

Comparing generalized additive neural networks with multilayer perceptrons

J.C. Goosen

Comparing generalized additive neural networks with multilayer perceptrons

Johannes Christiaan Goosen

B.Sc. (North-West University, Potchefstroom Campus)

B.Sc.Hons. (North-West University, Potchefstroom Campus)



Dissertation submitted to the School of Computer, Statistical and Mathematical Sciences at the Potchefstroom Campus of the North-West University in partial fulfilment of the requirements for the degree Magister Scientiae in Computer Science.

Supervisor: Dr. J.V. du Toit

Potchefstroom

May, 2011

Acknowledgements

The completion of the requirements for a Master of Science in Computer Science degree has been a challenge and a privilege for me. During the course of this study, I have experienced good times and some of the worst times of my life. I would like to thank my family and friends who shared the good times with me and supported me through the difficult ones. A special thanks to my mother, sister and brother-in-law, who always encouraged me to do my best. To my father, thank you for believing in me and teaching me to believe in myself: You were and always will be in my thoughts.

I would like to thank my supervisor, Dr. Tiny du Toit, who helped and guided me through this study, always with a smile.

I am grateful for the opportunity that I had to complete this degree as a full-time student; this would not have been possible without the Centre of Excellence bursary from Telkom. I also want to extend my appreciation to SAS Institute Inc. for providing SAS® software with which the results in this dissertation were computed.

Finally, all honour and gratitude, to my Heavenly Father who gave me strength when I was weak, lifted me when I was down and blessed me with the opportunity and guidance to finish my Master of Science degree.

Abstract

In this dissertation, generalized additive neural networks (GANNs) and multilayer perceptrons (MLPs) are studied and compared as prediction techniques. MLPs are the most widely used type of artificial neural network (ANN), but are considered black boxes with regard to interpretability. There is currently no simple a priori method to determine the number of hidden neurons in each of the hidden layers of ANNs. Guidelines exist that are either heuristic or based on simulations that are derived from limited experiments. A modified version of the neural network construction with cross-validation samples (N2C2S) algorithm is therefore implemented and utilized to construct good MLP models. This algorithm enables the comparison with GANN models. GANNs are a relatively new type of ANN, based on the generalized additive model. The architecture of a GANN is less complex compared to MLPs and results can be interpreted with a graphical method, called the *partial residual plot*. A GANN consists of an input layer where each of the input nodes has its own MLP with one hidden layer. Originally, GANNs were constructed by interpreting partial residual plots. This method is time consuming and subjective, which may lead to the creation of suboptimal models. Consequently, an automated construction algorithm for GANNs was created and implemented in the SAS® statistical language. This system was called *AutoGANN* and is used to create good GANN models.

A number of experiments are conducted on five publicly available data sets to gain insight into the similarities and differences between GANN and MLP models. The data sets include regression and classification tasks. In-sample model selection with the SBC model selection criterion and out-of-sample model selection with the average validation error as model selection criterion are performed. The models created are compared in terms of predictive accuracy, model complexity, comprehensibility, ease of construction and utility.

The results show that the choice of model is highly dependent on the problem, as no single model always outperforms the other in terms of predictive accuracy. GANNs may be suggested for problems where interpretability of the results is important. The time taken to construct good MLP models by the modified N2C2S algorithm may be shorter than the time to build good GANN models by the automated construction algorithm.

Keywords: ANN, artificial neural network, AutoGANN, GANN, generalized additive neural network, in-sample model selection, MLP, multilayer perceptron, N2C2S algorithm, out-of-sample model selection, prediction, predictive modelling, SBC, Schwarz information criterion.

In hierdie verhandeling word veralgemeende additiewe neurale netwerke (VANN'e) en multilaag-perseptrone (MLP'e) as voorspellingstegnieke bestudeer en vergelyk. MLP'e is die mees algemeen gebruikte tipe kunsmatige neurale netwerk (KNN), maar word as ondeursigtig beskou met betrekking tot interpreteerbaarheid. Tans is daar geen eenvoudige voor-data-insamelingsmetode om die aantal versteekte neurone in elk van die versteekte lae van KNN'e te bepaal nie. Riglyne bestaan wat óf heuristies van aard is, óf op simulasië-afleidings van beperkte eksperimente gebaseer is. 'n Aangepaste weergawe van die neurale netwerk konstruksie met kruis-validasie steekproewe (N2K2S)-algoritme is dus geïmplementeer en gebruik om goeie MLP-modelle te bou. Hierdie algoritme maak die vergelyking met VANN-modelle moontlik. VANN'e is 'n relatief nuwe tipe KNN wat op die veralgemeende additiewe model gebaseer is. Die argitektuur van 'n VANN is minder kompleks in vergelyking met MLP'e en resultate kan geïnterpreteer word met 'n grafiese metode, genaamd die *parsiële residu-grafiek*. 'n VANN bestaan uit 'n invoerlaag waar elk van die invoernodes sy eie MLP met een versteekte laag het. Oorspronklik was VANN'e gebou deur die interpretasie van parsiële residu-grafieke. Hierdie metode is tydrowend en subjektief, wat kan lei tot die skepping van suboptimale modelle. Gevolglik is 'n outomatiese konstruksie-algoritme vir VANN'e geskep en geïmplementeer in die SAS® statistiese taal. Hierdie stelsel is *AutoGANN* genoem en word gebruik om goeie VANN-modelle te skep.

'n Aantal eksperimente is op vyf vrylik beskikbare datastelle uitgevoer om insig te verkry oor die ooreenkomste en verskille tussen VANN- en MLP-modelle. Die datastelle sluit regressie- en klassifikasietake in. In-steekproefmodel-seleksie met die SBC-model-seleksiëkriterium en buite-steekproefmodel-seleksie met die gemiddelde valideringsfout as model-seleksiëkriterium word uitgevoer. Die modelle wat geskep is, word vergelyk in terme van voorspellende akkuraatheid, modelkompleksiteit, verstaanbaarheid, gemak van konstruksie en nut.

Die resultate toon dat die keuse van die model baie afhanklik van die probleem is aangesien daar geen enkele model is wat altyd beter as die ander is in terme van voorspellingsakkuraatheid nie. VANN'e kan voorgestel word vir probleme waar verstaanbaarheid van die resultate belangrik is. Die tyd wat dit neem om goeie MLP-modelle te bou deur die veranderde N2K2S-algoritme kan korter wees as die tyd wat dit neem om goeie VANN-modelle te bou met die outomatiese konstruksie-algoritme.

Sleutelwoorde: AutoGANN, buite-steekproefmodel-seleksie, in-steekproefmodel-seleksie, KNN, kunsmatige neurale netwerk, MLP, multilaag perseptron, N2K2S-algoritme, SBC, Schwarz-inligtingskriterium, VANN, veralgemeende additiewe neurale netwerk, voorspelling, voorspellingsmodellering.

Contents

1	Introduction	1
1.1	Problem statement	3
1.2	Method of work	4
1.3	Outline of dissertation	4
2	Artificial neural networks	6
2.1	History	7
2.2	Biological inspiration	8
2.3	Neuron model architecture	10
2.3.1	Single-input neuron	10
2.3.2	Multiple-input neuron	13
2.3.3	The perceptron	13
2.3.4	A layer of neurons	17
2.4	Multilayer perceptrons	18
2.5	Artificial neural network learning	21
2.5.1	The perceptron learning rule	22
2.5.2	The backpropagation algorithm	27
2.6	Multilayer perceptron construction	31
2.6.1	The N2C2S algorithm	33
2.6.2	The modified N2C2S algorithm	35
2.6.3	Implementation of the modified N2C2S algorithm	35
2.6.4	Example	38
2.7	Conclusion	39
3	Generalized additive neural networks	40
3.1	Smoothers	41
3.1.1	Scatterplot smoothing	43
3.1.2	The running-mean smoother	44
3.1.3	Smoothers for multiple predictors	45

3.1.4	The bias-variance trade-off	45
3.2	Additive models	47
3.2.1	Multiple regression and linear models	47
3.2.2	Additive models defined	48
3.2.3	Fitting additive models	49
3.2.4	Generalized additive models defined	50
3.3	Generalized additive neural network architecture	51
3.4	The interactive construction methodology	52
3.4.1	Example	54
3.5	The automated construction methodology	58
3.5.1	Definition of terms	58
3.5.2	Model selection	60
3.5.3	The automated construction algorithm	64
3.5.4	Implementation of the automated construction algorithm	69
3.5.5	Example	77
3.6	Conclusion	79
4	Experimental Results	81
4.1	Experimental design	82
4.1.1	GANN experiments	83
4.1.2	MLP experiments	83
4.1.3	Experiment identification	84
4.2	The Adult data set	84
4.2.1	GANN results	85
4.2.2	MLP results	88
4.2.3	Comparison of MLP and GANN results	90
4.3	The Boston Housing data set	90
4.3.1	GANN results	91
4.3.2	MLP results	94
4.3.3	Comparison of MLP and GANN results	95
4.4	The Ozone data set	96
4.4.1	GANN results	97
4.4.2	MLP results	99
4.4.3	Comparison of MLP and GANN results	101
4.5	The SO ₄ data set	102
4.5.1	GANN results	102
4.5.2	MLP results	104
4.5.3	Comparison of MLP and GANN results	106

4.6	The Spambase data set	106
4.6.1	GANN results	107
4.6.2	MLP results	110
4.6.3	Comparison of MLP and GANN results	112
4.7	Conclusion	113
5	Comparative discussion on MLPs and GANNs	114
5.1	Predictive accuracy	114
5.2	Model complexity	117
5.3	Comprehensibility	119
5.4	Ease of construction	119
5.5	Utility	119
5.6	Conclusion	120
6	Conclusion	121
6.1	Summary of findings	121
6.2	Summary of contributions	122
6.3	Suggestions for future work	123
6.4	Conclusion	123
A	MLP construction program code	124
B	MLP brute force method results	147
	Bibliography	159

“As a general rule the most successful man in life is the man who has the best information.”

Benjamin Disraeli

1

Introduction

Currently, the amount of raw data in the world can be overwhelming for us humans (Witten and Frank, 2005). We cannot make sense or process all of this data to obtain useful information without assistance. This is where the incredible computing power of the modern day computer can be helpful. Computers may not be as complex as the human brain, but when it comes to raw computing power, they can do mathematics much faster than humans. This is one reason why statistical models are implemented in computer programs. Even with the present computing power, the usual statistical techniques that are used to gather information from data may not be efficient enough to recognize complex patterns and relationships from large amounts of data. Fortunately, there is a way in which modern day computing power can be used to learn, and consequently obtain useful information and discover useful relationships in the data. Artificial neural networks (ANNs) are statistical models that can learn and generalize from data. One of the ANN's best known features is that it is able to recognize complex patterns in the data. This is useful in fields where prediction is the objective. ANNs are already successfully used in many real-world applications where vast amounts of data are used to obtain useful information.

ANNs are popular, since they have been proven to be successful in many prediction and decision-support applications (Berry and Linoff, 1997). They form a class that consists of general-purpose tools that are very powerful and can be applied to clustering, prediction and classification with relatively ease. A broad range of industries have applied ANNs, which span from number recognition on checks, engine failure rate prediction, financial series prediction, medical diagnosis and identifying groups of valuable customers to identifying credit card fraud, to name a few.

People are good at generalizing from experience, but computers generally excel at performing explicit in-

structions over and again. ANNs are appealing, since they overcome this gap by simulating the human brain's neural connections on a digital computer. They mimic the ability of humans to learn from experience with their ability to learn from data and to generalize when used in well-defined domains. It is this ability that makes ANNs useful for prediction and exciting for research with the future promise of new and better results.

However, there is a drawback. ANNs are considered black boxes with mysterious internal workings. This is as a result of the internal weights that are distributed throughout the network as the result of training an ANN. These weights are not easily understandable, but more and more advanced techniques for examining ANNs help in providing some explanation. ANNs do, however, have business value, which is in many instances more important than understandability.

The history of ANNs in the chronological order of computer science is interesting. In the 1940s, before digital computers really existed, the original work on how neurons function was done. Warren McCulloch, a neurophysiologist, and Walter Pits, a logician, needed a simple model to explain the workings of biological neurons in 1943. They tried to understand the brain's anatomy, but this model turned out to provide a new way of solving certain problems that do not fall in the realm of neurobiology.

Models that are based on the work of McCulloch and Pits, called *perceptrons*, were implemented by computer scientists when digital computers first became available in the 1950s. These early networks solved, for example, the problem of how to balance a broom standing upright on a moving cart. This was done by controlling the motion of the cart. The cart learnt to move to the left if the broom started to fall to the left in order to keep it upright. There were some limited successes in the laboratory using perceptrons, but for general problem-solving, the results were disappointing.

The fact that the most powerful computers of that era were less powerful than today's inexpensive desktop computers is one reason for the limited usefulness of the early ANNs. Seymour Papert and Marvin Minsky were researchers at the Massachusetts Institute of Technology and showed in 1969 that these simple ANNs had theoretical deficiencies that also contributed to their limited usefulness. Research on ANN implementations on computers slowed down drastically during the 1970s as a result of these deficiencies. Then, in 1982, the theoretical pitfalls of ANNs were overcome by a new way of training, called *backpropagation*, that was invented by John Hopfield. This development helped to foster renewed interest in ANN research, which shifted from the labs into the commercial world throughout the 1980s. Since then, ANNs have been applied to virtually every industry to solve both operational and prediction problems.

Statisticians were extending the capabilities of statistical models by taking advantage of computers at the same time that ANNs were developed as a model for biological activity. Logistic regression is a technique that proved especially useful for understanding complex functions of many variables. Logistic regression, like linear regression, attempts to fit a curve to observed data. A function called the *logistic* or *sigmoid* function is, however, used instead of a line. ANNs can be used to represent logistic regression and even the more familiar linear regression. Statistical concepts like distribution, likelihoods and probability among others can, in fact, be used to explain the entire theory of ANNs.

As a result of the convergence of several factors, ANNs became more popular in the 1980s. Firstly, the

availability of computer power improved, particularly where data was available, like in the business community. Secondly, the realisation that ANNs are closely related to known statistical methods made analysts more comfortable with these models. Thirdly, since operational systems in most companies had already been automated, there were relevant data. Fourthly, building useful applications to help people became more important than building artificial people. The utility of ANNs has been proven and as a result they are, and will continue to be, popular for prediction and to encourage further research that will result in even more powerful ANNs in the future.

In this dissertation, two different types of ANNs, called *multilayer perceptrons* and *generalized additive neural networks*, are compared. The problem statement of this study is presented in Section 1.1, followed by the method of work in Section 1.2 and finally, an outline of this dissertation is given in Section 1.3.

1.1 Problem statement

Generalized additive neural networks (GANNs) (Potts, 1999) are a relatively new architecture, based on the generalized additive model (Hastie and Tibshirani, 1986; Wood, 2006). The structure of a GANN is less complex if compared to the most common type of neural network, the multilayer perceptron (MLP) (Ripley, 1996). A GANN consists of an input layer where each of the input nodes has its own MLP with one hidden layer. The latter is connected to the output layer. Currently, there is no simple method to determine the number of hidden neurons in each of the hidden layers. Guidelines exist that are either heuristic or based on simulations that are derived from limited experiments (Zhang, Patuwo and Hu, 1998). Originally, GANNs were constructed by interpreting partial residual plots (Larsen and McCleary, 1972; Ezekiel, 1924; Berk and Booth, 1995). This method is time consuming and subjective, which may lead to the creation of suboptimal models. Consequently, Du Toit (2006) created an automated construction algorithm for GANNs and implemented it in the SAS® statistical language. The system was named *AutoGANN*.

The automated construction algorithm organizes the GANN models into a search tree and performs a best-first search to identify the best model. To speed up the process, a heuristically chosen GANN model is utilized as the starting point. During each iteration of the algorithm, the best GANN model that is based on an objective model selection criterion is identified for expansion. This model is then grown and pruned. While searching for the best model, no human intervention is needed. This process continues until the search space is exhausted or a predetermined time has passed.

The MLP is the most popular and widely used type of neural network (Zhang et al., 1998). MLPs are used in a variety of applications, especially in prediction, because of their inherent capability of subjective input-output mapping. The inputs of an MLP that is used for explanatory forecasting problems are usually independent variables and thus the MLP is functionally equivalent to a nonlinear regression model. For time series forecasting problems, the inputs are typically the past observations and the output is the future value of the data series, thus the MLP is equivalent to a nonlinear autoregressive model.

An MLP consists of two or more layers (Figure 2.13) (Hagan, Demuth and Beale, 1996). The first or the

lowest layer is known as the first hidden layer and this is where external information is received. The last or the highest layer is the output layer where the problem solution is obtained. Between the first hidden layer and the output layer, there may be more hidden layers. Each layer contains a number of neurons. The neurons in adjacent layers are fully connected from a lower layer to a higher layer. With the default constructing method for MLPs, the number of hidden layers and neurons in the hidden layers are manually altered after each session in an attempt to find a better architecture (Zhang et al., 1998).

In this study, GANNs and their construction by the AutoGANN system will be compared to MLPs and the neural network construction with cross-validation samples (N2C2S) construction method (Setiono, 2001) on five publicly available data sets. A modified version of the N2C2S algorithm will be utilized to enable a comparison with the AutoGANN system. A similar comparison was done by Campher (2008), in which GANNs were compared to decision trees and alternating conditional expectations. When comparing the two types of neural networks, consideration will be given to the following:

- The predictive accuracy of the neural networks
- The model complexity of the two types of neural networks
- The comprehensibility of the resulting network, i.e. is the network considered to be a black box?
- The ease in constructing the best neural network model
- The utility of the two types of neural networks and the construction methods that are used to build the best model

1.2 Method of work

In order to obtain a better understanding of these two types of neural networks, a literature study on MLPs and GANNs is performed. The literature study is comprehensive and does not only contain information about the neural network models itself, but also about the methods that are used to construct the architectures. The next step is to develop a program to search for good MLP models on the different data sets. The data sets that are utilized, are the Adult data set (Frank and Asuncion, 2010), Boston Housing data set (Frank and Asuncion, 2010), Ozone data set (Breiman and Friedman, 1985), SO₄ data set (Xiang, 2001) and the Spambase data set (Frank and Asuncion, 2010). Experiments are then conducted to obtain results that can be compared. A number of different experiments will be performed to get a broad perspective on the results. These experiments will include in-sample model selection and out-of-sample model selection. The results are then finally compared with regard to different aspects in order to reach meaningful conclusions.

1.3 Outline of dissertation

In Chapter 2, a short history of artificial neural networks (ANNs) is presented. ANNs are based on biological neural networks and this biological inspiration is examined. The architecture of the artificial neuron model is

then discussed. A single-input neuron and a multiple-input neuron are considered. One of the first ANNs, called the *perceptron*, is discussed, followed by a layer of neurons. The multilayer perceptron (MLP) architecture is considered next. Learning of ANNs is then discussed. Firstly, the perceptron learning rule that is used to train a perceptron network is regarded, followed by the backpropagation algorithm that can be used to train MLPs. The construction of MLPs is discussed next. The N2C2S algorithm is examined, followed by a modified version of the algorithm. This altered version of the N2C2S algorithm was created to enable the comparison with the automated construction algorithm for generalized additive neural networks. Finally, the implementation of the modified N2C2S algorithm which is used in this study to create good MLP models, is explained.

The generalized additive neural network (GANN) which is the neural network implementation of a generalized additive model (GAM), is discussed in Chapter 3. Smoothing, which forms the basis of estimating additive models with the backfitting algorithm, is discussed. Scatterplot smoothing and the running-mean smoother are regarded, as well as smoothers for multiple predictors. Next, the bias-variance trade-off is explained to determine the value of the smoothing parameter. As an introduction to additive models, linear models and multiple regression models are discussed. Additive models are consequently defined, followed by the estimation of these models. Then GAMs are considered, which lead to the GANN architecture. In addition, the interactive- and automated construction algorithms for GANNs are presented. Improvements to the automated construction algorithm are explained and finally the implementation of this algorithm is discussed.

In Chapter 4, the experimental results of the comparison between the GANN and MLP models are presented. The experimental design, which include multiple experiments involving these two types of models, is explained. Then the experiments that were conducted on five publicly available data sets are presented. First, the experiments conducted on the Adult data set are considered, followed by those on the Boston Housing data set, the Ozone data set, the SO_4 data set and finally the Spambase data set. For each data set, the experiments are discussed as follows: First, the data set is introduced, followed by the GANN experiments that were performed by the AutoGANN system and a discussion of the experimental results that were obtained. Then the MLP experiments that were performed by the modified N2C2S algorithm and the brute force method are considered, followed by a discussion of these results. The brute force method is applied to gain more insight into the results that were obtained by the modified N2C2S algorithm. Finally, a comparison between the GANN and MLP experimental results is presented.

The experimental results that were obtained, are discussed on a higher level in Chapter 5 in terms of the predictive accuracies, complexity, comprehensibility, and ease of construction of the models, as well as the utility of both the models and the construction methods.

In Chapter 6, a summary of the findings in this study is presented, followed by a summary of the contributions of the study. Finally, some suggestions for future work are made.

“The beginning of knowledge is the discovery of something we do not understand.”

Frank Herbert

2

Artificial neural networks

The human brain consists of a neural network that has about 10^{11} neurons which are highly interconnected (Hagan et al., 1996). This network helps a person to read, breathe, move and think. A biological neuron is a rich assembly of tissue and chemistry which is as complex as a microprocessor. Persons are born with some of their neural network structure and other parts are established by experience.

Scientists have only begun to understand biological neural networks. All the biological neural functions, including memory, are stored in the connections between neurons and in the neurons itself. The process where new connections are made between neurons and where old connections are modified, is known as the process of learning. Even with the current basic understanding of biological neural networks, it is possible to create an artificial neural network (ANN) that can be trained to serve a useful purpose.

The artificial neurons that are used, are extremely simple abstractions of biological neurons. These artificial neurons can be implemented as part of a program or silicon circuits. Although ANNs can be trained to perform useful functions, they do not have a fraction of the power of the human brain.

Currently, ANNs are considered to be powerful tools that are used by researchers and practitioners in the field of prediction. Research have also shown that ANNs have powerful pattern classification and pattern recognition capabilities (Zhang et al., 1998).

ANNs are data-driven, self-adaptive models that learn from examples and are able to capture subtle functional relationships among the data, even if the underlying relationships are unknown or difficult to describe. ANNs are consequently well suited for problems that have enough data or observations and where the solutions require knowledge that is difficult to specify. One of the most popular and widely used ANNs is the multilayer

perceptron (MLP).

In Section 2.1, a short history of ANNs will be presented, followed by the biological inspiration for ANNs in Section 2.2. The artificial neuron model architecture will then be discussed in Section 2.3, followed by the multilayer perceptron in Section 2.4. In Section 2.5, artificial neural network learning will be considered. Construction of a multilayer perceptron will be discussed in Section 2.6. Finally, some conclusions are presented in Section 2.7.

2.1 History

In order for a technology to advance, at least two components are needed: concept and implementation (Hagan et al., 1996). The history of the heart is a good example of how a different concept changed the technology. The heart was initially thought to be a source of heat or the centre of the soul, but in the 17th century, medical practitioners gained the concept that the heart's function is to pump blood in order for the blood to circulate in the body. Experiments were then designed to test the pumping action of the heart. These experiments inspired the modern day view of the circulatory system of the body. However, concepts are not sufficient for a technology to develop if it is not able to be implemented. A good example of this statement is the computer-aided tomography (CAT) scans. The mathematics that were necessary to reconstruct the images of a CAT scan were known for many years before sufficient high-speed computers and effective algorithms made it possible to implement the CAT scan system. ANNs have also progressed through new concepts and implementation developments. However, the advancements made in ANNs seem to have occurred in bursts rather than following a steady development.

