., a Tarantino, E. (1997). Artificial neural networks optimization by means of evolutionary algorithms.

[Flood and Kartam, 1994] Flood, I. a Kartam, N. (1994). Neural networks in civil engineering. i: Principles and understanding. *Journal of Computing in Civil Engineering*, 8(2):131–148.

[Fogel et al., 1995] Fogel, D. B., III, E. C. W., a Boughton, E. M. (1995). Evolving neural networks for detecting breast cancer. *Cancer Letters*, 96(1):49 – 53.

[Hagan and Demuth, 1999] Hagan, M. a Demuth, H. (1999). Neural networks for control. In *American Control Conference, 1999. Proceedings of the 1999*, volume 3, pages 1642 –1656 vol.3.

[Herbrich et al., 1999] Herbrich, R., Keilbach, M., Graepel, T., Bollmann-Sdorra, P., a Obermayer, K. (1999). Neural networks in economics. In Brenner, T., editor, *Computational Techniques for Modelling Learning in Economics*, volume 11 of *Advances in Computational Economics*, pages 169–196. Springer US.

[Hilborn, 2000] Hilborn, R. (2000). *Chaos and Nonlinear Dynamics: An Introduction for Scientists and Engineers*. Oxford University Press.

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., a White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2(5):359–366.

[Hutter et al.,] Hutter, F., Hoos, H. H., a Leyton-Brown, K. Identifying key algorithm parameters and instance features using forward selection.

[Kariya and Kurata, 2004] Kariya, T. a Kurata, H. (2004). *Generalized least squares*. Wiley.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C. D., a Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

[Lawrence et al., 1997] Lawrence, S., Giles, C., Tsoi, A. C., a Back, A. (1997). Face recognition: a convolutional neural-network approach. *Neural Networks, IEEE Transactions on*, 8(1):98 –113.

[Ma and Khorasani, 2004] Ma, L. a Khorasani, K. (2004). Facial expression recognition using constructive feedforward neural networks. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(3):1588 –1595.

[Makridakis et al., 1982] Makridakis, S., Andersen, A., Carbone, R., Fildes, R., Hibon, M., Lewandowski, R., Newton, J., Parzen, E., a Winkler, R. (1982). The accuracy of extrapolation (time series) methods: Results of a forecasting competition. *Journal of Forecasting*, 1(2):111153.

[Makridakis and Wheelwright, 1978] Makridakis, S. a Wheelwright, S. (1978). *Forecasting: methods and applications*. Wiley/Hamilton series in management and administration. Wiley.

[Mario Graff and Gonzalez, 2013] Mario Graff, Hugo Jair Escalante, J. C.-J. a Gonzalez, A. A. (2013). Models of performance of time series forecasters. Accepted for publication in Neurocomputing.

[Minka, 2000] Minka, T. (2000). Bayesian linear regression. Technical report, Citeseer.

[Narendra and Parthasarathy, 1990] Narendra, K. a Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. *Neural Networks, IEEE Transactions on*, 1(1):4 –27.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., a Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Ravdin et al., 1992] Ravdin, P., Clark, G., Hilsenbeck, S., Owens, M., Vendely, P., Pandian, M., a McGuire, W. (1992). A demonstration that breast cancer recurrence can be predicted by neural network analysis. *Breast Cancer Research and Treatment*, 21:47–53.

[Reinaldo et al., 2009] Reinaldo, F., Camacho, R., Reis, L. P., a Magalhaes, D. R. (2009). Fine-tune artificial neural networks automatically. In Mastorakis, N., Mladenov, V., a Kontargyri, V. T., editors, *Proceedings of the European Computing Conference*, volume 27 of *Lecture Notes in Electrical Engineering*, pages 39–43. Springer US.

[Rice, 1975] Rice, J. R. (1975). The algorithm selection problem.

[Riedmiller and Braun, 1993] Riedmiller, M. a Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS*, pages 586–591.

[Rouhani and Soleymani, 2009] Rouhani, M. a Soleymani, R. (2009). Neural networks based diagnosis of heart arrhythmias using chaotic and nonlinear features of hrv signals. In *Computer Science and Information Technology - Spring Conference, 2009. IACSITSC '09. International Association of*, pages 545 –549.

[Rowley et al., 1998] Rowley, H., Baluja, S., a Kanade, T. (1998). Neural network-based face detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(1):23 –38.

[Saritas, 2012] Saritas, I. (2012). Prediction of breast cancer using artificial neural networks. *J. Med. Syst.*, 36(5):2901–2907.

[Terasvirta et al., 1993] Terasvirta, T., Lin, C., a Granger, C. W. J. (1993). Power of the Neural Network Linearity Test. *Journal of Time Series Analysis*, 14:209–220.

[Tibshirani, 1994] Tibshirani, R. (1994). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288.

[Tourassi et al., 1995] Tourassi, G. D., Floyd, C. E., Sostman, H. D., a Coleman, R. E. (1995). Artificial neural network for diagnosis of acute pulmonary embolism: effect of case and observer selection. *Radiology*, 194(3):889–893.

[Treadgold and Gedeon, 1996] Treadgold, N. K. a Gedeon, T. D. (1996). The sarprop algorithm: A simulated annealing enhancement to resilient back propagation. In *Proceedings International Panel Conference on Soft and Intelligent Computing*.

[Tsoukalas and Uhrig, 1996] Tsoukalas, L. H. a Uhrig, R. E. (1996). *Fuzzy and Neural Approaches in Engineering*. John Wiley & Sons, Inc., New York, NY, USA, 1st edición.

[Udo, 1993] Udo, G. (1993). Neural network performance on the bankruptcy classification problem. *Computers & Industrial Engineering*, 25(14):377 – 380.

[Wang et al., 2009] Wang, X., Smith-Miles, K., a Hyndman, R. (2009). Rule induction for forecasting method selection: Meta-learning the characteristics of univariate time series. *Neurocomput.*, 72(10-12):2581–2594.

[Werbos, 1974] Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard University Press.

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

[Wilcoxon, 1945] Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83.

[Wolf et al., 1985] Wolf, A., Swift, J. B., Swinney, H. L., a Vastano, J. A. (1985). Determining lyapunov exponents from a time series. *Physica*, pages 285–317.

[Wolpert and Macready, Apr] Wolpert, D. a Macready, W. (Apr). No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82.

[Yao et al., 1996] Yao, X., Ieee, S. M., a Liu, Y. (1996). A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 8:694–713.

[Zhang and Wang, 2008] Zhang, Q. a Wang, C. (2008). Using genetic algorithm to optimize artificial neural network: A case study on earthquake prediction. In *Genetic and Evolutionary Computing, 2008. WGEC '08. Second International Conference on*, pages 128–131.