The interdisciplinary work in physics, psychology and neurophysiology, done by scientists such as Herman von Helmholtz, Ernst Mach and Ivan Pavlov from the late 17th century to the early 20th century, form some of the background work for the field of ANNs. The work consisted mostly of general theories on learning, vision and conditioning. Mathematical models of artificial neurons were not included in this work.

The modern view of ANNs commenced when Warren McCulloch and Walter Pitts proposed a model of artificial neurons (McCulloch and Pitts, 1943). This model was based on the human brain, where each neuron is connected to other neurons to form a network. They proposed that the artificial neuron could either be in an “on” or “off” state and that the activation switch would occur in response to stimulation by a certain number of neighbouring neurons. An activation switch is a mechanism that controls when the neuron is in an “on” or “off” state. ANNs could also have the ability to learn. ANN learning is achieved by applying a set of rules, known collectively as a learning rule, to update the connections between neurons. In 1949, Donald Hebb proposed a simple learning rule, now known as the *Hebbian learning rule* (Hebb, 1949). He suggested that neurons that are in the same state have a stronger relationship to each other, while neurons in an opposite state will have a weaker relationship to each other. This learning rule adjusts the connections to a better representation of the relationship between neurons.

The first neural network computer, called the *SNARC* (Stochastic Neural Analog Reinforcement Calculator),

was built in 1950 by Marvin Minsky and Dean Edmonds (Russell and Norvig, 2010). They used an automatic pilot mechanism from a B-24 bomber and 3000 vacuum tubes to simulate a 40 neuron network. Frank Rosenblatt proposed an artificial neuron that would classify its inputs into one of two categories (Rosenblatt, 1958). This artificial neuron was called a *perceptron*. Rosenblatt used these neurons to build the first neural network that was used in a practical application. He showed that this network could be used for pattern recognition. Frank Rosenblatt also introduced the perceptron learning rule that is used for training the perceptron neurons (Rosenblatt, 1962). The perceptron and the perceptron learning rule will be discussed in Section 2.3.3 and Section 2.5.1 respectively.

In 1969, Marvin Minsky and Seymour Papert published a book entitled *Perceptrons*, in which they stated that the problem-solving capabilities of single-layer neural networks were limited to linearly separable problems (Minsky and Papert, 1969). This book, and the lack of powerful digital computers at the time, caused many people to stop research in the field of artificial neural networks (Hagan et al., 1996).

Between the 1960s and the 1980s there were very little progress in the field of ANNs and general interest in neural networks declined heavily (Hagan et al., 1996). Fortunately, in the 1980s, new advances were made in the field of ANNs, more powerful computers could be built and, as a result, more researchers gained interest in this field. One of the new developments that was responsible for ANNs getting the attention of researchers was the invention of the backpropagation algorithm (Rumelhart and McClelland, 1986). With the backpropagation algorithm, a network consisting of multiple layers of perceptrons, called an *multilayer perceptron* (MLP), could be trained. This learning rule was the answer to the problems of perceptron networks that were raised by Minsky and Papert (1969) first. MLPs and the backpropagation algorithm will be discussed in more detail in Section 2.4 and Section 2.5.2 respectively. Another development that attracted attention to ANNs was the Hopfield network that could be used as an associative memory (Hopfield, 1982).

The field of ANNs has developed substantially since McCulloch and Pitts first introduced the idea and today ANNs are used in a variety of disciplines which include, among others, aerospace, automotive, banking, defence, electronics, entertainment, financial, insurance, manufacturing, medical, oil and gas (Hagan et al., 1996). When McCulloch and Pitts (1943) introduced the ANN, it was based on the human brain. In the next section the biological inspiration for ANNs will be discussed.

2.2 Biological inspiration

The human brain consists of a highly interconnected neural network. This neural network has about 10 billion neurons and 60 trillion connections (Negnevitsky, 2005). A biological neuron has a switching speed (the speed at which the output changes in response to the inputs) of 10^{-3} seconds, whereas an electrical circuit has a switching speed of 10^{-9} seconds (Hagan et al., 1996). The electrical circuit is clearly much faster than the biological neuron, but this does not mean that a computer is faster than the human brain. The high connectivity of the human brain's neurons and the fact that the human brain can use multiple neurons at the same time, is the

reason why it can do many tasks much faster when compared to a computer (Hagan et al., 1996; Negnevitsky, 2005).

A schematic representation of a biological neuron that is connected to another one, is shown in Figure 2.1.

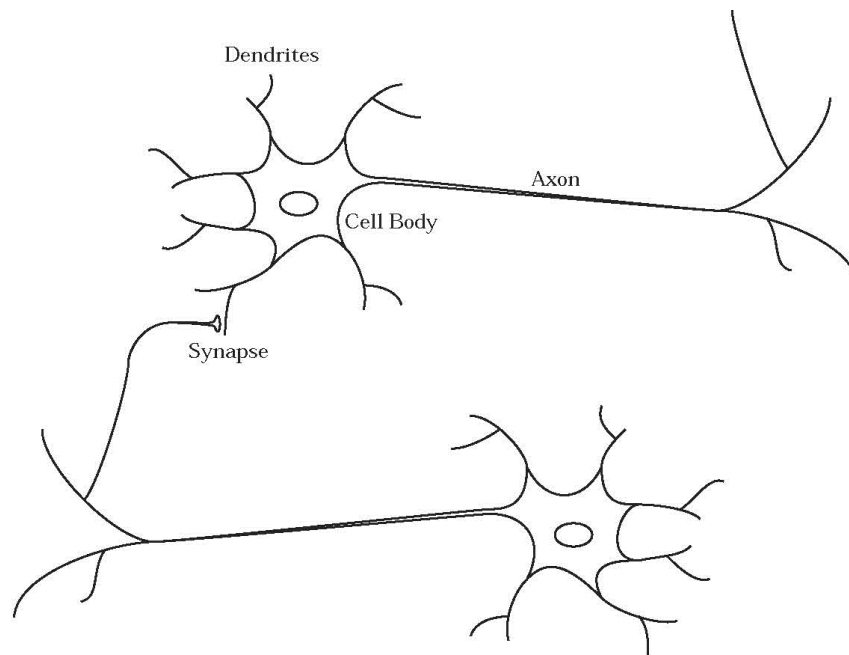


Figure 2.1: Biological neuron

A biological neuron consists of the following principle components: the dendrites, the axon, the cell body (soma) and the synapses (Hagan et al., 1996). The soma receives signals from other neurons via the dendrites. When the soma's threshold is reached, it sends a signal to other neurons through the axon. The connection between neurons, where the axon meets the dendrites, is called a *synapse*. The synapse releases a chemical content, which changes the potential difference of the soma (Negnevitsky, 2005). The function of the neural network is established by the arrangement of its neurons and the strengths of the individual connections between neurons (Hagan et al., 1996). The connection strengths and the arrangement of neurons are determined by a complex chemical process. Some of the neural structure is determined at birth, while other parts are developed through learning. The brain's ability to learn comes from a property of a neural network, called *plasticity* (Negnevitsky, 2005). Plasticity indicates that the neurons are able to make new connections to other neurons and that the connection strengths between neurons may change.

Even though an ANN is not nearly as complex as the brain, there are at least two similarities between them. Firstly, both networks consist of simple building blocks that are highly interconnected and secondly, the function of the network is determined by the connections between neurons (Coppin, 2004).

The biological neuron inspired the creation of artificial neurons which can be combined to form an artificial neural network. In the next section, the neuron model architecture of an artificial neuron is considered.

2.3 Neuron model architecture

In this section, a mathematical model for an artificial neuron will be introduced. First, an artificial neuron that has only one input will be examined. A more complex artificial neuron that has multiple inputs will then be considered. After that, a simple ANN, called a *perceptron*, will be discussed and finally, a layer of neurons will be considered.

2.3.1 Single-input neuron

In a single-input neuron, a scalar input p is multiplied by a scalar weight w (Hagan et al., 1996). This product, wp , is then added to a bias b to form n (n is defined in (2.5)), which is the net input to the activation function f . The activation (transfer) function f produces the final output a . A single-input neuron model is shown in Figure 2.2.

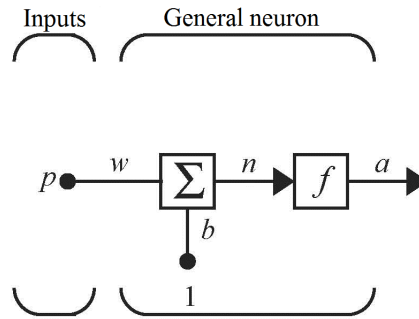


Figure 2.2: Single-input neuron

The output a of the single-input neuron is calculated as follows:

$$a = f(wp + b). \quad (2.1)$$

If, for example, $w = 5$, $p = 3$ and $b = -2.5$, then

$$a = f(5(3) - 2.5) = f(12.5). \quad (2.2)$$

The final output is determined by the activation function f . The activation function is chosen by the designer and a learning rule will adjust the parameters w and b in order for the input/output relationship to meet a specific goal that is set by the learning rule.

This simple artificial neuron can be compared to a biological neuron with regards to the following: The input p is the stimuli from an external source, the weight w can be considered as the strength of the synapse, the summation together with the activation function represent the soma and the output a represents the signal on the axon.

There are different activation functions for different purposes. Next, some of these activation functions will be discussed.

Activation functions

A specific activation function is used to meet some specification of the problem that must be solved by the neuron (Hagan et al., 1996). There are many different activation functions available. In Table 2.1, some of these activation functions are shown (Hagan et al., 1996).









Name	Input/output relation	Figure
Hard-limit	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$	
Symmetrical hard limit	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$	
Linear	$a = n$	
Saturating linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$	
Symmetric saturating linear	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$	
Log-sigmoid	$a = \frac{1}{1+e^{-n}}$	
Hyperbolic tangent sigmoid	$a = \frac{e^n - e^{-n}}{e^n + e^{-n}}$	
Positive linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n$	

Table 2.1: Activation functions

When a neuron is required to classify an input into two distinct classes, a hard-limit activation function can be used. The hard-limit activation function gives an output of 0 if the function input is less than 0, and an output of 1 if the function input is equal to or greater than 0. This activation function is shown in Figure 2.3.

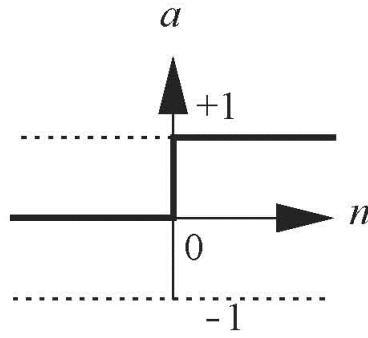


Figure 2.3: Hard-limit activation function

Some problems may need an activation function where the output is the same as the input:

$$a = n. \quad (2.3)$$

For these problems, a linear activation function is used. This activation function is shown in Figure 2.4.

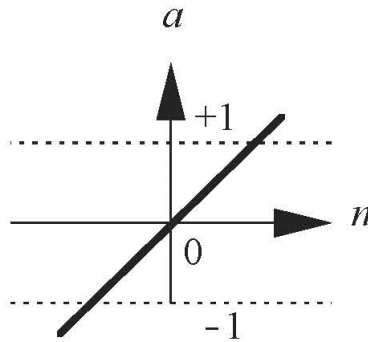


Figure 2.4: Linear activation function

The log-sigmoid activation function produces an output that is mapped between 0 and 1. This output is calculated according to the expression

$$a = \frac{1}{1 + e^{-n}}. \quad (2.4)$$

The log-sigmoid activation function is shown in Figure 2.5.

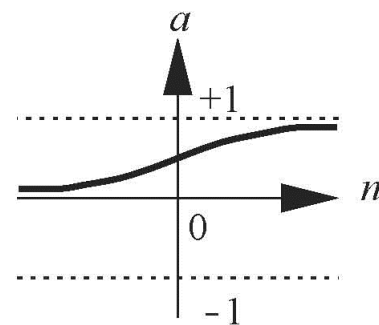


Figure 2.5: Log-sigmoid activation function

The single-input neuron model and some of the activation functions have been considered, but in most real-world problems, there are more than one variable that are used as inputs. In the next section, the multiple-input neuron model will be discussed.

2.3.2 Multiple-input neuron

Generally, a neuron will have more than one input (Hagan et al., 1996). A model of a multiple-input neuron is shown in Figure 2.6.

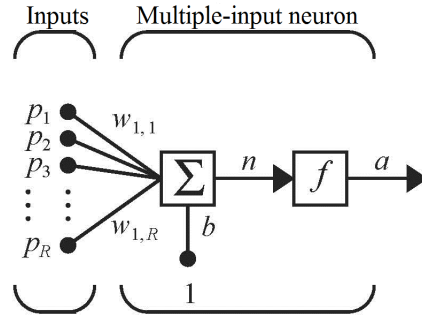


Figure 2.6: Multiple-input neuron

Each of the inputs p_1, p_2, \dots, p_R is multiplied by the corresponding weight, $w_{1,1}, w_{1,2}, \dots, w_{1,R}$, of the weight matrix \mathbf{W} . The notation of the weights can be explained as follows: the weight $w_{1,2}$ represents the connection from the second input to the first neuron. The net input n for the activation function is obtained by adding the bias to the weighted inputs. The net input can be written as

$$n = w_{1,1}p_1 + w_{1,2}p_2 + \dots + w_{1,R}p_R + b. \quad (2.5)$$

In matrix form, the latter expression is written as

$$n = \mathbf{W}\mathbf{p} + b, \quad (2.6)$$

where \mathbf{p} is a vector and, in the case of a single neuron, the matrix \mathbf{W} will have only one row. The output of the multiple-input neuron can thus be written as

$$a = f(\mathbf{W}\mathbf{p} + b). \quad (2.7)$$

One of the first ANNs was called a *perceptron*. This artificial neuron is able to classify multiple inputs into one of two classes. In the next section, the perceptron architecture will be considered.

2.3.3 The perceptron

The perceptron was introduced by Rosenblatt (1958) and is based on the neuron that was proposed by McCulloch and Pitts (1943). The perceptron architecture is a single-layer neural network with a hard-limit activation function (Hagan et al., 1996). Note that Hagan et al. (1996) does not consider the inputs as a layer. A single-neuron perceptron can classify the input vectors into one of two classes. To illustrate this capability, a two-input single-neuron perceptron will be considered. Figure 2.7 shows a single-neuron perceptron with two inputs.

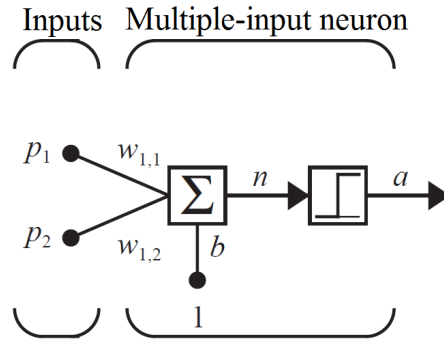


Figure 2.7: Single-neuron perceptron with two inputs

If $w_{1,1}$ and $w_{1,2}$ is, for example, -1 and 1 respectively, then the output will be defined as

$$a = \text{hardlim}\left(\begin{bmatrix} -1 & 1 \end{bmatrix} \mathbf{p} + b\right). \quad (2.8)$$

In this example, the weight matrix \mathbf{W} is a single row vector and if the product of the weight vector and the input vector \mathbf{p} is equal to or greater than $-b$, then the output a will be 1. The output will be 0 if the product of the input vector and weight vector is less than $-b$. The input space is now divided into two parts. Figure 2.8 shows the decision boundary where $b = -1$. The dotted line in the figure represents all the points where the net input

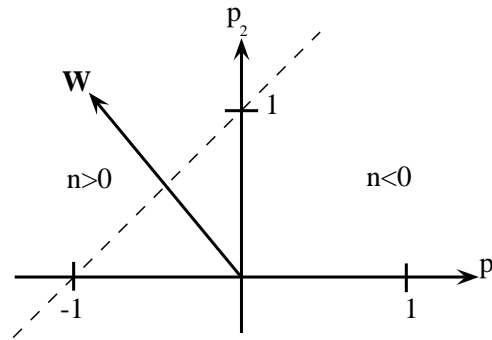


Figure 2.8: Decision boundary

n is equal to 0:

$$n = \begin{bmatrix} -1 & 1 \end{bmatrix} \mathbf{p} - 1 = 0. \quad (2.9)$$

The network output will be 1 for all the input vectors that are on the left side of the boundary line and 0 for all other input vectors. The decision boundary is determined by

$$\mathbf{W}\mathbf{p} + b = 0. \quad (2.10)$$

For a single-layer perceptron, the boundary must be linear and thus the single-layer perceptron's pattern recognition capabilities are limited to linearly separable problems. As a result, the decision boundary line must separate the input space into two areas where each area represents an output class. A decision boundary of a problem that is linearly separable is shown in Figure 2.9. In this figure, all the black dots fall into one class and the white dot falls in the other class. The dotted line separates the two classes; each point on the right side of the dotted line will represent one class and each point on the left side will represent the other class.

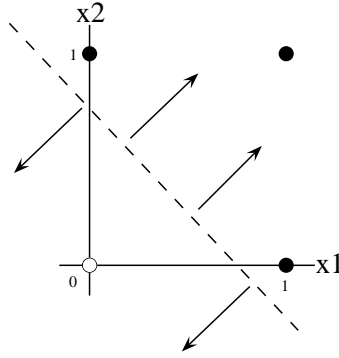


Figure 2.9: Linearly separable problem

Figure 2.10 represents a problem that is nonlinear. The black dots represent one class and the white dots the other class. As seen from this figure, it is impossible to separate these two classes by using a straight line.

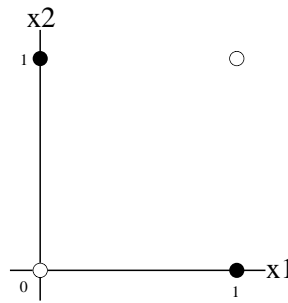


Figure 2.10: Nonlinearly separable problem

In the next example, a single-neuron perceptron will be used to classify a car into one of two classes: a family sedan (represented by 1) or a sports sedan (represented by 0). Three attributes will describe each car and as a result, the input vector will be three-dimensional. The perceptron will thus be defined as

$$a = \text{hardlim}\left(\begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} + b\right). \quad (2.11)$$

The first input p_1 will represent the drive method of the car, -1 for four-wheel drive (4x4) or 1 for two-wheel drive (4x2). The second input p_2 will indicate the car's engine power, -1 for cars with 120 kilowatt of power or more and 1 for cars with less than 120 kilowatt of power. The final input p_3 will represent the number of doors of the car, 1 for two doors and -1 for four doors.

The two car models that will be tested is a Volkswagen Jetta 1.6 (family sedan) and the BMW 325i (sports sedan). The Jetta is 4x2 driven, has 75 kilowatt of power and four doors. The Jetta's input vector is thus

$$\mathbf{p}_1 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}. \quad (2.12)$$

The BMW is also 4x2, has 160 kilowatt of power and four doors. The BMW's input vector is thus

$$\mathbf{p}_2 = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}. \quad (2.13)$$

The linear boundary that separates these two vectors symmetrically is the p_1, p_3 plane as shown in Figure 2.11.

The decision boundary, which is the p_1, p_3 plane, can be described by the expression

$$p_2 = 0, \quad (2.14)$$

or

$$\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} + 0 = 0, \quad (2.15)$$

since the weight vector must be orthogonal to the decision boundary in the direction of the prototype that is classified as 1.

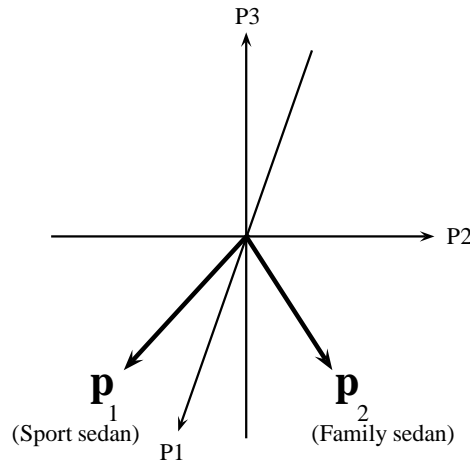


Figure 2.11: Input car vectors

The weight matrix \mathbf{W} is thus $\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$ and the bias b is 0. The latter is 0 because the decision boundary passes through the origin. If the Jetta's specifications are given as the input, then the output will be

$$a = \text{hardlim} \left(\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} + 0 \right) = 1 \quad (\text{family sedan}). \quad (2.16)$$

If the BMW's specifications are given as the input then, the output will be

$$a = \text{hardlim} \left(\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} + 0 \right) = 0 \quad (\text{sports sedan}). \quad (2.17)$$

Next, a new car will be classified by the perceptron, namely the *Audi TT 3.2 quattro*. This car is a sports car, but not a sedan. The Audi is a four-wheel drive car (4x4), has 184 kilowatt of power and two doors. The

Audi's input vector is the following:

$$\mathbf{p}_3 = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}. \quad (2.18)$$

The Audi's input vector is presented to the perceptron and the following output is obtained:

$$a = \text{hardlim} \left(\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} + 0 \right) = 0 \quad (\text{sports sedan}). \quad (2.19)$$

The perceptron classified the Audi as a sports sedan, because it has a closer resemblance to a sports sedan than a family sedan. If the car was closer to a family sedan (for example: four doors, 4x2 driven and less than 100 kilowatt of power), the perceptron would be able to determine it as well.

When many inputs are used, it is difficult to determine the weight matrix and the bias vector, as it is not possible to visualize the decision boundaries. This difficulty is overcome by a learning rule that train perceptron networks to solve classification problems. The perceptron learning rule will be discussed in Section 2.5.1.

In cases where a more complex ANN is needed, a layer of neurons can be used. This concept will be discussed in the next section.

2.3.4 A layer of neurons

In a single-layer neural network that consists of a number of neurons, each input is connected to each neuron. A single-layer neural network which has S neurons and R inputs is shown in Figure 2.12.

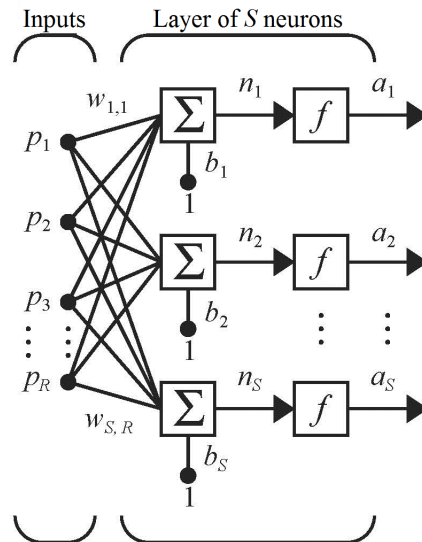


Figure 2.12: Single-layer neural network

The layer consists of the weight matrix \mathbf{W} , the bias vector \mathbf{b} , summation functions, activation functions and the vector \mathbf{a} as the output. Each input in the vector \mathbf{p} is connected through the weight matrix \mathbf{W} to each neuron. It is not unusual for the number of neurons S to differ from the number of inputs R . Each neuron i consists of a summation function, a bias b_i , an activation function f and an output a_i , where i is the neuron number. It is

possible for neurons to have different activation functions. This is accomplished by creating a composite layer of neurons, consisting of two or more single-layer networks in parallel where the neurons in individual layers will have the same activation functions. Thus, all the networks will have the same inputs and each network will give a part of the output.

The weight matrix in a layer of neurons is shown below:

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,R} \\ w_{2,1} & w_{2,2} & \dots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \dots & w_{S,R} \end{bmatrix}. \quad (2.20)$$

The notation that is used by the weight matrix can be explained as follows: $w_{4,3}$ for example, represents the weight connection from the third source to the fourth neuron.

When a single-layer neural network is not powerful enough to perform the task at hand, a multilayer neural network can be used. In the next section multilayer perceptrons will be discussed.

2.4 Multilayer perceptrons

Multilayer perceptrons (MLPs) are neural networks that have two or more layers that consist of one or more neurons in each layer (Rumelhart and McClelland, 1986). The first hidden layer receives the inputs from outside stimulation (Negnevitsky, 2005). The last layer is known as the output layer and is responsible for the final output of the neural network. Between the input and output layer, there can also be one or more hidden layers. The hidden layers' neurons detect patterns from the data. The weights of the neurons represent characteristics of the patterns hidden in the data. The output layer then uses these characteristics to determine the output pattern. Each input is connected to each neuron in the first hidden layer. Each neuron in the first hidden layer is then connected to each neuron in the next hidden layer. Finally, each neuron in the last hidden layer is connected to each output neuron. An MLP is classified as a feedforward network, which indicates that the input values are distributed from the input layer, layer by layer, to the output layer.

In the architecture of an MLP, each layer has a weight matrix \mathbf{W} , a bias vector \mathbf{b} , a net input vector \mathbf{n} and an output vector \mathbf{a} (Hagan et al., 1996). The number of each layer is appended as a superscript to each of these variables to distinguish between the different variables in the different layers. As a result, the weight matrix for the first layer and second layer will be written as \mathbf{W}^1 and \mathbf{W}^2 respectively. Note that Hagan et al. (1996) do not regard the inputs as a separate layer. A three-layer network (Hagan et al., 1996) is shown in Figure 2.13 to illustrate this multilayer notation. The final output \mathbf{a}^3 of this example can be defined as

$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3 \mathbf{f}^2(\mathbf{W}^2 \mathbf{f}^1(\mathbf{W}^1 \mathbf{p} + \mathbf{b}^1) + \mathbf{b}^2) + \mathbf{b}^3). \quad (2.21)$$

As shown in Figure 2.13, there are R inputs and S^k neurons in layer k . The different layers in the network can have a different number of neurons in each layer and even different activation functions. In this figure, the

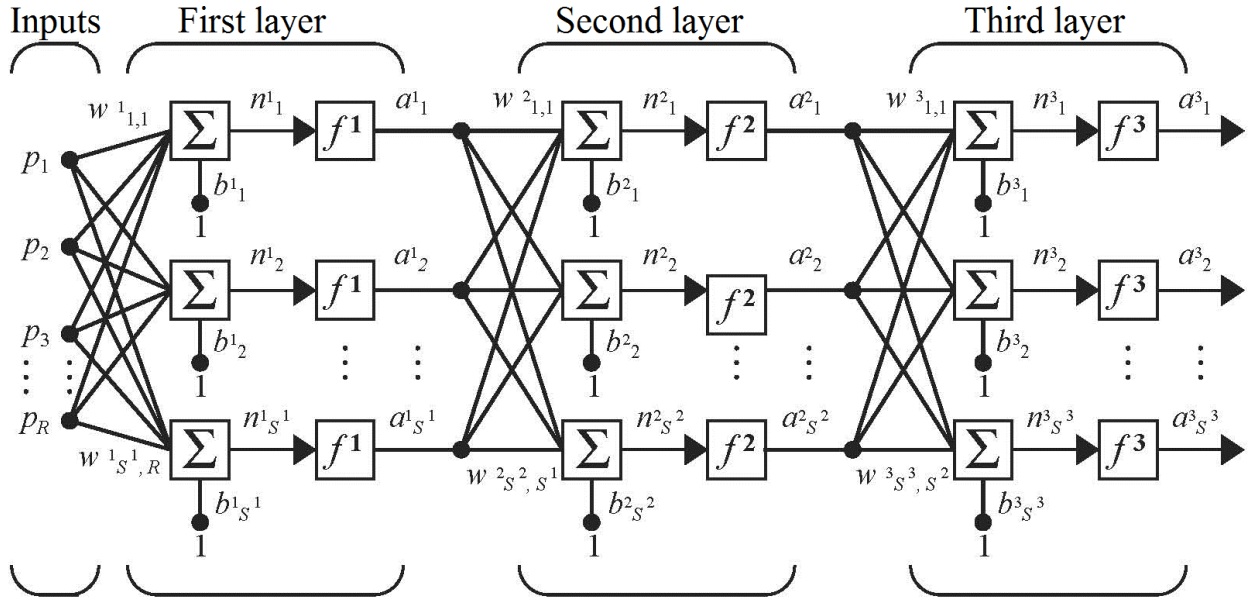


Figure 2.13: Three-layer neural network

first hidden layer is represented by the first layer, the second layer represents a second hidden layer and the third layer is the output layer. For the first hidden layer \mathbf{p} is given as input and the layer output is \mathbf{a}^1 , which in turn is given as input for the second hidden layer. The second hidden layer's output is \mathbf{a}^2 and is given as input for the output layer, which gives the final output \mathbf{a}^3 .

These MLPs are more powerful than single-layer perceptrons, as most functions can be approximated arbitrarily well with a two-layer network that uses a sigmoid activation function in the first hidden layer and a linear activation function in the output layer (Hagan et al., 1996).

When constructing an MLP with supervised learning, the goal is to develop a good model that is trained on a data set where the target is known. This model must then perform well on data that has not been seen before. When training an MLP on a training data set, the more complex the MLP, the more accurate the neural network will be on that data set, but this may lead to overfitting (Murtagh, 1991). The latter occurs when the network is too complex and learns the data from the training data set and performs well on that data, but performs badly on new, unseen data. Another problem with adding extra hidden layers to make the neural network more complex is the additional computing power needed for training that increases exponentially (Negnevitsky, 2005).

The number of neurons in the output (last) layer is determined by the problem specifications. For example, in some cases, if the data set used for training the network consists of one target attribute, then the output layer will have one neuron. For the number of neurons in the hidden layers there are no constant formula for all problems. The number of layers in a network may also differ, but more than two layers (a hidden layer and an output layer) are rarely used. Neurons may or may not contain biases. In many cases, a network will be more powerful when the neurons have biases, as an input of value 0 will result in a neuron output of 0 if there is no bias added. A construction algorithm is thus needed to guide the development of a neural network that will perform satisfactory for a specific problem. In Section 2.6, algorithms for the construction of MLPs will be discussed.

In order to show that an MLP can solve problems that a single-layer network cannot, the exclusive-or (XOR) problem will be considered. This problem was used by Minsky and Papert (1969) to show that a single-layer network is limited to a problem where the categories are linearly separable. The input data set contains the following data points:

$$\left\{ \mathbf{d}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, t_1 = 0 \right\}, \left\{ \mathbf{d}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, t_2 = 1 \right\}, \left\{ \mathbf{d}_3 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, t_3 = 1 \right\} \text{ and } \left\{ \mathbf{d}_4 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, t_4 = 0 \right\} \quad (2.22)$$

where t_i denotes the target values. As shown in Figure 2.14, the XOR problem is not linearly separable and thus a single-layer network would be unable to solve it. There are, however, many different MLPs that can solve the XOR problem, but for this example, a two-layer MLP will be used. This MLP can be seen in Figure 2.15.

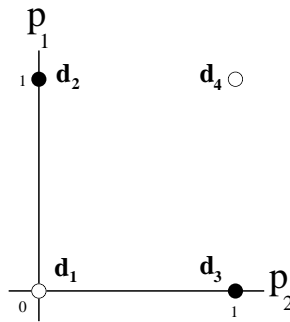


Figure 2.14: XOR problem space

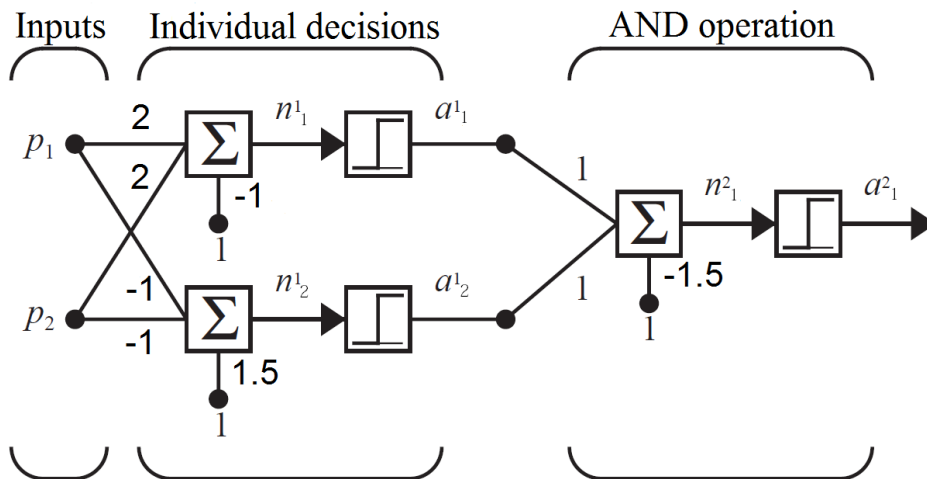


Figure 2.15: Two-layer XOR neural network

The first hidden layer consist of two neurons. Each of the two neurons create a decision boundary, as shown in Figures 2.16 and 2.17. The output layer have one neuron. This neuron combines the two decision boundaries, which then distinguish correctly between the target variable values. For this example, the hard-limit activation function is utilized. The classification is shown in Figure 2.18, where the inputs between the two boundaries will result in an output of 1.

The connections (weights) of ANNs are modified by means of rules that are known as learning rules. In the

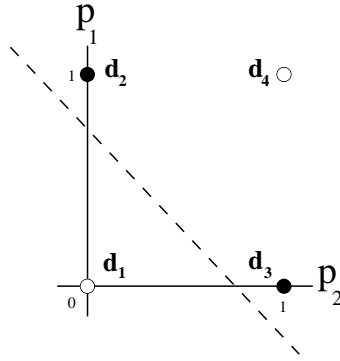


Figure 2.16: Layer 1 - neuron 1

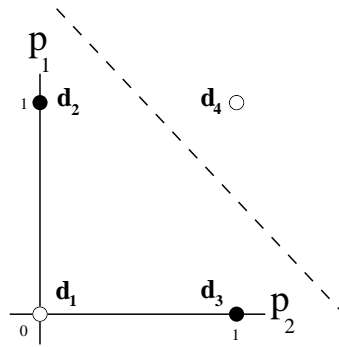


Figure 2.17: Layer 1 - neuron 2

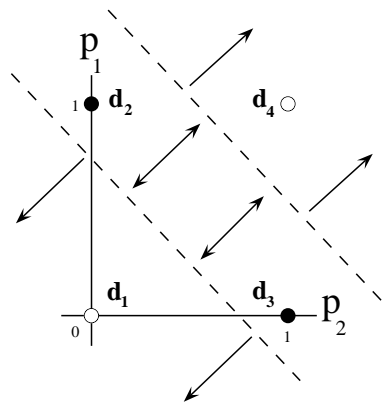


Figure 2.18: Final network output

next section, two learning rules are discussed: one that is used to train a perceptron and the other one to train a multilayer perceptron.

2.5 Artificial neural network learning

A learning rule is an algorithm that modifies the weights and biases of a neural network in order to train it to perform a task (Hagan et al., 1996). A learning rule is thus sometimes called a *training algorithm*. There are three main categories of learning rules:

- Unsupervised learning: With unsupervised learning, there is no target output available. The weights and

biases of the neural network are thus modified only in response to the inputs.

- **Supervised learning:** Supervised learning uses a training data set that contains inputs with the correct target output. The inputs are applied to the neural network and the output of the network is compared to the target output. The learning rule then makes changes to the weights and biases in order for the network output to be more accurate compared to the target output. In this study, supervised learning is performed.
- **Reinforcement learning:** Reinforcement learning works in the same way as supervised learning, except that a target output is not provided. Instead, the algorithm is given a grade that measures the neural network's performance over some succession of inputs.

2.5.1 The perceptron learning rule

The perceptron learning rule falls in the supervised learning category. In order to explain the perceptron learning rule, it would be helpful to be able to reference individual elements of the network output. First, the weight matrix can be denoted as follows:

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,R} \\ w_{2,1} & w_{2,2} & \dots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \dots & w_{S,R} \end{bmatrix}. \quad (2.23)$$

A vector that contains the elements of the i th row of \mathbf{W} can be defined as

$${}_i\mathbf{w} = \begin{bmatrix} w_{i,1} \\ w_{i,2} \\ \vdots \\ w_{i,R} \end{bmatrix}. \quad (2.24)$$

The weight matrix can now be partitioned as follows:

$$\mathbf{W} = \begin{bmatrix} {}_1\mathbf{w}^T \\ {}_2\mathbf{w}^T \\ \vdots \\ {}_S\mathbf{w}^T \end{bmatrix}, \quad (2.25)$$

where ${}_i\mathbf{w}^T$ denotes the transpose of ${}_i\mathbf{w}$. With the partitioned weight matrix, the i th element of the output vector can be written as

$$a_i = \text{hardlim}(n_i) = \text{hardlim}({}_i\mathbf{w}^T \mathbf{p} + b_i). \quad (2.26)$$

Consider a single neuron with two inputs, as shown in Figure 2.19, where the weights and bias will be chosen manually by means of a decision boundary.

The output a of this two-input single-neuron perceptron is determined by

$$a = \text{hardlim}(n) = \text{hardlim}(\mathbf{W}\mathbf{p} + b) = \text{hardlim}({}_1\mathbf{w}^T \mathbf{p} + b) = \text{hardlim}(w_{1,1}p_1 + w_{1,2}p_2 + b). \quad (2.27)$$

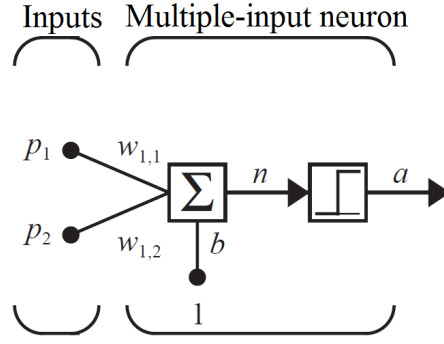


Figure 2.19: Single-neuron perceptron with two inputs

The decision boundary can be written as

$$n = {}_1\mathbf{w}^T \mathbf{p} + b = w_{1,1}p_1 + w_{1,2}p_2 + b = 0. \quad (2.28)$$

Let $w_{1,1} = 1$, $w_{1,2} = 1$ and $b = -1$, then the decision boundary will be

$$n = {}_1\mathbf{w}^T \mathbf{p} + b = w_{1,1}p_1 + w_{1,2}p_2 + b = p_1 + p_2 - 1 = 0. \quad (2.29)$$

The decision boundary defines a line in the input space where the output will be 0 on the one side and 1 on the other side of the line. In order to draw the line, the points where the line intercepts the p_1 and p_2 axes must be found. The p_1 intercept can be found by setting p_2 to 0:

$$p_1 = -\frac{b}{w_{1,1}} = -\frac{-1}{1} = 1. \quad (2.30)$$

The p_2 intercept can be found by setting p_1 to 0:

$$p_2 = -\frac{b}{w_{1,2}} = -\frac{-1}{1} = 1. \quad (2.31)$$

The decision boundary line can now be drawn, as shown in Figure 2.20. According to this figure, the output of the network will be 1 for all inputs that correspond to a point in the shaded area and 0 otherwise.

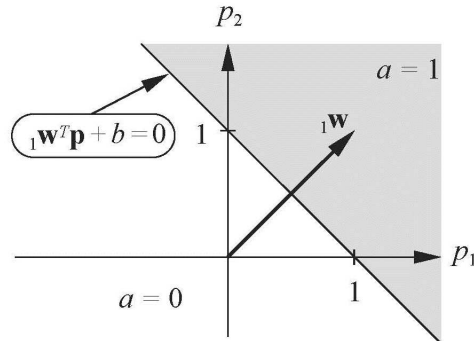


Figure 2.20: Decision boundary

To apply the perceptron learning rule, a data set is required that contains input/output pairs:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \dots, \{\mathbf{p}_Q, \mathbf{t}_Q\}, \quad (2.32)$$

where \mathbf{p}_q is an input and \mathbf{t}_q is the corresponding target output with $q = 1, 2, \dots, Q$. An example data set will be used for illustrating the perceptron learning rule (Hagan et al., 1996):

$$\left\{ \mathbf{p}_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, t_1 = 1 \right\}, \left\{ \mathbf{p}_2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, t_2 = 0 \right\}, \left\{ \mathbf{p}_3 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, t_3 = 0 \right\} \quad (2.33)$$

To simplify the illustration of the learning rule, a single-neuron perceptron without a bias (where bias $b = 0$) will be used, as shown in Figure 2.21.

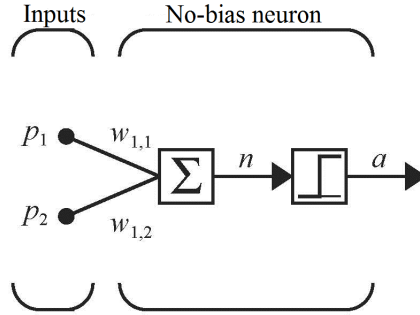


Figure 2.21: Single-neuron perceptron without a bias

The output of this perceptron is thus defined as

$$a = \text{hardlim}(\mathbf{W}\mathbf{p}). \quad (2.34)$$

From the example data set, it is known that there are two variables in the input vector and one target output. As a result, the learning rule only needs to adjust the weight matrix, which in this case consists of two elements. The first step that must be performed is to initialize these two weights with random values:

$${}_1\mathbf{w}^T = \begin{bmatrix} 1.0 & -0.8 \end{bmatrix}. \quad (2.35)$$

The first input vector \mathbf{p}_1 is now applied to the network:

$$a = \text{hardlim}({}_1\mathbf{w}^T \mathbf{p}_1) = \text{hardlim} \left(\begin{bmatrix} 1.0 & -0.8 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right) = \text{hardlim}(-0.6) = 0. \quad (2.36)$$

The target output is 1, but the network gave an output of 0. As shown in Figure 2.22, the initial weight values caused \mathbf{p}_1 to be incorrectly classified by the decision boundary. In this figure, the black dot represents \mathbf{p}_1 with an output of 1. The other two hollow dots represent \mathbf{p}_2 and \mathbf{p}_3 with an output of 0 each. As seen in the figure, the decision boundary does not separate the inputs correctly. Also note that the decision boundary must pass through the origin of the graph, as there is no bias. The weight vector is orthogonal to the decision boundary and, due to this, the decision boundary will shift if the weight vector changes.

The weight vector needs to be adjusted to improve the probability of classifying \mathbf{p}_1 correctly. To do this, \mathbf{p}_1 is added to ${}_1\mathbf{w}$. This results in ${}_1\mathbf{w}$ pointing more in the direction of \mathbf{p}_1 . If this is repeated with \mathbf{p}_1 , then ${}_1\mathbf{w}$ would asymptotically approach the direction of \mathbf{p}_1 . This rule can be described as follows:

$$\text{If } t = 1 \text{ and } a = 0, \text{ then } {}_1\mathbf{w}^{\text{new}} = {}_1\mathbf{w}^{\text{old}} + \mathbf{p}. \quad (2.37)$$

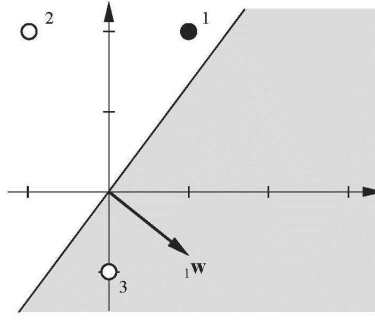


Figure 2.22: Incorrect decision boundary

Applying this rule to the example would result in the following:

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}_1 = \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2.0 \\ 1.2 \end{bmatrix}. \quad (2.38)$$

The resulting decision boundary, after adjusting the weight values, is shown in Figure 2.23. This figure shows how the weight vector changed and, consequently, how the decision boundary shifted.

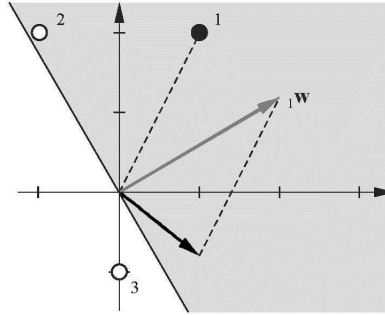


Figure 2.23: First adjusted decision boundary

Input vector \mathbf{p}_2 is now applied to the network:

$$a = \text{hardlim}({}_1\mathbf{w}^T \mathbf{p}_2) = \text{hardlim} \left(\begin{bmatrix} 2.0 & 1.2 \end{bmatrix} \begin{bmatrix} -1 \\ 2 \end{bmatrix} \right) = \text{hardlim}(0.4) = 1. \quad (2.39)$$

The output a is misclassified by the network, as the target associated with \mathbf{p}_2 is 0 and output a is 1. The weight vector now needs to be moved away from the input. This can be done with the following rule:

$$\text{If } t = 0 \text{ and } a = 1, \text{ then } {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}. \quad (2.40)$$

Applying this rule to the example would result in the following:

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}_2 = \begin{bmatrix} 2.0 \\ 1.2 \end{bmatrix} - \begin{bmatrix} -1 \\ 2 \end{bmatrix} = \begin{bmatrix} 3.0 \\ -0.8 \end{bmatrix}. \quad (2.41)$$

The resulting decision boundary, created by adjusting the weight values, is shown in Figure 2.24. In this figure, the decision boundary shifted again as the weight vector changed.

The final input vector in the example data set, \mathbf{p}_3 , is now applied to the network:

$$a = \text{hardlim}({}_1\mathbf{w}^T \mathbf{p}_3) = \text{hardlim} \left(\begin{bmatrix} 3.0 & -0.8 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \end{bmatrix} \right) = \text{hardlim}(0.8) = 1. \quad (2.42)$$

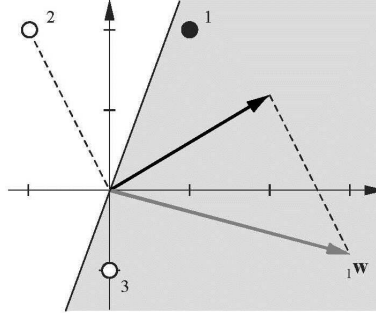


Figure 2.24: Second adjusted decision boundary

The input vector was misclassified again and, consequently, the weights need to be updated. The previous rule also applies to this situation and will be used:

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}_3 = \begin{bmatrix} 3.0 \\ -0.8 \end{bmatrix} - \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 3.0 \\ 0.2 \end{bmatrix}. \quad (2.43)$$

The resulting decision boundary, created by adjusting the weight values, is shown in Figure 2.25. As this figure shows, the network has learnt to classify all three input vectors correctly. The third and final rule is:

$$\text{If } t = a, \text{ then } {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}. \quad (2.44)$$

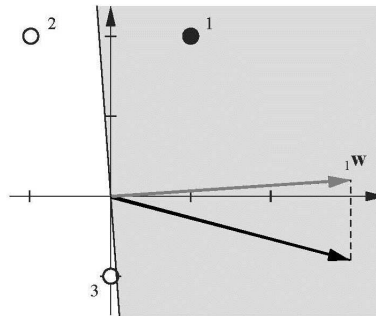


Figure 2.25: Third adjusted decision boundary

The three rules can be combined to form a single unified learning rule. First, a new variable, the perceptron error e , is defined:

$$e = t - a. \quad (2.45)$$

The three rules can be rewritten with the new variable e as:

$$\text{If } e = 1, \text{ then } {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}. \quad (2.46)$$

$$\text{If } e = -1, \text{ then } {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}. \quad (2.47)$$

$$\text{If } e = 0, \text{ then } {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}. \quad (2.48)$$

The unified rule can now be formulated as:

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + e\mathbf{p} = {}_1\mathbf{w}^{old} + (t - a)\mathbf{p}. \quad (2.49)$$

When a bias is added to the perceptron, it can be updated by using the same rule. A bias can be seen as a weight for which the input is always 1 and \mathbf{p} can thus be replaced by 1, resulting in the following rule:

$$b^{new} = b^{old} + e. \quad (2.50)$$

These two rules for updating the weights and the bias can also be modified to be used in multiple neuron perceptrons. To modify the i th row of the weight matrix, the following rule will be used:

$${}_i\mathbf{w}^{new} = {}_i\mathbf{w}^{old} + e_i\mathbf{p}. \quad (2.51)$$

To modify the i th element of the bias vector, the following rule will be used:

$$b_i^{new} = b_i^{old} + e_i. \quad (2.52)$$

These two rules, one for updating the weights and the other for updating the biases, are known collectively as the perceptron learning rule.

A more complex set of rules, called the backpropagation algorithm, can be used to train MLPs. In the next section, this learning algorithm will be discussed.

2.5.2 The backpropagation algorithm

In the discussion of the backpropagation algorithm (Hagan et al., 1996), an abbreviated notation will be used. An MLP with three layers is shown graphically with the abbreviated notation in Figure 2.26, where

$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p} + \mathbf{b}^1) + \mathbf{b}^2) + \mathbf{b}^3). \quad (2.53)$$

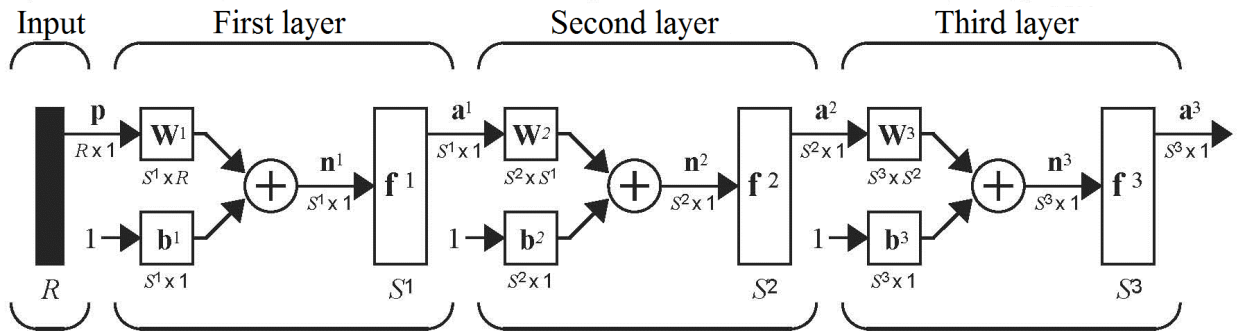


Figure 2.26: Multilayer perceptron in abbreviated notation

As discussed earlier, the output of one layer is used as the input for the next layer. This can be shown by the following:

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1}) \quad \text{for } m = 0, 1, \dots, M-1, \quad (2.54)$$

where the number of layers are represented by M . The first hidden layer receives its input from the external source:

$$\mathbf{a}^0 = \mathbf{p}. \quad (2.55)$$

The output of the last layer (output layer) is the final output of the MLP:

$$\mathbf{a} = \mathbf{a}^M. \quad (2.56)$$

As with the perceptron learning rule, the backpropagation algorithm uses a data set that contains input data as well as target output data:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \dots, \{\mathbf{p}_Q, \mathbf{t}_Q\}, \quad (2.57)$$

where \mathbf{p}_q is an input and \mathbf{t}_q is the corresponding target output with $q = 1, 2, \dots, Q$.

The backpropagation algorithm uses the mean squared error (MSE) to estimate the network parameters. The network computes an output for each input that is supplied to the network. This output is compared to the target and the network parameters are adjusted to minimize the MSE:

$$F(\mathbf{x}) = E[e^2] = E[(t - a)^2], \quad (2.58)$$

where \mathbf{x} represents the vector containing the weights and biases of the network. This can be generalized to the following if the network have multiple outputs:

$$F(\mathbf{x}) = E[\mathbf{e}^T \mathbf{e}] = E[(\mathbf{t} - \mathbf{a})^T (\mathbf{t} - \mathbf{a})]. \quad (2.59)$$

The MSE is approximated by

$$\hat{F}(\mathbf{x}) = (\mathbf{t}(k) - \mathbf{a}(k))^T (\mathbf{t}(k) - \mathbf{a}(k)) = \mathbf{e}^T(k) \mathbf{e}(k), \quad (2.60)$$

where the squared error at iteration k has replaced the expectation of the squared error. To approximate the MSE, the following steepest descent algorithm is used:

$$w_{i,j}^m(k+1) = w_{i,j}^m(k) - \alpha \frac{\partial \hat{F}}{\partial w_{i,j}^m}, \quad (2.61)$$

$$b_i^m(k+1) = b_i^m(k) - \alpha \frac{\partial \hat{F}}{\partial b_i^m}, \quad (2.62)$$

where α represents the learning rate.

The partial derivatives are calculated by using the chain rule of calculus. This is done because the error is an indirect function of the weights in the hidden layers. To review the chain rule of calculus, suppose an explicit function f exists for the variable n . If the derivative of f with respect to a third variable w must be determined, then:

$$\frac{df(n(w))}{dw} = \frac{df(n)}{dn} \times \frac{dn(w)}{dw}. \quad (2.63)$$

Consider the following example of the chain rule: If

$$f(n) = e^n \quad \text{and} \quad n = 2w, \quad \text{so that} \quad f(n(w)) = e^{2w}, \quad (2.64)$$

then

$$\frac{df(n(w))}{dw} = \frac{df(n)}{dn} \times \frac{dn(w)}{dw} = (e^n)(2). \quad (2.65)$$

This concept is used to find the derivatives in (2.61) and (2.62):

$$\frac{\partial \hat{F}}{\partial w_{i,j}^m} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial w_{i,j}^m} \text{ and} \quad (2.66)$$

$$\frac{\partial \hat{F}}{\partial b_i^m} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial b_i^m}. \quad (2.67)$$

As the net input to layer m is an explicit function of the weights and bias in that layer, the following equation can be used to compute the second terms in (2.66) and (2.67):

$$n_i^m = \sum_{j=1}^{S^{m-1}} w_{i,j}^m a_j^{m-1} + b_i^m. \quad (2.68)$$

Hence

$$\frac{\partial n_i^m}{\partial w_{i,j}^m} = a_j^{m-1}, \frac{\partial n_i^m}{\partial b_i^m} = 1. \quad (2.69)$$

The sensitivity of \hat{F} to change in the i th element of the net input of layer m can be defined as follows:

$$s_i^m \equiv \frac{\partial \hat{F}}{\partial n_i^m}. \quad (2.70)$$

As a result, (2.66) and (2.67) can be simplified to

$$\frac{\partial \hat{F}}{\partial w_{i,j}^m} = s_i^m a_j^{m-1} \text{ and} \quad (2.71)$$

$$\frac{\partial \hat{F}}{\partial b_i^m} = s_i^m. \quad (2.72)$$

The approximate steepest descent algorithm can now be expressed as

$$w_{i,j}^m(k+1) = w_{i,j}^m(k) - \alpha s_i^m a_j^{m-1} \text{ and} \quad (2.73)$$

$$b_i^m(k+1) = b_i^m(k) - \alpha s_i^m. \quad (2.74)$$

This approximate steepest descent algorithm can be written in matrix form as

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T \text{ and} \quad (2.75)$$

$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{s}^m, \quad (2.76)$$

where

$$\mathbf{s}^m \equiv \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \begin{bmatrix} \frac{\partial \hat{F}}{\partial n_1^m} \\ \frac{\partial \hat{F}}{\partial n_2^m} \\ \vdots \\ \frac{\partial \hat{F}}{\partial n_{S^m}^m} \end{bmatrix}. \quad (2.77)$$

The backpropagation algorithm gets its name from the way in which the sensitivity is calculated. The sensitivity at layer m is calculated from the sensitivity at layer $m+1$. The recurrence relationship for the sensitivities can be derived by using the following Jacobian matrix:

$$\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^m} \equiv \begin{bmatrix} \frac{\partial n_1^{m+1}}{\partial n_1^m} & \frac{\partial n_1^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_1^{m+1}}{\partial n_{S^m}^m} \\ \frac{\partial n_2^{m+1}}{\partial n_1^m} & \frac{\partial n_2^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_2^{m+1}}{\partial n_{S^m}^m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_1^m} & \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_{S^m}^m} \end{bmatrix}. \quad (2.78)$$

Next, an expression for this matrix is sought. Consider the i,j element of the matrix:

$$\frac{\partial n_i^{m+1}}{\partial n_j^m} = \frac{\partial \left(\sum_{l=1}^{S^m} w_{i,l}^{m+1} a_l^m + b_i^{m+1} \right)}{\partial n_j^m} = w_{i,j}^{m+1} \frac{\partial a_j^m}{\partial n_j^m} \quad (2.79)$$

$$= w_{i,j}^{m+1} \frac{\partial f^m(n_j^m)}{\partial n_j^m} = w_{i,j}^{m+1} \dot{f}^m(n_j^m) \quad (2.80)$$

where

$$\dot{f}^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m}. \quad (2.81)$$

Hence, the Jacobian matrix can be written as:

$$\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^m} = \mathbf{W}^{m+1} \dot{\mathbf{F}}^m(\mathbf{n}^m), \quad (2.82)$$

where

$$\dot{\mathbf{F}}^m(\mathbf{n}^m) = \begin{bmatrix} \dot{f}^m(n_1^m) & 0 & \dots & 0 \\ 0 & \dot{f}^m(n_2^m) & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \dot{f}^m(n_{S^m}^m) \end{bmatrix}. \quad (2.83)$$

The recurrence relation for the sensitivity can be written by using the chain rule in matrix form:

$$\mathbf{s}^m = \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \left(\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^m} \right)^T \frac{\partial \hat{F}}{\partial \mathbf{n}^{m+1}} = \dot{\mathbf{F}}^m(\mathbf{n}^m) (\mathbf{W}^{m+1})^T \frac{\partial \hat{F}}{\partial \mathbf{n}^{m+1}} \quad (2.84)$$

$$= \dot{\mathbf{F}}^m(\mathbf{n}^m) (\mathbf{W}^{m+1})^T \mathbf{s}^{m+1}. \quad (2.85)$$

This shows that the sensitivities are propagated backward, from the last layer to the first layer, through the network:

$$\mathbf{s}^M \rightarrow \mathbf{s}^{M-1} \rightarrow \dots \rightarrow \mathbf{s}^2 \rightarrow \mathbf{s}^1. \quad (2.86)$$

The last step is to formulate the starting point \mathbf{s}^M for the recurrence relation of (2.84), which is obtained at the final layer:

$$s_i^M = \frac{\partial \hat{F}}{\partial n_i^M} = \frac{\partial (\mathbf{t} - \mathbf{a})^T (\mathbf{t} - \mathbf{a})}{\partial n_i^M} = \frac{\partial \sum_{j=1}^{S^M} (t_j - a_j)^2}{\partial n_i^M} = -2(t_i - a_i) \frac{\partial a_i}{\partial n_i^M}. \quad (2.87)$$

Since

$$\frac{\partial a_i}{\partial n_i^M} = \frac{\partial a_i^M}{\partial n_i^M} = \frac{\partial f^M(n_i^M)}{\partial n_i^M} = \dot{f}^M(n_i^M), \quad (2.88)$$

s_i^M can be written as:

$$s_i^M = -2(t_i - a_i) \dot{f}^M(n_i^M). \quad (2.89)$$

This can be written in matrix form:

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t} - \mathbf{a}). \quad (2.90)$$

The three steps of the backpropagation algorithm can be summarized as shown in Algorithm 2.1. The backpropagation algorithm seeks to minimize the MSE. If the MSE is sufficiently small, the network is considered to have converged. An extended example of the backpropagation algorithm can be found in Hagan et al. (1996).

1. The input is propagated forward through the network:

$$\mathbf{a}^0 = \mathbf{p},$$

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1}) \text{ for } m = 0, 1, \dots, M-1,$$

$$\mathbf{a} = \mathbf{a}^M.$$

2. The sensitivity is propagated backward through the network:

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t} - \mathbf{a}),$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T \mathbf{s}^{m+1}, \text{ for } m = M-1, \dots, 2, 1.$$

3. The approximate steepest descent rule is used to update the weights and biases:

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T,$$

$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{s}^m.$$

Algorithm 2.1: Backpropagation algorithm

The MLP architecture and the backpropagation algorithm that are used to train an MLP have been discussed. Since the architecture of MLPs may differ for different problems, MLP construction will be regarded in the next section.

2.6 Multilayer perceptron construction

One of the most important tasks in ANN design is to determine the appropriate number of hidden layers and the number of neurons in the hidden layers (hidden neurons) (Basheer and Hajmeer, 2000). If too few hidden layers and neurons are used, then underfitting will occur. As a result, the network will not be able to distinguish between complex patterns. On the other hand, if the network has too many hidden layers and neurons then the result would be overfitting, as discussed in Section 2.4. As research have shown that an MLP with one hidden layer can approximate any continuous function with any desired accuracy (Zhang et al., 1998), only construction methodologies that are used to create MLPs with one hidden layer will be considered.

Normally, the data analyst wants to find the number of hidden neurons that work for a given problem. This task is much easier than determining the theoretically optimum architecture, but it may still be very difficult to decide before training what number of hidden neurons are appropriate for a given problem (Reed and Marks II, 1999). Some heuristics are available as guidelines to determine the number of hidden neurons. They include $x = 2n - 1$ (Lippmann, 1987), $x = 2n$ (Wong, 1991), $x = n$ (Tang and Fishwick, 1993) and $x = n/2$ (Kang, 1991), where x is the number of hidden neurons and n is the number of inputs. Unfortunately, there are no dependable

general rules for choosing the number of hidden neurons in advance. A common ad hoc approach is to determine the number of hidden neurons by trial and error, that is, to experiment with different configurations until one is found that performs well. This approach may be time consuming if many network configurations have to be tested before an adequate one is obtained.

Constructive methods seek to adjust the size of the network to the problem by starting with a small number of hidden neurons and adding neurons as needed until an adequate solution is found. The greatest advantage of this approach is that there is no need to make an estimate of the correct hidden layer size in advance. Reed and Marks II (1999) also discuss good reasons from a theoretical point of view to consider these algorithms.

Pruning methods, by contrast, train a larger than necessary network and then remove unneeded hidden neurons. These methods are complemented by constructive methods. Both constructive methods and pruning methods adjust the hidden layer size to the problem at hand. Even though pruning methods can be effective, they require an estimate of what number of hidden units are “larger than necessary”. Constructive methods can determine the hidden layer size without this estimate.

Sometimes, constructive methods add more neurons than necessary. For this reason, it is often useful to follow a construction phase with a pruning phase. Some algorithms let the processes compete simultaneously, one trying to add neurons, while the other attempts to remove them. At some point, the structure stabilizes when the processes balance.

An important issue that must be addressed with constructive methods, is when to stop adding new hidden neurons. By adding more hidden neurons, the training data set error can be made as small as desired. Unfortunately, each additional neuron will result in less and less benefit, according to the law of diminishing returns. It is a question of whether the incremental error reduction is worth the cost of the additional storage requirements, processing time and hardware costs. For the class of continuous problems, an infinite number of hidden neurons might be needed to achieve a zero error. In general, the data analyst must declare some nonzero error to be acceptably small and stop adding hidden neurons when it is achieved.

In addition to the question of efficiency, overfitting and generalization also cause a problem. Reed and Marks II (1999) discuss a number of factors which have an effect on generalization. When training on sampled data, the error on the training set is only an estimate of the true error, as the sampled data contain noise and other imperfections. These two error functions have a tendency to be similar, but slightly different. As a result, a change that reduces one will not always reduce the other. Normally, the error functions have large-scale similarities with small-scale differences. Both errors tend to decrease together, while learning progresses in the initial stages, as the network fits the large scale features of the training set. At some point of training the network starts to fit small-scale features where the two functions differ and additional training starts to have a negative effect on the true error. When this happens, improvements in the training error no longer correspond to improvements in the generalization error and, consequently, the network begins to overfit the data. It is often desirable to stop training before the training-set error reaches zero in order to obtain good generalization. To avoid the problem, some implementations pass it to the pruning algorithm. The constructive stage is allowed to continue well past the point of overfitting and is then followed by a pruning stage to satisfy generalization

criteria.

Constructive algorithms also have a secondary advantage. It may decrease overall training times, since useful learning occurs when the network is still small. Although a small network may not satisfy the error criteria, it may learn the dominant characteristics of the target function and consequently simplify learning in later stages. In contrast, with nonconstructive methods, an inadequate network would be abandoned and anything that was learnt would have to be relearned by the next network that was tested. The learning is retained with constructive methods and finer details are picked up as more hidden neurons are added.

The advantages of constructive algorithms over pruning algorithms justify the use of such a technique in this study. Some examples of constructive methods are Dynamic Node Creation (Ash, 1989), Cascade-Correlation (Fahlman and Lebiere, 1990), the Upstart algorithm (Freen, 1990), the Tiling algorithm (Mézard and Nadal, 1989), Marchand's algorithm (Marchand, Golea and Ruján, 1990), Meiosis Networks (Hanson, 1990), Principal Components Node Splitting (Wynne-Jones, 1992), construction from a Voronoi Diagram (Bose and Garga, 1993) and the N2C2S algorithm (Setiono, 2001). The latter constructive algorithm was chosen for this study, as it builds a feedforward neural network with a single hidden layer. To determine when to stop adding new hidden neurons, the algorithm utilizes a subset of the available training samples for cross-validation. The algorithm was originally designed to perform classification and was modified to enable a comparison with the automated construction algorithm of generalized additive neural networks (GANNs) that are discussed in the next chapter. In the next section, the algorithm is discussed.

2.6.1 The N2C2S algorithm

Assume there are P data samples $(\mathbf{x}_p, \mathbf{y}_p), p = 1, 2, \dots, P$, where input $\mathbf{x}_p \in \mathbb{R}^N$, target $\mathbf{y}_p \in [0, 1]^M$, N is the dimensionality of the input data and M is the number of classes. The algorithm also requires that the data set must be split randomly into two disjointed subsets, called the *training data set* (T) and the *cross-validation data set* (C). The training data set is used to find the optimal weights of the connections and the validation data set is used to determine the architecture of the final MLP. The objective of the algorithm is to construct and train an MLP that performs good on unseen data. Let T be a data set containing training samples $(\mathbf{x}_p, \mathbf{y}_p), p = 1, 2, \dots, P_1$, C a data set that contains cross-validation samples $(\mathbf{x}_p, \mathbf{y}_p), p = 1, 2, \dots, P_2$, with $P = P_1 + P_2$, H be the starting number of neurons in the hidden layer, and h the number of hidden neurons that are added to each new network. The N2C2S algorithm is presented in Algorithm 2.2. This algorithm begins by training an MLP with H neurons in the hidden layer and then training another MLP with $H+h$ neurons. The weights of the second MLP are set to the optimal weights that are found in the first MLP. These two MLPs are then compared to determine if the second network performs better than the first one. The accuracy results on the training data set and the cross-validation data set are added together for each MLP and then compared to each other. If the second network performed better than the first one, another network is built with h more neurons added to the hidden layer. After that, the second and third MLP will be compared. This process of adding more neurons to the new network will continue until the newer network does not perform better than the previous one. If this happens, the same network will be trained again, but

1. Construct an MLP, called N_1 , with N inputs, M outputs, and H neurons in the hidden layer.
2. Initialize the connection weights of N_1 randomly and train the network, using data set T . Score the network on the data set C and let the accuracy on data set T and C be A_{T1} and A_{C1} respectively.
3. Create a new MLP and call it N_2 , with N inputs, M outputs and $H + h$ neurons in the hidden layer.
4. Set the weights of the connections to and from the first H neurons in the hidden layer of network N_2 to the optimal weights of network N_1 . The rest of the connection weights must be set randomly. Train N_2 on the training data set T and test it on the cross-validation data set C . Let the accuracy on data set T and C be A_{T2} and A_{C2} respectively.
5. (A) If $(A_{T2} + A_{C2}) > (A_{T1} + A_{C1})$, then
 - Set $H := H + h$.
 - Let $N_1 := N_2, A_{T1} := A_{T2}, A_{C1} := A_{C2}$.
 - If $H < \text{Max}H$; go to step 3.
 (B) Else:
 - Create a new MLP, called N_3 , with $H + h$ neurons in the hidden layer. Assign random values to all the weights. Then train N_3 on the training data set T and test it on the cross-validation data set C . Let the accuracy on data set T and C be A_{T3} and A_{C3} respectively.
 - If $(A_{T3} + A_{C3}) > (A_{T1} + A_{C1})$, then
 - Set $H := H + h$.
 - Let $N_1 := N_3, A_{T1} := A_{T3}, A_{C1} := A_{C3}$.
 - If $H < \text{Max}H$; go to step 3.
6. Network N_1 is used as the final constructed MLP.

Algorithm 2.2: N2C2S algorithm

this time with randomly assigned weights. If this network also does not perform better, then the algorithm will stop, otherwise the process of adding a neuron and comparing it to the previous network will continue.

The N2C2S algorithm was modified to make the comparison with the automated construction algorithm of GANNs possible. The reasons for the alteration, modifications and resulting algorithm that are used in the study are presented next.

2.6.2 The modified N2C2S algorithm

The automated construction algorithm for GANNs (discussed in Section 3.5.3) can perform in- and out-of-sample model selection and variable selection (Du Toit, 2006). In-sample model selection is performed by choosing the best model, based on an in-sample model selection criterion. Out-of-sample model selection is obtained by splitting the input data set into a training set and a validation set. Out-of-sample performance is then measured on the validation set. While searching for the best GANN model, variable selection is done simultaneously. Only one output node is allowed for binary classification (e.g. probabilities) or regression. As a result of these possible tasks which can be performed and the restrictions on the GANN architecture, the N2C2S algorithm was modified as follows:

The target is restricted to one node ($M = 1$) and $y_p \in [0, 1]$ when binary classification is performed or $y_p \in \mathbb{R}$ for regression tasks. When in-sample model selection is done, the model is trained and evaluated on the training set. In-sample model selection criterion values are utilized for the accuracy measurements A_{T1} , A_{T2} , A_{T3} , with $A_{C1} = 0$, $A_{C2} = 0$ and $A_{C3} = 0$. For cross-validation, $A_{T1} = 0$, $A_{T2} = 0$, $A_{T3} = 0$ and the out-of-sample performance is used for the accuracy measurements A_{C1} , A_{C2} and A_{C3} . Since hidden neurons are added and removed one at a time with the automated GANN construction algorithm, $h = 1$. To ensure that search commences from the most simple architecture, $H = 1$. Variable selection was not implemented, in order to keep the modified algorithm simple and as close as possible to the original N2C2S algorithm. Algorithm 2.3 was obtained after applying these changes to the original N2C2S algorithm.

In the next section, the implementation of the modified N2C2S algorithm will be discussed.

2.6.3 Implementation of the modified N2C2S algorithm

The modified N2C2S algorithm was implemented in the SAS® Macro Language and used to search for a good MLP model with one hidden layer. This implementation has two parts. The first part uses the modified N2C2S algorithm to search for a good MLP model and the other part is a brute force method that is used to train a succession of MLP models, defined by the user, to find a good MLP model. The program is hardcoded to work with the five data sets that are used in this study. These data sets and the experiments that are conducted will be discussed in Chapter 4. In order to run a specific experiment, the user must configure some settings in the program. By changing these settings, the user can specify the data set and experiment that must be conducted on the data. The following settings are available to the user:

- *DataSelection*: Select the data set. The options are:
 - *Adult* for the Adult data set.
 - *House* for the Boston Housing data set.
 - *Ozone* for the Ozone data set.
 - *SO4* for the SO_4 data set.
 - *Spam* for the Spambase data set.

1. Construct an MLP, called N_1 , with N inputs, 1 output, and $H = 1$ neuron in the hidden layer.
2. Initialize the connection weights of N_1 randomly and train the network by using data set T . When performing in-sample model selection, evaluate the network on the data set T with the model selection criterion, let the accuracy on data set T be A_{T1} and set $A_{C1} = 0$. For cross-validation, score the network on the data set C , let the accuracy on data set C be A_{C1} and set $A_{T1} = 0$.
3. Create a new MLP and call it N_2 , with N inputs, 1 output and $H + 1$ neurons in the hidden layer.
4. Set the weights of the connections to and from the first H neurons in the hidden layer of network N_2 to the optimal weights of network N_1 . The rest of the connection weights must be set randomly. Train N_2 on the training data set T . When performing in-sample model selection, evaluate the network on the data set T with the model selection criterion, let the accuracy on data set T be A_{T2} and set $A_{C2} = 0$. For cross-validation, score the network on the data set C , let the accuracy on data set C be A_{C2} and set $A_{T2} = 0$.
5. (A) If $(A_{T2} + A_{C2}) > (A_{T1} + A_{C1})$, then
 - Set $H := H + 1$.
 - Let $N_1 := N_2, A_{T1} := A_{T2}, A_{C1} := A_{C2}$.
 - If $H < MaxH$; go to step 3.
 (B) Else:
 - Create a new MLP, called N_3 with $H + 1$ neurons in the hidden layer. Assign random values to all the weights. Thereafter, train N_3 on the training data set T . When performing in-sample model selection, evaluate the network on the data set T with the model selection criterion, let the accuracy on data set T be A_{T3} and set $A_{C3} = 0$. For cross-validation, score the network on the data set C , let the accuracy on data set C be A_{C3} and set $A_{T3} = 0$.
 - If $(A_{T3} + A_{C3}) > (A_{T1} + A_{C1})$, then
 - Set $H := H + 1$.
 - Let $N_1 := N_3, A_{T1} := A_{T3}, A_{C1} := A_{C3}$.
 - If $H < MaxH$; go to step 3.
6. Network N_1 is used as the final constructed MLP.

Algorithm 2.3: Modified N2C2S algorithm

- *Dir*: Specify the directory of the data set.

- *k*: Specify how the model is trained and validated on the data set. The following values are available:
 - $k = 0$: The complete data set is used for training and validation.
 - $k = 1$: The data set is split into training and validation subsets with user-defined sizes, specified by the *SplitTrain* and *SplitVal* values.
 - $k > 1$: K -fold cross-validation is performed where k is the number of folds.
- *SplitTrain*: The size of the training data set as a percentage of the full data set (when $k = 1$).
- *SplitVal*: The size of the validation data set as a percentage of the full data set (when $k = 1$).
- *Prelim*: The number of preliminary runs that are performed when determining the initial weights of an MLP model.
- *NetOptions*: Choose between a deviance (*dev*)- or negative log-likelihood (*like*)-based objective function.
- *Criterion*: Select the model selection criterion (when *HidNodes* = 0). The following options are available:
 - *AIC*: Akaike information criterion.
 - *VAVERR*: Average validation error.
 - *SBC*: Schwarz Bayesian criterion.
- *HidNodes*: Select the number of hidden neurons from where the brute force enumeration will commence. When *HidNodes* is set to 0, the modified N2C2S algorithm will be executed to determine the number of hidden neurons.
- *hMax*: Specify the maximum number of hidden neurons that are allowed in the model.

The brute force part of the MLP model selection program creates different MLP model architectures by enumerating through a number of neurons in the hidden layer. This is done when the parameter *HidNodes* is set to a value greater than 0. An MLP will be created with the number of neurons in the hidden layer that are set to the value of the parameter *HidNodes*. The MLP will then be trained and tested by using either the whole data set, the data set split into two subsets or K -fold cross-validation is performed. The latter technique is discussed in Section 3.5.2. A new MLP will then be created with an extra neuron in the hidden layer and with randomly assigned weights. This will continue until the number of neurons in the hidden layer reach the value of the parameter *hMax*. The results of the brute force method are utilized as a baseline for the modified N2C2S algorithm's results in Chapter 4. Both the modified N2C2S algorithm and the brute force enumeration will produce a results file which contains fit statistics.

The program code of the MLP model selection program that contains the modified N2C2S algorithm and the brute force method that was used to search for a good MLP model is shown in Appendix A. In the next section, an example which utilizes the Concrete data set (Frank and Asuncion, 2010) is given to illustrate the modified N2C2S algorithm.

2.6.4 Example

The instances of this data set represent information about the mixture of cement and the compressive strength of that specific mixture. The data set consists of 1 030 instances and has 9 attributes. This is a regression problem and the objective is to predict the compressive strength of a cement mixture (*Concrete_compressive_strength*) by using the remaining 8 attributes as inputs. These attributes are described in Table 2.2.

Attribute	Description	Attribute scale
<i>Cement</i>	Cement in a m ³ mixture, measured in kg	Interval
<i>Blast_furnace_slag</i>	Blast furnace slag in a m ³ mixture, measured in kg	Interval
<i>Fly_ash</i>	Fly ash in a m ³ mixture, measured in kg	Interval
<i>Water</i>	Water in a m ³ mixture, measured in kg	Interval
<i>Superplasticizer</i>	Superplasticizer in a m ³ mixture, measured in kg	Interval
<i>Coarse_aggregate</i>	Coarse aggregate in a m ³ mixture, measured in kg	Interval
<i>Fine_aggregate</i>	Fine aggregate in a m ³ mixture, measured in kg	Interval
<i>Age</i>	Age of the mixture in days (1-365)	Interval
<i>Concrete_compressive_strength</i>	The compressive strength of the concrete mixture, measured in megapascal (MPa)	Interval

Table 2.2: Concrete data set attributes

Methodology

The use of the modified N2C2S algorithm was selected to search for a good MLP model (*HidNodes* = 0). In-sample model selection was utilized (*k* = 0) and the SBC value was chosen as model selection criterion (*Criterion* = *SBC*). The network architecture was restricted to a maximum of 15 hidden neurons (*hMax* = 15). Finally, *Prelim* was set to 10 and *NetOptions* to *dev*.

In step 1, an MLP was created with 8 inputs and 1 hidden neuron. In step 2, the weights of this network were initialized randomly and the network was trained and scored by using the whole data set.

In step 3, a new MLP was created with 2 hidden neurons. In step 4, the weights of the connection to and from the first hidden neuron were set to the optimal weights of the previous model. The rest of the connections were set to random values and the network was trained and scored.

The result of the test in step 5(A) was positive, since the new MLP with 2 hidden neurons performed better than the previous one. This resulted in the algorithm returning to step 3 and creating a new MLP with 3 hidden neurons and continuing the process of creating and scoring more complex models which inherit the previous optimal weight values and then comparing them to the previous best model. However, when a model with 6 hidden neurons was created, it did not perform better than the previous model and thus a new MLP, still with 6 hidden neurons, was created, but with random weight values (step 5(B)). This model was then compared to the previous best model in terms of the SBC value and the indication was that the previous best model was better

than the new model. It resulted in the termination of the program.

Results

The best model that was found was reported to be the MLP with 5 hidden neurons. The SBC value of this model was -3 472.04 and the MSE value 25.10081. The complexity of the model was reported to be 51 (number of parameters). It took 16s55ms to complete the experiment to find a good MLP model for the Concrete data set.

In the next section, a conclusion to this chapter is presented.

2.7 Conclusion

In this chapter, the history of ANNs was considered and it was showed that the biological neuron inspired the development of the artificial neuron. This biological inspiration of the modern day ANN was also discussed. Next, the neuron model architecture was considered, which included single-input neurons, multiple-input neurons, the perceptron and a layer of neurons. This was followed by the MLP model. Neural network learning was then considered, which included the perceptron learning rule and the backpropagation algorithm. Finally, the construction of MLPs was considered. The original N2C2S algorithm for constructing MLPs with one hidden layer by using cross-validation was first considered, followed by a modified version of this algorithm and the implementation of it.

With this chapter, a better understanding of ANNs and MLPs was obtained and a method for constructing MLPs was selected and modified to be used in the experiments (Chapter 4) in order to compare MLPs and GANNs.

The GANN is a relatively new type of neural network that is based on the generalized additive model. This type of neural network uses an MLP with one hidden layer for each input, which enable the modeller to adjust the complexity of each input's MLP individually. GANNs attempt to overcome some of the difficulties that are associated with MLPs. One of these difficulties is the problem of selecting an appropriate network architecture for a specific data set. Potts (1999) suggested an interactive construction algorithm which uses partial residual plots and human judgement to select a good GANN architecture. Du Toit (2006) improved on this interactive construction algorithm by introducing an automated construction algorithm which uses a model selection criterion to select a good GANN model objectively. In the next chapter, GANNs will be considered together with the interactive construction algorithm and the automated construction algorithm that is implemented in a system called *AutoGANN*.

“Integrity without knowledge is weak and useless, and knowledge without integrity is dangerous and dreadful.”

Samuel Johnson

3

Generalized additive neural networks

The main reason for neural networks’ popularity is their flexible nonlinear modelling and powerful pattern recognition capabilities (Du Toit, 2006). Neural networks are data driven without any restrictive assumptions that constrain the functional relationship between the target variable and the input variables. Neural networks with this unique characteristic are highly desirable in many situations where ample data are generally available, but where the underlying data-generating mechanism is often unknown or untestable. There are, however, some practical difficulties with the utilizing of neural networks for prediction problems. Three of these difficulties are inscrutability, model selection and troublesome training (Potts, 1999).

Multilayer perceptrons (MLPs) are commonly regarded to be black boxes with respect to interpretation. The relationship of certain inputs to the target can depend on the values of other inputs in complicated ways. Certain pattern recognition applications, like handwriting recognition, where pure prediction is the goal, does not require an explanation of how the neural network derived the answer. On the other hand, in some problems, like hypothesis testing, understanding how the neural network derived the output is more important than the output itself. Certain domains often have both goals, namely understanding the outcome of the neural network and obtaining the output of the neural network. An example of such a domain is database marketing. The ultimate purpose of predictive modelling is the scoring of new cases, but some understanding, even informal, of the factors affecting the prediction can be helpful in finding out how to market to segments of people that are likely to respond. Decisions about costly data acquisitions can also be guided by an understanding of the effects of the inputs. The black box characteristic of the model can have legal consequences in credit scoring. Creditors are required, by the US Equal Credit Opportunity Act, to provide an argument with specific reasons

to support an adverse action. The argument that the applicant failed to achieve the qualifying score on the creditor's scoring system is regarded by the regulation to be insufficient.

The second practical difficulty is the fact that when a neural network is configured for a problem, there are a huge number of configurations to choose from. The number of layers, number of neurons in each layer, activation functions, type of connections et cetera need to be chosen in order for the neural network to perform adequately. Currently, trial and error is the most reliable method for the construction of a neural network.

The third practical difficulty is the computational effort that is required when training a neural network, as a large number of parameters (weights and biases) must be optimized. Local minima are troublesome, since different starting values can lead to different (faulty) solutions. Multiple runs from different starting values are frequently the best solution.

These difficulties are reduced with the use of generalized additive neural networks (GANNs), since their architecture is constrained. Graphical methods can be used to interpret the effect of each input on the fitted model. The network complexity can be visually determined with partial residual plots, while generalized linear models can be used to initialize GANNs with the addition of skip layers (direct connections). Unfortunately, determining the network complexity visually with partial residual plots is subjective to human judgement and can be time consuming for a large number of variables (Du Toit, 2006). Consequently, Du Toit (2006) developed an automated construction algorithm for GANNs and called the implementation *AutoGANN*. This system can overcome these drawbacks by relying on a model selection criterion or cross-validation to search for good GANN models. While searching for the best GANN model, no human interaction is needed.

Since a GANN is the neural network implementation of a generalized additive model (GAM), a discussion on GANNs would be incomplete if GAMs and the backfitting algorithm for the estimation of GAM models were not considered. Smoothing, which summarizes the trend of a response measurement as a function of one or more predictor measurements, is considered in Section 3.1. To illustrate smoothing, the running-mean smoother is used and the bias-variance trade-off for determining the value of the smoothing parameter is explained. Additive models are discussed in Section 3.2. The backfitting algorithm that is used for estimating additive models is explained and the GAM, which is an extension of additive models, is considered. The backfitting algorithm utilizes the scatterplot smoother, which is a special type of smoother. In Section 3.3 the GANN architecture is discussed and the interactive construction methodology is considered in Section 3.4. The automated construction methodology is explained in Section 3.5 by defining certain terms and considering in-sample and out-of-sample model selection criteria. The automated construction algorithm and the implementation of this algorithm is also discussed. Finally, a conclusion is presented in Section 3.6. Note that, since there is so little literature on GANNs, the content of this chapter is mainly obtained from Du Toit (2006).

3.1 Smoothers

The linear model is simple in structure, it's least-squares theory is elegant and it is interpretable by the user. Since computing power has grown significantly, the linear model can be augmented with new models that

assume less and therefore, potentially, discover more. One of these new models, called the *additive model* (Hastie and Tibshirani, 1990), is described in Section 3.2. The additive model is a generalization of the linear model. An input's linear function is replaced with an unspecified smooth function. The additive model consists of a sum of smooth functions. These functions are estimated by using scatterplot smoothers in an iterative manner. The estimated additive model consists of a function for each input. This can help data analysts to discover the appropriate shape of each of the input effects.

The additive model retains some of the interpretability of the linear model by assuming additivity of effects. To estimate the univariate function would have been computationally unthinkable four decades ago, but with the fast computers that are available today, it can be achieved.

A smoother summarizes the trend of a response measurement Y as a function of one or more predictor measurements X_1, \dots, X_p . The name *smoother* comes from the fact that the estimate of the trend that is produced is less variable than Y itself. A smoother does not assume a rigid form for the dependence of Y on X_1, \dots, X_p and, consequently, a smoother is often referred to as a tool for nonparametric regression. An example of a simple smoother is the running-mean (moving average) smoother. A regression line with a rigid parametric form is not strictly thought of as a smoother. A *smooth* is the name given to the estimate that is produced by a smoother. The most common case is that of a single predictor and is called *scatterplot smoothing*.

To illustrate scatterplot smoothing, the Diabetes data set (Sackett, Daneman, Clarson and Ehrich, 1987) is utilized. The Diabetes data set originated from a study of the factors that affect patterns of insulin-dependent mellitus in children. This study investigated the dependence of the level of serum C-peptide on several other factors to understand the patterns of residual insulin secretion. In this illustration, only a subset of two factors that were studied in Sackett et al. (1987) are used. Information about the attributes of this data set is given in Table 3.1. The predictor attributes are *Age* and *Base_deficit* and the response attribute is *Log(C-peptide)*.

Attribute	Description	Attribute scale
<i>Age</i>	The age of the child	Interval
<i>Base_deficit</i>	A measure of acidity	Interval
<i>Log(C-peptide)</i>	The logarithm of C-peptide concentration found at the diagnosis	Interval

Table 3.1: Diabetes data set attributes

There are two main functions of smoothers, of which description is the first one. The visual appearance of the scatterplot of Y versus X is enhanced with a scatterplot smoother. This helps the data analyst to pick out the trend in the plot. Figure 3.1 shows a plot of $\log(C-peptide)$ versus age. It seems that $\log(C-peptide)$ has a strong dependence on age and a scatterplot smoother can provide assistance in describing the relationship between $\log(C-peptide)$ and age. The second function of a smoother is to estimate the dependence of the mean of Y on the predictors, and consequently serves as a building block for the estimation of additive models.

Local averaging is used by most smoothers. Local averaging averages the Y -values of observations which have predictor values that are close to a target value. The averaging is done within neighbourhoods around

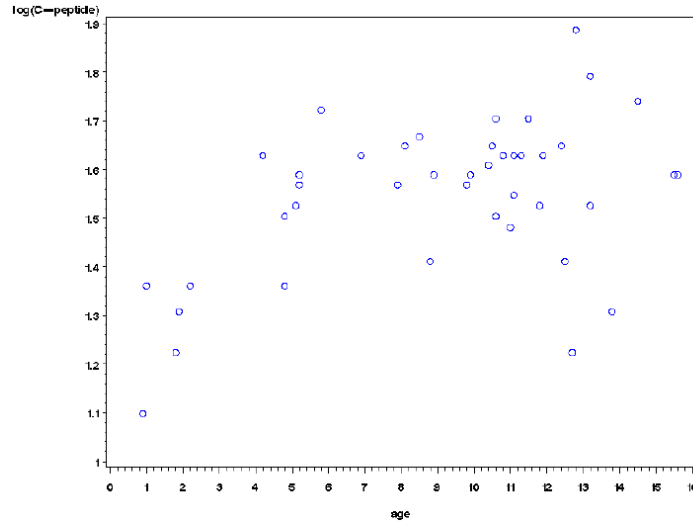


Figure 3.1: Scatterplot of $\log(C\text{-peptide})$ versus *Age*

the target value. There are two decisions to be made when using scatterplot smoothing. The first decision is concerned with how the response values in each neighbourhood should be averaged. The decision is thus which type of smoother to use, because smoothers differ mainly in their method of averaging. The second decision is concerned with how large the neighbourhoods should be made. This decision is typically expressed in terms of an adjustable smoothing parameter. A small neighbourhood will produce an estimate with high variance, but potentially low bias and a large neighbourhood will produce an estimate with a low variance but, potentially high bias. As a result, the smoothing parameter controls the trade-off between bias and variance. The amount of smoothing is calibrated according to the number of equivalent degrees of freedom.

A formal definition of scatterplot smoothing is given in the next section.

3.1.1 Scatterplot smoothing

Assume that $\mathbf{y} = (y_1, \dots, y_n)^T$ exists at $\mathbf{x} = (x_1, \dots, x_n)^T$ where \mathbf{x} is design points, \mathbf{y} is response measurements, and $(y_1, \dots, y_n)^T$ and $(x_1, \dots, x_n)^T$ are the transpose of the vectors (y_1, \dots, y_n) and (x_1, \dots, x_n) . Also assume that measurements of variables Y and X are represented by each of \mathbf{y} and \mathbf{x} .

Not many duplicates are expected at any given value of X , as Y and X are noncategorical. It is assumed, for simplicity, that the data are sorted by X and that there are no duplicate X values, which means that $x_1 < \dots < x_n$. Weighted smoothers can be applied in case of duplicates.

A scatterplot smoother can be defined as a function of \mathbf{x} and \mathbf{y} that has the same domain as the values in \mathbf{x} : $s = S(\mathbf{y}|\mathbf{x})$. The function $S(\mathbf{y}|\mathbf{x})$ that is measured at x_0 , which is the set of instructions that determines $s(x_0)$, is generally defined for all $x_0 \in [-\infty, \infty]$. Sometimes, $s(x_0)$ is defined solely at x_1, \dots, x_n , which are the sample values of X . In this case, the estimates at other X -values are obtained by using some kind of interpolation.

A number of scatterplot smoothers, which include kernel smoothers, locally weighted running-line smoothers, running-line smoothers, cubic smoothing splines, bin smoothers, running-mean smoothers and regression splines, are discussed by Hastie and Tibshirani (1990). The trade-off between bias and variance governs the decisions about the complexity of models. To illustrate this trade-off, the running-mean smoother is discussed next in

more detail.

3.1.2 The running-mean smoother

Assume the target value x_0 is the same as one of the x_j s, say x_i . If there are duplicates at x_i , the average of the Y -values at x_i can be utilized to estimate $s(x_i)$. If there are no duplicates, the Y -values that correspond to X -values close to x_i are averaged. Selecting x_i itself, as well as k points to the left of x_i and k points to the right of x_i that are nearest in X -value to x_i is a simple way to choose points close to x_i . This way of selecting points is called a *symmetric nearest neighbourhood* and $N^S(x_i)$ refers to the indices of these points. The running-mean is thus defined by

$$s(x_i) = \text{ave}_{j \in N^S(x_i)}(y_j). \quad (3.1)$$

When it is not possible to pick k points, as many points as possible are taken from the left and right of x_i . A formal definition of a symmetric nearest neighbourhood is the following:

$$N^S(x_i) = \{\max(i - k, 1), \dots, i - 1, i, i + 1, \dots, \min(i + k, n)\}. \quad (3.2)$$

It is not apparent how to define the symmetric nearest neighbours for the target points x_0 other than the x_i . Linear interpolation between the fit of two values of X in the sample adjacent to x_0 is one solution to do this. Another solution is to ignore symmetry and pick the r closest points to x_0 , regardless of which side they are on. This is called a *nearest neighbourhood*. Arbitrary values of x_0 are treated in an uncomplicated and clean manner.

The running-mean smoother is also called a *moving average smoother*. This smoother is popular for equally-spaced time series data. Given its simplicity, it is valuable for theoretical calculations, but it does not work satisfactorily in practice. It tends to be wiggly and contains flattened-out trends near the endpoints. A running-mean smooth with $k = 11$, or about 25% of the 43 observations, is shown in Figure 3.2.

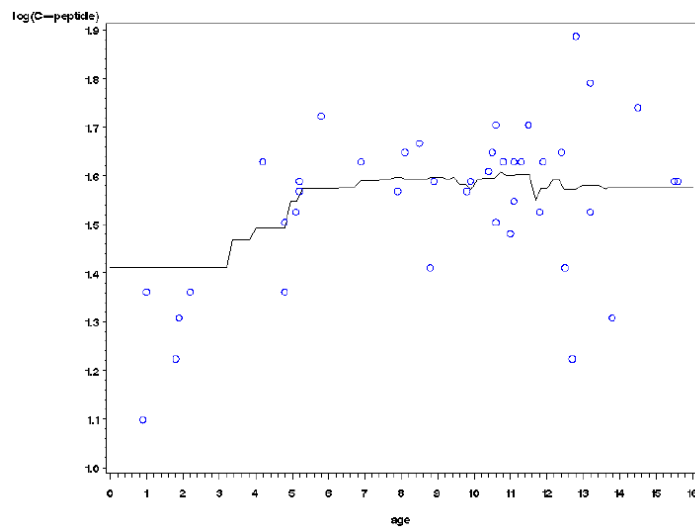


Figure 3.2: Running-mean smoother with 25% span

3.1.3 Smoothers for multiple predictors

Thus far, a scatterplot smoother which is used for a single predictor has been discussed. With more than one predictor present, say X_1, \dots, X_p , the problem is one of fitting a p -dimensional surface to Y . Multiple regression of Y on X_1, \dots, X_p allows for a simple, but very limited, estimate of the surface. In theory, the running mean can easily be generalized to this setting. A definition of a nearest neighbourhood of a point in p -space is required for this smoother. A distance measure is used to determine the nearest neighbourhood. The most apparent choice for the distance measure is the Euclidean distance. When $p > 1$, the concept of symmetric nearest neighbours is no longer significant. After a neighbourhood is defined, the generalization of the running mean utilizes the average of the response values in the neighbourhood and calculates the surface at the target point.

It is argued that for more than two or three predictors, multi-predictor smoothers are not very useful anymore (Hastie and Tibshirani, 1990). These type of smoothers also have many defects, such as difficulty of computation and interpretation.

3.1.4 The bias-variance trade-off

In the previous sections, no assumption was made of the formal relationship between the response variable Y and the predictor variable X . To set the basis for additive models, this assumption is now made. It is assumed that

$$Y = f(X) + \epsilon \quad (3.3)$$

where the expected value of ϵ , $E(\epsilon)$, is 0 and the variance of ϵ , $var(\epsilon)$, is σ^2 . It is also assumed that the errors ϵ are independent. The objective of a scatterplot smoother is to estimate the function f . From (3.3), $E(Y|X = x) = f(x)$. Note that \hat{f} is now used to denote the fitted functions, rather than the s that was used in the previous sections. Since the running mean is built by averaging Y -values corresponding to x -values close to a target value x_0 , this smoother can be seen as estimates of $E(Y|X = x)$. The averaging requires values of $f(x)$ near $f(x_0)$. This implies $E\{\hat{f}(x_0)\} \approx f(x_0)$, since $E(\epsilon) = 0$. For a cubic smoothing spline under certain regularity conditions, it can be shown that $\hat{f}(x) \rightarrow f(x)$, as $n \rightarrow \infty$ and the smoothing parameter $\lambda \rightarrow 0$, where n represents the number of design points and λ represents the window width. As a result, the smoothing-spline estimate will converge to the true regression function $E(Y|X = x)$ as more and more data are obtained.

A key trade-off exists between the bias and the variance of the estimate in scatterplot smoothing. The smoothing parameter controls this trade-off. The trade-off can easily be seen in the case of the running mean. The fitted running-mean smooth can be defined as

$$\hat{f}_k(x_i) = \sum_{j \in N_k^s(x_i)} \frac{y_j}{2k+1} \quad (3.4)$$

with expectation

$$E\{\hat{f}_k(x_i)\} = \sum_{j \in N_k^s(x_i)} \frac{f(x_j)}{2k+1} \quad (3.5)$$

and variance

$$var\{\hat{f}_k(x_i)\} = \frac{\sigma^2}{2k+1}. \quad (3.6)$$

It is assumed, for ease of notation, that x_i is close to the middle of the data, so that $N_k^S(x_i)$ contains the full $2k + 1$ points. From (3.4) and (3.5), it can be seen that the variance decreases as k is increased but since the expectation $\sum_{j \in N_k^S(x_i)} f(x_j) / (2k + 1)$ involves more terms with function values, $f(\cdot)$, which differs from $f(x_i)$, the bias tends to increase. In a similar manner, the variance increases as k is decreased, but this inclines to decrease the bias. This phenomenon is also encountered when adding or deleting terms from a linear regression model and is known as the *bias-variance trade-off*.

The running-mean smooths that are using 20%, 50% and 80% of the 43 observations of the diabetes data set is shown in Figures 3.3, 3.4 and 3.5. These figures show that smoother, but flatter, curves are produced with a larger percentage of observations.

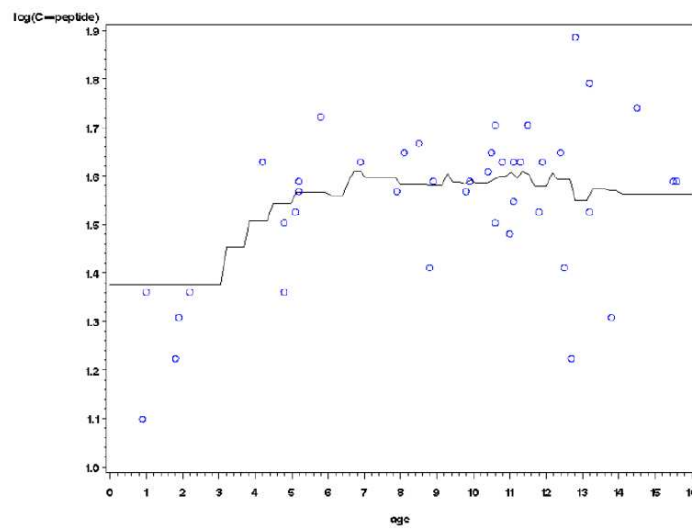


Figure 3.3: Running-mean smoother with 20% span

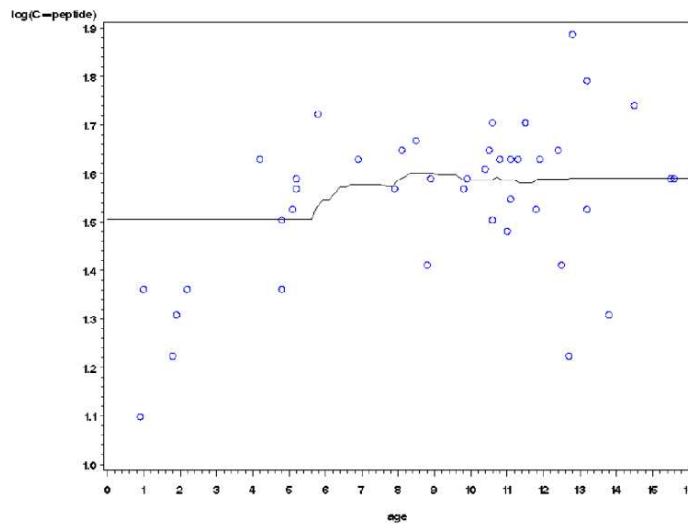


Figure 3.4: Running-mean smoother with 50% span

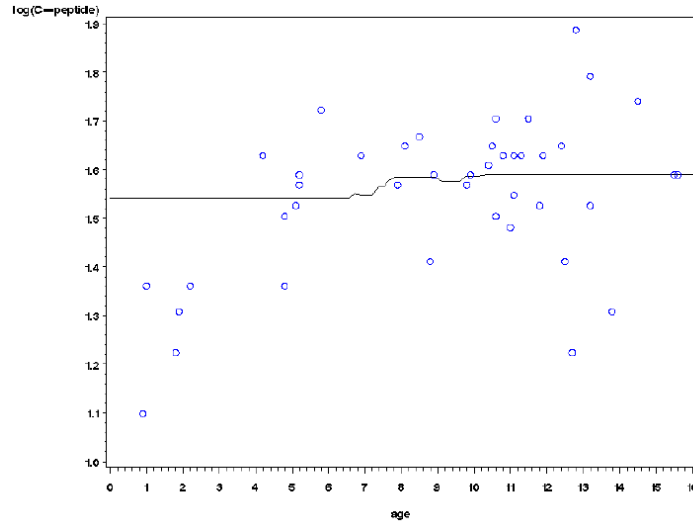


Figure 3.5: Running-mean smoother with 80% span

The additive model for multiple regression data is discussed in the next section, as well as the backfitting algorithm for its estimates. This backfitting algorithm uses scatterplot smoothers to determine the functional form of the additive model.

3.2 Additive models

The usual linear regression model is generalized to form the additive model. An outline of the limitations of the linear model is presented next and reasons why one would require to generalize it is important. An arbitrary regression surface would be a natural generalization. There are, regrettably, problems with the estimation and interpretation of fully general regression surfaces. These problems restrict attention to additive models.

3.2.1 Multiple regression and linear models

Suppose there are n observations on a response variable Y , with a multiple regression problem that is represented by $\mathbf{y} = (y_1, \dots, y_n)^T$ and measured at n design vectors $\mathbf{x}^i = (x_{i1}, \dots, x_{ip})$. The points \mathbf{x}^i may be measurements of random variables X_j for $j = 1, \dots, p$, or may be picked in advance, or both. These two situations are not distinguished.

There are several reasons to model the dependence of Y on X_1, \dots, X_p :

- Description: The dependence of the response on the predictors is described by using a model, so that more can be discovered about the process that produces Y .
- Inference: The proportional contributions in explaining Y are assessed for each of the predictors.
- Prediction: Y needs to be predicted for some set of values X_1, \dots, X_p by the data analyst.

The multiple linear regression model is the standard tool that is used by the applied statistician for these intentions. This model is defined as

$$Y = \alpha + \alpha_1 X_1 + \dots + \alpha_p X_p + \epsilon, \quad (3.7)$$

where $E(\epsilon) = 0$ and $\text{var}(\epsilon) = \sigma^2$. The model makes a strong assumption about the dependence of $E(Y)$ on X_1, \dots, X_p , namely that there is a linear dependence in each of the predictors. If this assumption holds more or less, then the linear regression model is very useful, since

- it provides a simple description of the data;
- a single coefficient sums up the contribution of each predictor; and
- new observations can be predicted with a simple method.

There are many ways in which the linear regression model can be generalized. One class of candidates is the surface smoothers and can be viewed as nonparametric estimates of the regression model

$$Y = f(X_1, \dots, X_p) + \epsilon. \quad (3.8)$$

Choosing the shape of the neighbourhood that defines local in p dimensions is troublesome with surface smoothers. An even more serious problem that is common to all surface smoothers is that neighbourhoods with a set number of points become less local as the dimensions increase. This problem has been called the *curse of dimensionality* by Bellman (1961).

Partially as an answer to the dimensionality problem, a number of multivariate nonparametric regression techniques have been devised. Projection pursuit regression and recursive-partitioning regression (Friedman and Stuetzle, 1981) are examples of these multivariate nonparametric regression techniques. These models have good predictive power when enough data are available. All of them are consistent to the true regression surface if they are under suitable conditions. Unfortunately, all of these methods suffer from being hard to interpret. A specific problem is how the effect of particular variables should be analyzed when a complicated surface has been fitted.

A crucial feature of the linear model that has made it so popular for statistical inference is stressed by the interpretation problem: The linear model is additive in the predictor effects. The predictor effects can be analyzed separately in the absence of interactions after the linear model has been fitted. This crucial characteristic of being additive in the predictor effects is retained by the additive models.

3.2.2 Additive models defined

The additive model can be written as

$$Y = \alpha + f_1(X_1) + \dots + f_p(X_p) + \epsilon, \quad (3.9)$$

where the errors ϵ are independent of the X_j s, $E(\epsilon) = 0$ and $\text{var}(\epsilon) = \sigma^2$. Each predictor has an unspecified univariate function, namely f_j . It is implied from the definition of additive models that $E\{f_j(X_j)\} = 0$. There would be free constants in each of the functions if this were not the case.

A crucial interpretive characteristic of the linear model is kept by the additive model: The values of the other predictors do not influence the variation of the fitted response surface that holds all but one predictor fixed. This results from the fact that each variable is represented individually in (3.9). The p univariate functions can thus

be plotted individually to analyze the roles of the predictors in modelling the response once the additive model is fitted to data. The additive model is nearly always an approximation to the true regression surface, but hopefully a useful one. This is unfortunately the price that has to be paid for simplicity. It is usually not assumed that a linear regression model is correct when it has been fitted. Instead, it is believed that the model will be a good first order approximation to the true surface, and that the important predictors and their roles can be exposed using the approximation. Additive models are more general approximations than linear regression models.

An additive model's estimated functions correspond to the coefficients in a linear regression. Additive models are prone to all the possible problems that are found in interpreting linear regression models and these problems can be expected to be more serious. Care must be taken not to have insignificant variables affect important functions when interpreting these functions.

The backfitting algorithm for estimating additive models is considered next.

3.2.3 Fitting additive models

There are many ways to approach the formulation and estimation of additive models. A number of methods, including regression splines, more general versions of multiple regression, multiple regression and smoothing splines, are discussed by Hastie and Tibshirani (1990). An arbitrary smoother is used by the most general method to estimate the functions. A data analyst can fit an additive model by using any regression-type fitting mechanism with the general backfitting algorithm. The price for this added generality is the fact that the algorithm is an iterative fitting procedure.

Conditional expectations allow for a simple intuitive motivation for the backfitting algorithm. Assuming that the additive model in (3.9) is correct, then for any k ,

$$E(Y - \alpha - \sum_{j \neq k} f_j(X_j) | X_k) = f_k(X_k), \quad (3.10)$$

where α is the constant term. An iterative algorithm for calculating all the f_j s is immediately proposed by the conditional expectations in (3.10). This iterative algorithm is presented next in terms of data and arbitrary scatterplot smoothers S_j .

1. Initialize: $\alpha = \text{ave}(y_i), f_j = f_j^0, j = 1, \dots, p$
2. Cycle: $j = 1, \dots, p, 1, \dots, p, \dots$

$$f_j = S_j(\mathbf{y} - \alpha - \sum_{k \neq j} \mathbf{f}_k | \mathbf{x}_j)$$
3. Continue 2. until there are no changes in the individual functions.

The $(\mathbf{y} - \alpha - \sum_{k \neq j} \mathbf{f}_k | \mathbf{x}_j)$ expression denotes the partial residual in the backfitting algorithm. All of the effects of the other variables are removed from \mathbf{y} before this partial residual is smoothed against x_j , when the univariate function f is being readjusted. Only if all the functions are correct (and therefore the iteration), this is appropriate.

To start the algorithm, initial functions (f_j^0) must be provided. A reasonable starting point might be the

linear regression of \mathbf{y} on the predictors, if no previous knowledge of the functions exists. The backfitting algorithm is frequently used within some bigger iteration, where the functions from the previous big iteration loop supply starting values. The convergence of the backfitting algorithm for a number of different types of smoothers is discussed by Hastie and Tibshirani (1990). No proof of convergence exists for certain types of smoothers, like locally-weighted running-line smoothers, but their experience has been reassuring and counter examples are difficult to find.

So far, the discussion deals with the linear regression model that is extended by a type of model, called *the additive model*, where the average of the response is modelled as an additive sum of the predictors. An additive extension of the family of generalized linear models is described in the next section. The predictor effects are assumed to be linear in the predictors with generalized linear models, but the distribution of the responses and the link between the predictors and this distribution can be universal.

3.2.4 Generalized additive models defined

Generalized linear models is extended by generalized additive models in the same way as the linear regression model is extended by the additive model.

The generalized linear model (McCullagh and Nelder, 1989) is given by

$$g_0^{-1}(E(Y)) = \alpha_0 + \alpha_1 X_1 + \dots + \alpha_p X_p + \epsilon, \quad (3.11)$$

where $E(\epsilon) = 0$ and $\text{var}(\epsilon) = \sigma^2$. In (3.11), a link function g_0^{-1} , which is the inverse of the (neural network) activation function g_0 , is utilized to constrain the range of response values. The logit link function is appropriate when the expected response is bounded between 0 and 1, such as probability. The logit link function is defined as

$$g_0^{-1}(E(Y)) = \ln \left(\frac{E(Y)}{1 - E(Y)} \right). \quad (3.12)$$

The hyperbolic tangent link function can be utilized when an expected response is bounded between -1 and 1. The latter is defined as

$$g_0^{-1}(E(Y)) = 1 - \frac{2}{1 + \ln(2E(Y))}. \quad (3.13)$$

A generalized additive model (GAM) (Hastie and Tibshirani, 1987; Wood, 2006) is given by

$$g_0^{-1}(E(Y)) = \alpha + f_1(X_1) + \dots + f_p(X_p) + \epsilon, \quad (3.14)$$

where $E(\epsilon) = 0$ and $\text{var}(\epsilon) = \sigma^2$.

The most widely used type of artificial neural network for supervised prediction is the multilayer perceptron (MLP), which was discussed in Chapter 2. MLPs are, theoretically, universal approximators that are able to model any continuous function (Ripley, 1996) and, as a result, MLPs can be utilized as the univariate functions of GAMs. Generalized additive neural networks (GANNs) are the neural network implementation of GAMs. With GANNs, backfitting is unnecessary, since any method suitable for the fitting of MLPs can be used to simultaneously estimate the parameters of GANN models. The usual optimization and model complexity issues thus also apply to GANN models.

Next, the GANN architecture is discussed.

3.3 Generalized additive neural network architecture

An MLP that has one hidden layer with h neurons is defined as

$$g_0^{-1}(E(y|\mathbf{x})) = w_0 + w_1 \tanh(w_{01} + \sum_{j=1}^p w_{j1}x_j) + \dots + w_h \tanh(w_{0h} + \sum_{j=1}^p w_{jh}x_j), \quad (3.15)$$

where \tanh is the hyperbolic tangent activation function, as suggested by Potts (2000). In (3.15), the link-transformed expected value of the target is expressed as a linear combination of nonlinear functions of linear combinations of the inputs. This model consists of $h(p+1) + 1$ unknown parameters (weights and biases). Some suitable measure of fit to the training data, for example the negative log likelihood, is numerically optimized to estimate the parameters.

In the basic structure of a GANN, each input has a separate MLP with one hidden layer of h neurons, and can be defined as

$$f_j(x_j) = w_{1j} \tanh(w_{01j} + w_{11j}x_j) + \dots + w_{hj} \tanh(w_{0hj} + w_{1hj}x_j). \quad (3.16)$$

The individual bias terms are absorbed by the overall bias α . There are $3h$ parameters for each individual univariate function, where h could differ for each input.

Figure 3.6 shows an example GANN with two inputs. The first input has an MLP with three neurons in the hidden layer and the second input has an MLP with two neurons in the hidden layer. Neurons in the consolidation layer correspond to the univariate functions. The weights are fixed at 1.0 between the consolidation layer and the output layer. The first univariate function of this example is given by

$$f_1(x_1) = w_{11} \tanh(w_{011} + w_{111}x_1) + w_{21} \tanh(w_{021} + w_{121}x_1) + w_{31} \tanh(w_{031} + w_{131}x_1) \quad (3.17)$$

and the second univariate function is defined as

$$f_2(x_2) = w_{12} \tanh(w_{012} + w_{112}x_2) + w_{22} \tanh(w_{022} + w_{122}x_2). \quad (3.18)$$

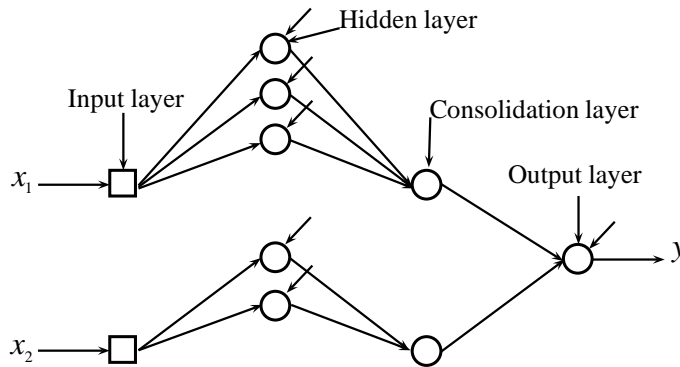


Figure 3.6: Basic GANN architecture

The generalized linear model can be regarded as a special case when enhancing this basic architecture to include an additional parameter for a direct connection (skip layer), so that

$$f_j(x_j) = w_{0j}x_j + w_{1j} \tanh(w_{01j} + w_{11j}x_j) + \dots + w_{hj} \tanh(w_{0hj} + w_{1hj}x_j). \quad (3.19)$$

An example of this enhanced GANN architecture with three inputs is shown in Figure 3.7. The first input has an MLP with one neuron in the hidden layer and a skip layer. The second input has an MLP with two neurons in the hidden layer and the third input has an MLP with three neurons in the hidden layer. The first univariate function in this example is given by

$$f_1(x_1) = w_{01}x_1 + w_{11} \tanh(w_{011} + w_{111}x_1), \quad (3.20)$$

the second univariate function is

$$f_2(x_2) = w_{12} \tanh(w_{012} + w_{112}x_2) + w_{22} \tanh(w_{022} + w_{122}x_2), \quad (3.21)$$

and the third and final univariate function is

$$f_3(x_3) = w_{13} \tanh(w_{013} + w_{113}x_3) + w_{23} \tanh(w_{023} + w_{123}x_3) + w_{33} \tanh(w_{033} + w_{133}x_3). \quad (3.22)$$

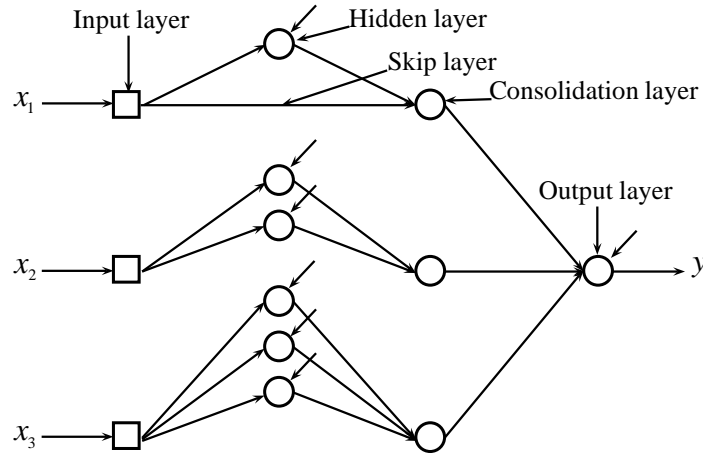


Figure 3.7: Enhanced GANN architecture

An iterative algorithm for constructing GANNs is presented in the next section. This methodology guides the modeller in visually deciding on the appropriate complexity of the individual univariate functions.

3.4 The interactive construction methodology

To analyze nonlinear relationships between the target and input variables in multiple regression models, a diversity of diagnostic plots have been used for more than half a century. There are, in general, two complementary approaches to analyze the assumption of linearity: formal tests and informal graphical methods (Cai and Tsai, 1999). Larsen and McCleary (1972) named an informal graphical method that was introduced by Ezekiel (1924) as the *partial residual plot*. This method is still often used.

Plots of the fitted univariate functions, $\hat{f}_j(x_j)$, overlaid on the partial residuals versus the corresponding j th input, are used for visual diagnostics to assist the model selection process for GANNs. The partial residuals is defined as:

$$pr_j = g_0^{-1}(y) - \alpha - \sum_{l \neq j} \hat{f}_l(x_l) = (g_0^{-1}(y) - g_0^{-1}(\hat{y})) + \hat{f}_j(x_j). \quad (3.23)$$

A first-order approximation is usually utilized when g_0^{-1} is nonlinear:

$$pr_j = \frac{\partial g_0^{-1}(\hat{y})}{\partial y}(y - \hat{y}) + \hat{f}_j(x_j). \quad (3.24)$$

The effect of the individual inputs that are adjusted for the effect of the other inputs can be analyzed with partial residuals. The j th partial residual is the difference between the actual values and that portion of the fitted model that does not involve x_j .

The interactive construction algorithm starts with a GANN architecture that consists of an MLP with one hidden neuron and a skip layer for each input, instead of the linear model. The linear fit is solely used for initialization. Berk and Booth (1995) discussed the effectiveness of partial residual plots for visualizing the underlying curve. They showed that the partial residuals that are based on a linear fit are less reliable than those that are based on a GAM fit and that it is also common practise with GAM estimation to start with four parameters.

To simplify optimization and model selection, the following set of instructions for constructing a GANN interactively (Potts, 1999) utilizes their constrained form. This algorithm consists of six steps, as shown in Algorithm 3.1.

1. A GANN must be constructed with a skip layer and one hidden neuron for each input. This initial GANN is defined as follows:

$$f_j(x_{ji}) = w_{0j}x_{ji} + w_{1j}\tanh(w_{01j} + w_{11}x_{ji}).$$

This gives a degree of freedom (number of parameters) of 4 for each input. Binary inputs must only have a skip layer and no hidden neurons.

2. Next a generalized linear model must be fitted to give initial estimates of α and w_{0j} .
3. The remaining 3 parameters must be initialized in each hidden layer as random values from a normal distribution with mean zero and variance equal to 0.1.
4. The full GANN model must now be fitted.
5. Each of the fitted univariate functions that are overlaid on their partial residuals must then be analyzed.
6. Remove neurons (prune) from the hidden layers with evidently linear effects and add neurons (grow) to hidden layers where the nonlinear trend seems to be underfitted. If this step is repeated, the final estimates from previous fits can be utilized as initial values.

Algorithm 3.1: Interactive construction algorithm

The Kyphosis data set (Bell, Walker, O'Connor, Orrel and Tibshirani, 1989) is used in the next section to

illustrate the interactive construction methodology.

3.4.1 Example

Kyphosis is a spinal deformity that can occur after certain spinal surgeries have been performed on children. The Kyphosis data set has 83 instances with 4 attributes. Each instance represents a child that underwent spinal surgery. The 4 attributes are described in Table 3.2.

Attribute	Description	Attribute scale
<i>Age</i>	The age of the child in months	Interval
<i>Number</i>	The number of vertebrae that are involved in the child's spinal surgery	Interval
<i>Start</i>	The starting vertebra number of the range of the vertebrae involved in the operation	Interval
<i>Kyphosis</i>	Indicates whether the child has Kyphosis (1) or not (0)	Binary

Table 3.2: Kyphosis data set attributes

The goal of this prediction task is to use the *Age*, *Number* and *Start* attributes as inputs to predict whether a child has Kyphosis or not. This task is consequently one of classification. In order to find a good GANN model for this problem, the following methodology is used.

Methodology

With the first step of the interactive construction methodology, a GANN is created with a skip layer and one hidden neuron for each input. In the second step, a generalized linear model is fitted to give the initial estimates of the constant term α and the w_{0j} . Step 3 initializes the remaining three parameters in each hidden layer with random values from a normal distribution with variance equal to 0.1 and mean zero. In step 4, the full GANN model is fitted. In step 5, the fitted univariate functions that are overlaid on their partial residuals are analyzed. Three partial residual plots are created, one for each input variable, and must be inspected visually to determine the appropriate bias-variance trade-off. The partial residual plots of step 4 of the first iteration of the interactive construction algorithm are shown in Figures 3.8, 3.9 and 3.10.

The functions are presented as fitted splines that are overlaid on the partial residuals to help guide the modeller in determining the appropriate complexity of the univariate functions. If the univariate function is constant for the full range of input values and consequently does not contribute towards describing variation in the response, the spline will form a horizontal or near horizontal line. The input can be removed from the model in this case. A linear relationship between the input and the response is presented by a spline that forms a line with a substantial positive or negative slope. The input can be set to only a skip layer in these instances (e.g. Figure 3.10). A spline that forms a curve indicates a nonlinear relationship and can be modelled by one or more neurons in the hidden layer of the input (e.g. Figure 3.8). If, however, too many neurons are added, the univariate function will have a high variance and low bias (e.g. Figure 3.9).

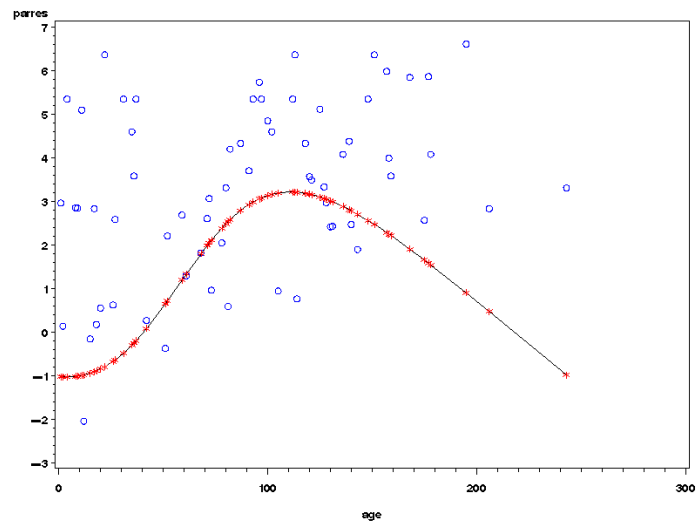


Figure 3.8: Partial residual plot of *Age*

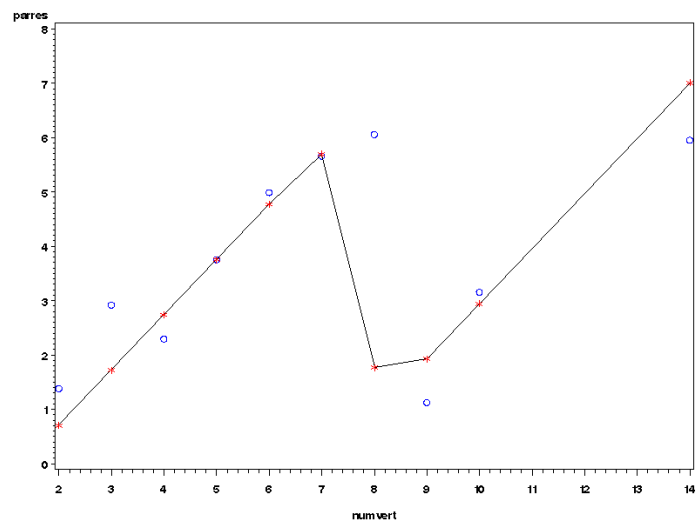


Figure 3.9: Partial residual plot of *Number*

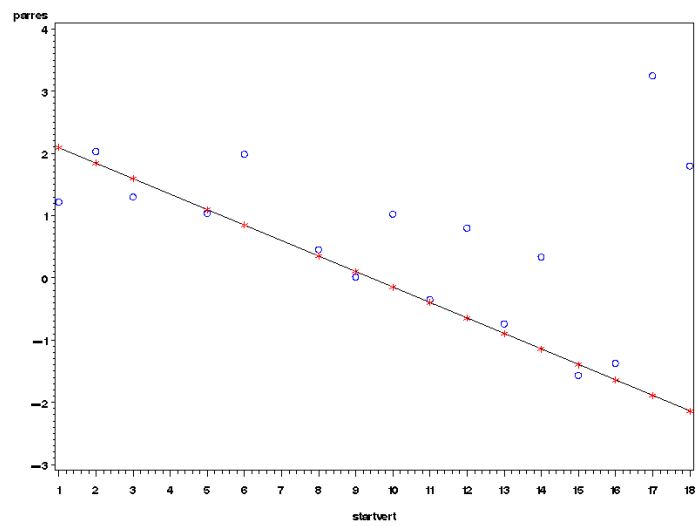


Figure 3.10: Partial residual plot of *Start*

In step 6, some architectural changes are made after inspection of the partial residual plots. Sometimes, several iterations of steps 4, 5, and 6 are required to make the changes that result in the best GANN model. A total of 4 iterations were needed to find the best GANN model for this data set. In this model, the *Age* variable had a skip layer and two hidden neurons, while the *Number* and *Start* variables had only skip layers. The partial residual plots of this final GANN model are shown in Figures 3.11, 3.12 and 3.13.

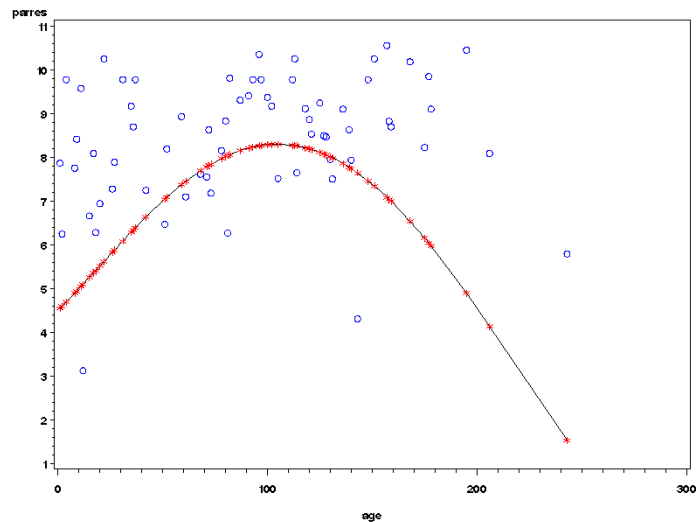


Figure 3.11: Partial residual plot of *Age*

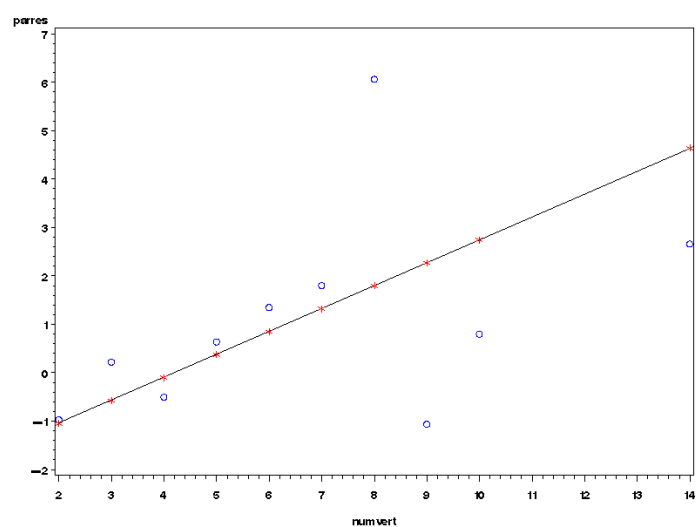


Figure 3.12: Partial residual plot of *Number*

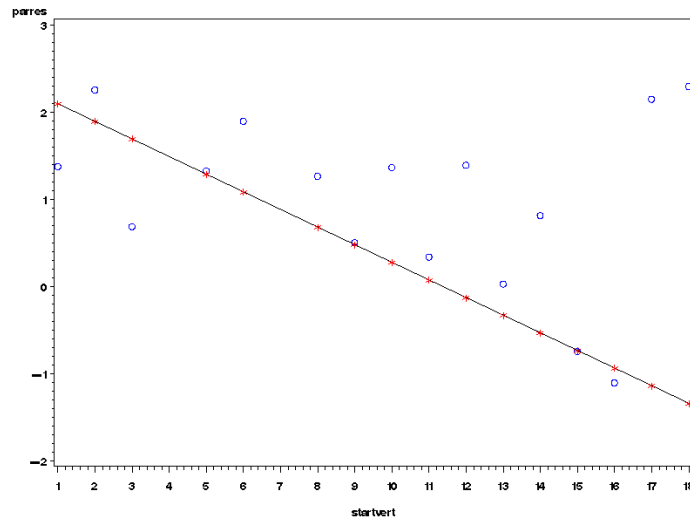


Figure 3.13: Partial residual plot of *Start*

Results

The GANN models that are created with each iteration of the interactive construction algorithm can be compared by means of the Schwarz Bayesian criterion (SBC) and mean squared error (MSE). The SBC and MSE value of each of the iterations are shown in Table 3.3.

Iteration	SBC	MSE
1	107.338	0.118
2	92.519	0.114
3	95.805	0.144
4	98.709	0.123

Table 3.3: Kyphosis results

According to this table, the best GANN model (in terms of the SBC value) is the one that was created at iteration 2, with an SBC value of 92.59 (smaller is better). The best model that was found by examining the partial residual plots, is that of iteration 4, which has an SBC value 98.71. The motivation behind the use of the SBC model selection criterion is described in detail in Section 3.5.2.

Conclusions

Human judgement is needed to interpret the partial residual plots when GANNs are constructed interactively. This can become a time consuming and daunting task when there are a large number of variables. Human judgement is also subjective, which might result in the development of models that are suboptimal. In the next section, an objective approach is discussed that incorporates a formal measure of fit into the process. As a result, an automated method can be used that is based on the search for models by using model selection

criteria. With this new approach, partial residual plots are not used primarily for model building, but as a tool to give insight into the models that were constructed. The modeller also have more time to interpret the results, since no human interaction is needed while building the GANN models.

3.5 The automated construction methodology

Even though neural networks are successfully applied to a number of prediction tasks, there are still several unresolved issues in neural network model building (Du Toit, 2006). One of the biggest issues is how to choose an appropriate network architecture for a specific prediction problem. In traditional linear prediction problems, model selection is a nontrivial issue, but in nonlinear models such as neural networks, it is an especially tricky issue.

To help overcome some of the issues that are faced when constructing GANNs, an automated approach to the construction of GANNs is considered in this section. The method is objective and relies solely on a model selection criterion for model selection. As a result, no human interaction is needed while searching for the best model. In order to describe the automated construction algorithm, some terms need to be defined.

3.5.1 Definition of terms

These definitions are illustrated with the example GANN of Figure 3.14.

Definition 3.1 (Neural network) *A neural network is an arrangement of many simple processing elements. These elements work in parallel and the function is determined by connection strengths, network structure, and the processing performed at computing elements or nodes (also called neurons) (DARPA, 1988).*

Definition 3.2 (GANN sub-architecture) *A specific input's neural network structure.*

Definition 3.3 (GANN sub-architecture identifier) *The symbol that is used to denote a specific input's sub-architecture.*

In Table 3.4 (De Waal and Du Toit, 2011), ten standard sub-architecture identifiers are listed. These identifiers proved to be adequate when automating the construction of GANNs.

Definition 3.4 (GANN architecture identifier) *The list of GANN sub-architecture identifiers $[identifier_1, identifier_2, \dots, identifier_k]$ which refers to a specific GANN model's architecture that has k inputs, x_1, x_2, \dots, x_k , where the sub-architecture of input x_i , with $i = 1, 2, \dots, k$ is referred to by $identifier_i$.*

An example of a GANN architecture identifier that represents a GANN with five inputs is: [5,1,3,0,2]. This list represents a GANN architecture where the first input has an MLP with a skip layer and 2 hidden neurons. The

second input has only a skip layer. The third input has an MLP with 1 hidden neuron and a skip layer. The fourth input is removed and the final input has an MLP with 1 hidden neuron and no skip layer. This example GANN architecture is shown in Figure 3.14.

Definition 3.5 (GANN architecture) *A complete GANN model that is formed from a combination of GANN sub-architectures.*

Definition 3.6 (GANN sub-architecture identifier function) *The GANN sub-architecture identifier for a particular GANN model and input x_i is returned with the function $sub(x_i)$.*

From Figure 3.14, the GANN sub-architecture identifier function produces the following results: $sub(x_1) = 5$, $sub(x_2) = 1$, $sub(x_3) = 3$, $sub(x_4) = 0$ and $sub(x_5) = 2$.

Definition 3.7 (GANN sub-architecture space) *The set that contains all the possible GANN sub-architectures.*

As shown in Table 3.4, the GANN sub-architecture space is $\{0,1,2,3,4,5,6,7,8,9\}$. The GANN sub-architectures are restricted to these neural network structures. The reason for this restriction is to prevent the development of a model that is too complex and which will consequently overfit the data. This restriction also helps to decrease the number of possible models in the search space.

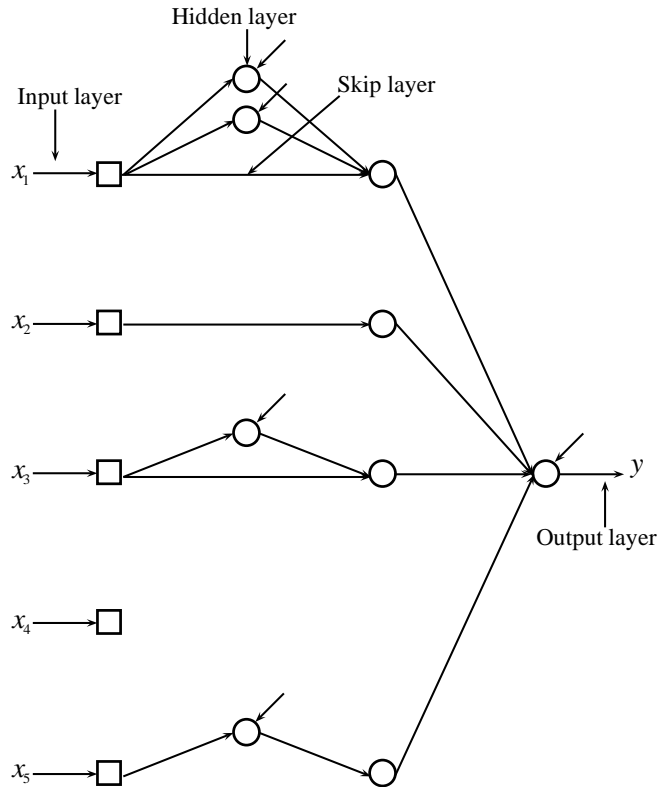


Figure 3.14: Example GANN model with five inputs

GANN sub-architecture symbol	GANN sub-architecture description
0	Input is not used in the model
1	MLP with only a direct connection
2	MLP with only 1 hidden neuron
3	MLP with a direct connection and 1 hidden neuron
4	MLP with only 2 hidden neurons
5	MLP with a direct connection and 2 hidden neurons
6	MLP with only 3 hidden neurons
7	MLP with a direct connection and 3 hidden neurons
8	MLP with only 4 hidden neurons
9	MLP with a direct connection and 4 hidden neurons

Table 3.4: GANN sub-architecture symbols

The automated construction algorithm uses the GANN sub-architectures that are defined in Table 3.4 to automate the interactive construction algorithm of Potts (1999). In order for the automated construction algorithm to select the best model, a model selection criterion is used. In the next section, model selection criteria is considered.

3.5.2 Model selection

The modelling of nonlinear relationships has experienced growing interest lately (Du Toit, 2006). A variety of test procedures that detect nonlinearities have also been developed. However, the uncovering of nonlinearities is not enough when prediction is the aim of the analysis. An adequate nonlinear model is necessary to describe these nonlinearities. Unfortunately for many applications, neither the correct functional form of the model nor the relevant input variables are suggested by the appropriate theory in the process of model building.

In this section, model selection strategies are discussed. These strategies are based on statistical concepts and for models like GANNs, a statistical point of view is particularly important because of the lack of knowledge regarding an adequate functional form of the underlying model. A systematic comparison of statistical selection strategies for neural network models is provided by Anders and Korn (1999). They also consider the concepts of information criteria, hypothesis testing and cross-validation methods. The conclusion that statistical analysis should become an integral part of neural network modelling was reached when they discussed the application of these three methods to neural networks.

A good model is one that will fit the data set well and if more variables are added to the model, the apparent fit becomes better. Model selection aims, among other things, to balance the increase in fit against the increase in model complexity. A better defining quality of a good model is possibly the performance of the model on unseen data from the same process. A model should fit any data set which arises from a process just as well as it fits any other data set that originates from the same process. Overfitting may occur when the model is too

complicated. When this happens, the model may fit the current data set well, but may fit subsequent data sets poorly. Underfitting may occur, on the other hand, when a model is too simple and does not fit any of the data sets well.

The data analyst must select some appropriate model from a set of models that were generated by collecting data after a probabilistic model had been proposed for the experiment. There may, however, be more than one definition of “appropriate”. The use of model selection criteria is one way to select the most appropriate model. However, certain model selection criteria performs best for specific model types and, as a result, there is no single model selection criterion that will always be better than another.

There are two broad types of model selection approaches (Du Toit, 2006). The first is out-of-sample model selection by means of cross-validation. The second is in-sample model selection that relies solely on a certain model selection criterion for model selection. In this section, out-of-sample model selection is considered, followed by a historical overview of the most prominent in-sample model selection criteria. A discussion of the two opposing views on in-sample model selection is then considered and finally the SBC, one of the most widely used in-sample model selection criteria, is considered. The latter criterion is utilized in this study.

Out-of-sample model selection

For out-of-sample model selection, a certain model selection criterion is utilized, together with cross-validation, to determine the proficiency of the model. The latter technique, also known as the holdout method, uses part of the data for training the model, part of the data for testing the model and, if required, part of the data for validation (Witten and Frank, 2005). Common practice is to use a third of the data for out-of-sample testing and the remaining data for training. This can, however, lead to a training or testing data set that is not representative of the full data set. Although there are in general no way to tell if a sample is representative of the full data set, there is a check that can be performed that may help to tell if the sample might not be representative at all. This is done by checking whether each class in the full data set is represented in the sample data sets in about the right proportions. Stratification is a process that divides the full data set into two or more subsets by selecting random instances and ensuring that each class is represented about equally in each subset. For better results, stratification can be used with K -fold cross-validation. With the latter technique, the data set is split into K approximately equal partitions. If, for example, 3-fold cross-validation is used, then the data set will be split into three partitions. Each partition is in turn used for testing, while the remaining two partitions are used for training. Two thirds of the data set is thus used for training, while the remaining third is used for testing. The procedure is repeated three times, so that each partition has been used exactly once for testing. According to Witten and Frank (2005), given a single fixed sample of data, stratified 10-fold cross-validation is the standard way to predict the error rate of a learning technique like neural networks. For this reason, 10-fold cross-validation was utilized to determine the accuracy of the models under consideration. To check the accuracy of the model using cross-validation, the average validation error (VAVERR) is used. This error measurement is chosen to enable a comparison between GANNs and MLPs. The automated construction algorithm for GANNs, described in Section 3.5.3, uses the VAVERR value for out-of-sample model selection. Consequently,

the VAVERR criterion is also utilized for out-of-sample model selection with MLPs. To determine the accuracy of a model with K -fold cross-validation, the VAVERR value is averaged over the K -folds.

In general, this method of out-of-sample model selection and testing is quite effective when it comes to stopping the tendency of neural networks to overfit the data (Du Toit, 2006). It has, however, some limitations. First, it requires a fairly large data sample size and second, splitting the data set may result in subsets that do not represent the full data set accordingly. Finally, the variability of the estimates may also increase by splitting the data set (Faraway, 1992). In-sample model selection criteria are also used in this study. A historical overview of this error measurement is considered next.

Historical overview of in-sample model selection criteria

Univariate and multiple regression models were the focus point in the past of much of the research on model selection criteria (Hurvich and Tsai, 1989). The adjusted R-squared R_{adj}^2 was the first model selection criterion that was widely used. It still appears in many regression literature today. When a variable is added to the model, the R^2 always increases. Without regards to the relative contribution to model fit, the R^2 will always recommend additional complexity, as this will increase its value. To attempt to correct for this always-increasing property, the R_{adj}^2 was introduced. The most notable model selection criteria research that was done in the late 1960s and early 1970s was Akaike's FPE (Akaike, 1969) and Mallow's Cp (Mallows, 1973). The Akaike information criterion (Akaike, 1974) appeared in the 1970s and was based on the Kullback-Leibler discrepancy (Kullback and Leibler, 1951). In the late 1970s, much research on information theory appeared with the proposal of the Bayesian information criterion (BIC) (Akaike, 1978), the Hannan and Quinn (HQ) (Hannan and Quinn, 1979), GM (Geweke and Meese, 1981), the Schwarz information criterion (SIC) (Schwarz, 1978), and FPE α (Bhansali and Downham, 1977). An improved small-sample unbiased estimator of the Kullback-Leibler discrepancy, called AICc, was created by Hurvich and Tsai (1989) in the late 1980s by adapting the results of Sugiura (1978). The AICc proved itself to be one of the best model selection criteria.

Two model selection paradigms

The notion of asymptotic efficiency as a paradigm for selecting the most appropriate model appeared in the literature of 1980. On the other hand, associations with the notion of consistency included the SIC, HQ, and GM model selection criteria. The philosophies of efficient and consistent model selection criteria are considered next.

Efficient criteria

In regression and time series, the assumption that the true model has infinite dimensions, or that the true model is not in the set of candidate models, is usually made. Given a set of finite dimensional candidate models, the objective is to choose one model that best approximates the true model. The appropriate choice is assumed to be the model nearest to the true model. A well-defined distance or information measure is needed to assess the model that is "closest". A model selection criterion that is said to be asymptotically efficient, is one that picks

the model with minimum mean squared error distribution in large samples (Shibata, 1980). The AIC, AICc, Cp, and FPE are all examples of asymptotically efficient models. Models that are based on efficiency are preferred by researchers when they believe that all the important variables cannot be measured or that the system under consideration is infinitely complicated. The improvement of efficient (“correct”) criteria for small-samples has been the focus of much research. AICc may be the best known correct version (Sugiura, 1978; Hurvich and Tsai, 1989).

The most significant property of a candidate model is sometimes its predictive ability. An example of an early model selection criterion that modelled mean squared prediction error is PRESS (Allen, 1974). Another model selection criterion that selects models that make good predictions is Akaike’s FPE. FPE and PRESS are both efficient. It is also worth noting that asymptotic efficiency and prediction are related (Shibata, 1980).

Consistent criteria

It is assumed by many researchers that the set of candidate models includes the true model and thus is of finite dimension. To identify the true model correctly from the list of candidates is thus the objective of model selection. A model selection criterion is said to be consistent if it identifies the correct model asymptotically with a probability of one. The HQ, GM, and SIC are examples of consistent criteria. The researcher believes, in this case, that the list of all significant variables can be identified, since adequate knowledge exists about the physical system under consideration and that all variables can be measured. These are strong beliefs to many statisticians. They may, however, hold in fields where there are large bodies of theories to justify such beliefs, like the field of physics. With these theories, it is assumed that the true model belongs to the set of candidate models.

Asymptotic arguments are used to derive many of the consistent model selection criteria. The fact that the consistent criteria do not estimate some distance function or discrepancy is part of the reason why more work has been dedicated to find improvements to efficient criteria rather than to consistent criteria.

The choice between efficiency or consistency is highly subjective and there is little agreement on which philosophy is better. The choice depends on the assessment of the complexity and measureability of the modelling problem of the individual researcher.

The most widely used in-sample model selection criteria are criteria that penalize large models that tend to overfit, such as the information-based criteria AIC and SIC. It was decided to use the SIC as in-sample model selection criterion for this study. In the next section, this criterion is discussed.

Schwarz information criterion

The Schwarz information criterion (SIC or SBC) was developed from a Bayesian perspective, where each model has equal prior probability and, given the model, the parameters have very vague priors. It was assumed that simple prediction, rather than scientific understanding of the process or system under consideration, was the goal of the SBC-selected model.

In the literature, a number of different forms of the SBC have been suggested. The generic SBC definition

(Burnham and Anderson, 2002) is

$$\text{SBC} = -2\log(\mathcal{L}(\hat{\theta}|y)) + K\log(n), \quad (3.25)$$

where, given the data y and K the number of estimable parameters in the approximating model, $\log(\mathcal{L}(\hat{\theta}|y))$ represents the natural logarithm of the likelihood function of the parameter vector.

For the special case of the Gaussian error model, the SBC is defined as

$$\text{SBC} = n\log(\hat{\sigma}^2) + K\log(n) \quad (3.26)$$

with

$$\hat{\sigma}^2 = \frac{\sum \hat{\epsilon}_i^2}{n} \quad (\text{the MLE of } \sigma^2), \quad (3.27)$$

where the estimated residuals for a specific candidate model is represented by ϵ_i and K represents the total number of estimated regression parameters, including the intercept and σ^2 .

Schwarz (1978) and Rissanen (1978) both developed (3.25) independently. It was showed by Rissanen (1978) that a consistent estimate of the order of an AR model is produced by the SBC. If the true data-generating model belongs to the finite-parameter family under consideration, then the SBC is a consistent model selector. Models that tend to underfit is selected by the SBC for exponential families, if the previous assumption does not hold (Haughton, 1989).

The SBC is sometimes abbreviated accidentally as BIC. However, the penalty term of the Bayesian information criterion (BIC) differs from that of the SBC. The BIC is defined as

$$\text{BIC} = -2\log(\mathcal{L}(\hat{\theta}|y)) + K + K\log(n). \quad (3.28)$$

The creation of a search space of possible GANN models, together with an effective search procedure to find the best model by using some model selection criterion, form the basis for the automation of the interactive construction algorithm. In the next section, the automated construction algorithm is considered.

3.5.3 The automated construction algorithm

A criterion has to be defined to rank the models from “good” to “bad” for the automated construction algorithm to be effective. The automation of the interactive construction algorithm is made possible with a model selection criterion that is used to evaluate the predictive accuracy of the models. When a validation data set is present, models are tested on the validation set, which allows the algorithm to perform cross-validation. Feature selection (Guyon and Elisseeff, 2003; Blum and Langey, 1997) is also performed automatically by the algorithm. The automated construction algorithm consists of 7 steps. These steps are shown in Algorithm 3.2.

A best-first search strategy (Rich and Knight, 1991) is utilized by the automated construction algorithm (De Waal and Du Toit, 2011). The state space search problem can be formulated as follows:

- States: Any GANN model that is represented by a valid string of digits that represents the GANN sub-architectures.
- Initial state: The sub-architecture string that represents a linear GANN model.

1. A GANN model with a direct connection for each input is created. The univariate functions are initialized to

$$f_j(x_{ji}) = w_{0j}x_{ji}. \quad (3.29)$$

Each input has one parameter now.

2. Initial estimates of the constant term α and w_{0j} are obtained by using a generalized linear model.
3. The full GANN model is fitted. The model is evaluated by using the model selection criterion. To indicate that the model is available for expansion, the *expanded* flag is set to false. Finally, the model is denoted as the root of the tree.
4. Where the *expanded* flag is false, a search for the best GANN model m is performed by using the model selection criterion. If such a model is found, the *expanded* flag is set to true to indicate that the model is expanded. If a model cannot be found with the *expanded* flag set to false, the tree is searched for the best model. This model is then reported and the program terminated.
5. For each input x_i of m (the model identified in step 4): If $1 \leq \text{sub}(x_i) \leq 9$, then a GANN model n is created with the sub-architecture of x_i set to $\text{sub}(x_i)-1$ and the remaining sub-architectures of m are left unchanged. A check is performed to determine whether n has previously been created in the tree and, if not, then n is evaluated with the model selection criterion and added as a child node to m . Finally, the *expanded* flag of n is set to false.
6. For each input x_i of the model m : If $0 \leq \text{sub}(x_i) \leq 8$, then a GANN model n is created with the sub-architecture of x_i set to $\text{sub}(x_i)+1$ and the remaining sub-architectures of m are left unchanged. A check is performed to determine whether n has previously been created in the tree and, if not, then n is evaluated with the model selection criterion and added as a child node to m . Finally, the *expanded* flag of n is set to false.
7. Go back to step 4.

Algorithm 3.2: Automated construction algorithm

- Successor function: Any valid sub-architecture string with one digit changed.
- Goal test: When the given time runs out, or after the whole search space has been exhausted, the best GANN model that was found is used.
- Path cost: Since only the best GANN model that was found is of importance, the path to that model has no use and, consequently, there is no path cost.

The most promising model of those that were created so far is selected (step 4) at each step of the best-first search process. To achieve this, the model selection criterion value of each generated model is considered. Steps 5 and 6 are then applied to expand the chosen model in order to generate its successors. All the newly created models are added to the set of models that were created so far. The most promising model is chosen again for expansion and the process continues. The paths followed by the best-first search is influenced by the order in which the sub-architectures are defined in Table 3.4. Usually, the size of the search space is reduced when a problem is solved by choosing a subset of the sub-architectures in Table 3.4.

To implement the best-first search algorithm, two lists of nodes are needed:

- Open: models that have been created, but not expanded, with the model selection criterion value that were calculated. The *expanded* flag are set to *false* for these nodes.
- Closed: models that have been expanded. The *expanded* flag are set to *true* for these nodes.

With best-first search, the merit of each model that is created is estimated with a heuristic function. It is for this purpose that the model selection criterion is used and it allows the algorithm to search more promising paths first.

The automated algorithm's actual operation is simple, as it proceeds in steps. At each step it picks the most promising of the models that have been created so far without being expanded. The successors of a chosen model are created and a check is done to determine if any of the models have been created previously. A heuristic function is then applied to the successors and the successors are added to the list of open models. By performing this check, it is guaranteed that each model only appears once in the tree, even though many models may point to one model as a predecessor. To guarantee that there are a finite number of models that must be searched, and for efficiency reasons, a finite number of sub-architectures are picked before trying to solve any real problem.

Since best-first search always moves forward from the model that seems closest to the goal model value, the paths that were found by the best-first search are likely to be shorter than those found with other methods (Winston, 1992). In the case of the automated construction algorithm, the goal node is the one with the lowest model selection criterion value. This is a complete search strategy, since, given the finite number of sub-architectures, all possible models will have been constructed when the algorithm terminates.

To further enhance the automated construction algorithm, De Waal and Du Toit (2011) added a multi-step expansion feature. This feature is considered next.

Multi-step expansion

So far, the automated construction algorithm only allows for one change to the GANN architecture in each child node. With this restriction, each change in the architecture requires one iteration of the algorithm. To arrive at the best GANN architecture, the number of successor functions that need to be applied become unrealistically large for a large number of variables. The multi-step expansion relaxes this restriction.

It could be sensible to allow multiple changes when two or more child nodes have better model selection

criterion values than that of the parent node. The following can be done to accomplish this:

Identify all the child nodes that have better model selection criteria values than their parent node. This corresponds to nodes 2, 4 and 5 in the SO_4 data set example of Figure 3.15. Note that the three values of each node indicate the order of creation, GANN architecture and model selection criterion value respectively. All the changes that were identified in the children are then used to create a new child node (node 6) from the common parent node. Nodes 2 and 4 denote the removal and addition of a neuron to the sub-architecture of the first input. The change in the best child node (node 4) is used when two or more child nodes (nodes 2 and 4) have better model selection criterion values than the parent for the same change in the sub-architecture. Node 5 represents the only change to the sub-architecture of the second input that has a better criterion value. Consequently, this sub-architecture is utilized as the sub-architecture of the second input of the newly created node (node 6). There is a good chance that all the changes that are applied collectively will be worthwhile, since all the changes that were identified, are worthwhile independently of each other. Multiple changes are thus permitted during an iteration of the algorithm.

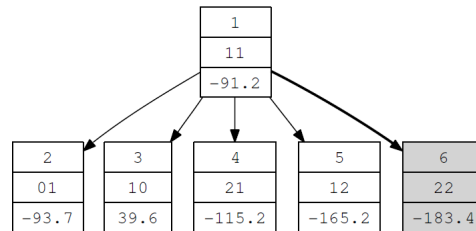


Figure 3.15: SO_4 multi-step expansion

The multi-step also occurs in the interactive construction algorithm. Multiple changes to the GANN architecture can be made during each iteration of the interactive construction algorithm that is based on the inspection of partial residual plots. The multi-step expansion thus models one iteration of the interactive construction algorithm.

The multi-step expansion decreases the number of iterations that are needed for the automated construction algorithm to converge. This heuristic was then further enhanced by De Waal and Du Toit (2011), and is considered next.

Improved multi-step expansion

There are no restrictions placed in step 6 of the interactive construction algorithm on the number of hidden neurons that may be added to or removed from a hidden layer. For efficiency reasons, the automated construction algorithm restricts the successor function to slightly more complex or slightly less complex sub-architectures. The following can be done to remove this restriction:

Apply another successor function to the generated child node if that child node's model selection criterion value is better than that of the parent node and then examine the newly created node's model selection criterion. Repeat the process of creating additional child nodes if the model selection criteria values improve, until the model selection criterion value of the last node becomes worse. The best node that is created with this recursive

process is considered to be the result of the applied successor function. This improved multi-step expansion permits the move in the sub-architectures to non-adjacent rows of Table 3.4. An example of such a step from the SO_4 data set is shown in Figure 3.16, where the grey coloured node represents the best node.

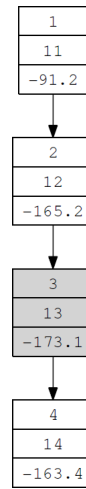


Figure 3.16: SO_4 improved multi-step expansion

This strategy has a major drawback, even though it is sound and may decrease the number of iterations that are needed in the automated construction algorithm. The drawback is that enumeration partly replaces intelligent search and is, consequently, counter productive. Based on the inspection of partial residual plots, it is also very improbable that even an experienced modeller will be able to make such big changes, since it is hard to judge the complexity versus degrees of freedom trade-off. The strategy can be approximated as follows:

Only an increase in the complexity or number of hidden neurons in the GANN architecture are considered, as the starting point of the search in the automated strategy is a linear model. Rather than re-evaluating each new child node, which requires the training of a neural network, the first child node's (node 2 in Figure 3.16) model selection criterion value can simply be adjusted to accommodate the extra degrees of freedom (K) that is relevant to each additional child node (nodes 3 and 4 in Figure 3.16), while keeping the maximum likelihood estimates constant. This produces an approximation to the model selection criterion values of the child node and can be calculated immediately without any optimization. Note that this approximation can be applied to the SBC model selection criterion, but will not work with cross-validation. During each evaluation of each new child node, the model selection criterion will become worse and at some point will be worse than that of the parent node. This results in an approximation to the maximum possible moves in the GANN model's sub-architecture.

The improved multi-step expansion further decreases the number of iterations that are needed to reach the best GANN model. De Waal and Du Toit (2011) devised a final heuristic which performs an intelligent guess of the best GANN architecture. The search for the best GANN model then commences from this intelligent start architecture. This heuristic can also decrease the number of iterations that are performed by the automated construction algorithm.

Intelligent start

With the basic automated construction algorithm (Algorithm 3.2), search is initiated from a GANN model where each input has a sub-architecture identifier of 1. The intelligent start replaces the initial starting model with a more promising one. An analysis of the results from a stepwise regression is used to determine the intelligent start architecture.

Stepwise regression is used for variable selection. The goal is to remove the input variables that do not have a significant effect on the output variables and to keep the variables that have a significant effect on the output (Jiao and Li, 2010). With stepwise regression, all the variables are included in the initial model. Stepping is then performed on this model by adding or removing variables according to the stepping criterion. Due to the complexity of the relationships between variables and outputs, the importance of a specific variable may change when other variables are added to the model. After each new variable is added, a test is thus performed to determine if some variables can be removed without having a significant increase in the residual sum of squares. The stepwise regression will stop when the model has been optimized or when a specified number of steps have been reached.

A GANN is reformulated as a regression problem after it has been constructed with a skip layer and one neuron in the hidden layer for each input, and has been trained with the default optimization algorithm. Each skip layer and hidden layer are used as separate variables in the reformulated regression problem. Stepwise regression is then performed and the results are interpreted as a GANN model. This GANN model is then used as the new starting point (root node) for the search algorithm. The intelligent start function is presented in Algorithm 3.3. The automated construction algorithm with the implemented intelligent start and multi-step expansion techniques are shown in Algorithm 3.4. Note that Algorithms 3.3 and 3.4 are presented in the style of Luger (2005), who utilizes lists to manage the search tree.

The automated construction algorithm is the solution to the difficulty of constructing GANNs interactively and with the improvements that are made to the algorithm, the time taken to arrive at the best GANN model is drastically decreased. In the next section, the implementation of the automated construction algorithm is considered.

3.5.4 Implementation of the automated construction algorithm

The SAS® programming language is an assemblage of reporting, data management and analysis tools which are all integrated (SAS Institute Inc., 2005). The user can read and combine data files in many ways with the data management features. Simple frequency distributions, through to complex multivariate techniques, can be performed with the analysis capabilities of SAS®. Finally, the user can present data management and analysis results in a large number of formats with the reporting features.

The SAS® system is powerful, since it is integrated. The analysis and reporting components are able to use data that are handled by the data management facilities without the need to modify it. As a result, data formats and structures are of minimal concern to the user. The system can also be used throughout different computing environments. A SAS® program that has been developed on an IBM-compatible personal computer can be

begin

a GANN is constructed with one hidden neuron and a skip layer for each input V_j , call it

Guess with $f_j(x_{ji}) = w_{0j}x_{ji} + w_{1j}\tanh(w_{01j} + w_{ji})$;

fit the GANN model ;

for *each input variable V_j in the GANN model* **do**

$V_j_skip := w_{0j}x_{ji}$;

$V_j_hidden := w_{1j}\tanh(w_{01j} + w_{ji})$;

end

using the new variables V_j_skip and V_j_hidden , fit a stepwise regression model ;

for *each input variable V_j in the GANN* **do**

case *the selected regression model only included V_j_skip* **do**

remove the one neuron for variable V_j from *Guess*, consequently $f_j(x_{ji}) = w_{0j}x_{ji}$;

end

case *the selected regression model only included V_j_hidden* **do**

remove the skip layer for variable V_j from *Guess*, thus $f_j(x_{ji}) = w_{1j}\tanh(w_{01j} + w_{ji})$;

end

case *the selected regression model does not include V_j_skip nor V_j_hidden* **do**

remove variable V_j from *Guess* ;

end

end

return *Guess* ;

end

Algorithm 3.3: Intelligent start

used, with almost no alteration, on a mini-computer or mainframe. As a result of all these features, Du Toit (2006) chose the SAS® Macro Language to implement the automated construction algorithm. This implementation was named *AutoGANN* and was incorporated into the SAS® Enterprise Miner™ solution. By supporting all necessary tasks within one, integrated solution, Enterprise Miner™ streamlines the full prediction process from data access to model deployment, all while providing the flexibility for efficient workgroup collaborations.

The SAS® Macro Language has a procedure, PROC GAM, that uses the backfitting algorithm to fit GAMs. An array of powerful tools are provided by this procedure, which are based on nonparametric regression and smoothing techniques. However, this procedure is not implemented as a modelling node in Enterprise Miner™. *AutoGANN* fills this gap from a neural network perspective, since it is implemented as a modelling node in Enterprise Miner™. The implementation of the automated construction algorithm in SAS® provides a more user-friendly tool to the data analyst than PROC GAM.

```

begin
    start := Intelligent_Start ;
    open := [start] ;
    closed := [ ] ;
    while, open  $\neq$  [ ] and specified time not reached do
        remove the first state from open (leftmost state) and call it X ;
        begin
            crossover := [ ] ;
            threshold := heuristic value of X ;
            minimum := heuristic value of X ;
            for, each child of X do
                if, not open nor closed, contains the child do
                    a heuristic value is assigned to the child ;
                    the child is added to open ;
                    if, the child's heuristic value < threshold do
                        the child is added to crossover ;
                        minimum := the child's heuristic value ;
                    end
                end
            end
            if there are at least two states in crossover do
                a super child is constructed from crossover and called S ;
                if S  $\notin$  open and S  $\notin$  closed do
                    a heuristic value is assigned to S ;
                    if the heuristic value of S < minimum do
                        S is added to open ;
                    end
                end
            end
            end
            X is added to closed ;
            sort states in open by heuristic value (best leftmost) ;
        end
    return best state from open and closed by using the heuristic value ;
end

```

Algorithm 3.4: Updated automated construction algorithm

The Macro Language of Base SAS® allows for the design of meta-programs that create and execute other programs. With this powerful capability, AutoGANN can create GANN source code in real time and then analyze the predictive power of the models after the code has been executed. With AutoGANN, GANN models can be evaluated in a fraction of the time it would take an analyst to do it by hand.

AutoGANN description

The SAS® Macro Facility was used for implementing the AutoGANN system. The SAS® Macro Facility is a tool within the Base SAS® software that allows the use of macros (Carpenter, 2004). The macro facility generates source code and incorporates a macro processor to translate macro code into statements. These statements can be used by SAS® and the Macro Language. Communication with the macro processor is provided by the Macro Language. With the latter, the user can send information between DATA and PROC steps. The user can create SAS® code dynamically after the program has been submitted for execution. The Macro Language also enables the user to create flexible and generalizable code. A DATA step enables the programmer to perform a number of tasks which include:

- reading of raw data or other SAS® data sets;
- creating a SAS® data set;
- writing of reports; and
- writing to external files.

A PROC step invokes a SAS® procedure and is part of a SAS® program. There are four basic steps in AutoGANN, as indicated by Figure 3.17. These steps are considered next.

Initialize AutoGANN system

Enough main memory is reserved for the execution of the program with this first step. A consistency check is performed on the six parameters of AutoGANN: *Criterion*, *Start Architecture*, *Search Space*, *Partial Residual Plots*, *Time* and *Number of Models*.

The model selection criterion that is used by AutoGANN to evaluate different GANN models is set with the *Criterion* parameter. The SBC criterion for a least squares analysis, *SBC_dev*, is the default value. When the likelihood of the model must be used to calculate the SBC criterion, *SBC_like* must be selected by the user. The Akaike Information Criterion (Anders and Korn, 1999) can also be selected as the AutoGANN model selection criterion, with *AIC_dev* for least squares analysis and *AIC_like* for likelihood analysis. The average error on the validation data set, *Valid_ave_err*, is also included. The *Start Architecture* parameter sets the GANN architecture for the root node of the search tree. The default value is *Intelligent*, which specifies the intelligent start as discussed in Section 3.5.3. Another option is *Linear*, which indicates a linear model to start with. The *Search Space* parameter has a default value of 012345 and defines the sub-architecture space. The *Partial Residual Plots* parameter turns the creation of partial residual plots on or off. The default value for this parameter is

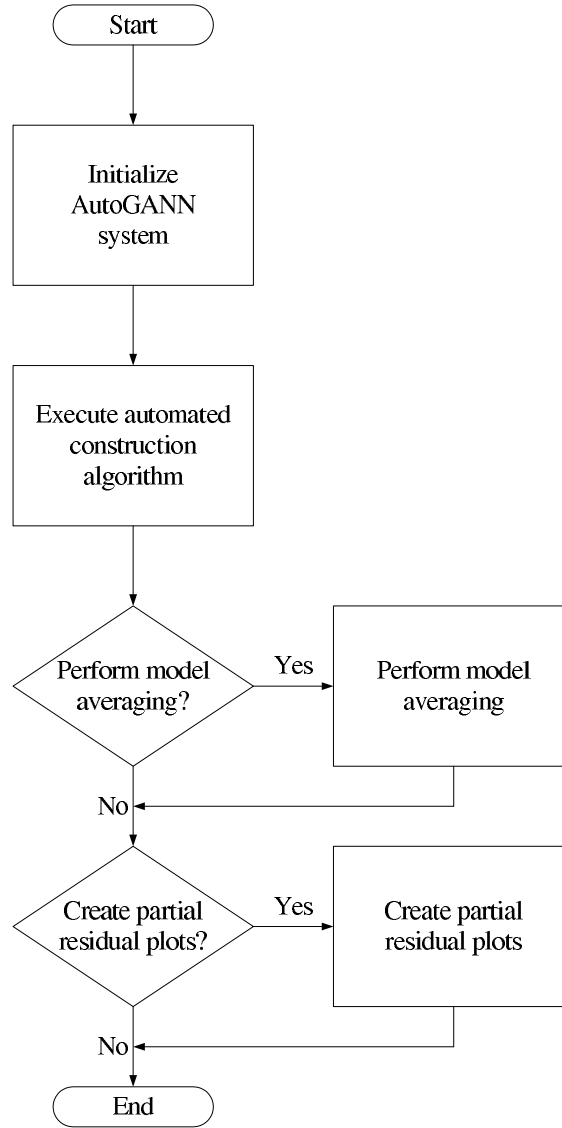


Figure 3.17: AutoGANN flowchart

Yes. To force the algorithm to stop after a certain length of time, the *Time* parameter can be set. The default value is *Initialize*, which causes the algorithm to stop after the root node has been created and evaluated. Other values range from 5 seconds to 2 days. The final parameter is the *Number of Models*. This parameter sets the number of models that are used for model averaging. The default value is 1. To find a more stable GANN model, the model averaging technique can be used to average the chosen number of best models found. This number can range from 1 to 10. Model averaging (Du Toit, 2006) is beyond the scope of this study and was not considered. System malfunctions may occur when there are inconsistencies in these parameters. This will cause the program to terminate and generate an error message.

The source code for a skeletal GANN model is generated after the input data set is analyzed. This skeletal GANN code can be configured to represent any GANN model in the search space. The automated construction algorithm modifies the skeletal GANN code to create different GANN models in the search tree.

Execute automated construction algorithm

The second step of the AutoGANN system is to execute the automated construction algorithm. A list data

structure that is stored inside the computer's main memory is used to maintain the tree of GANN models. The list is ordered so that the best GANN model, according to the model selection criterion, is always found at the start of the list. The results that are found by the AutoGANN system are exported to external files when the algorithm terminates, so that other programs can use it. The fit statistics are calculated and presented for the best GANN model. If a score data set exists, score code is generated and applied to it automatically. The only factor that limits the size of the problems that the AutoGANN system can solve, is the amount of available memory. Table 3.5 shows the activation and error functions that are implemented in AutoGANN.

Activation function	Link function	Target scale	Error function
Identity	Identity	Interval on $[-\infty, +\infty]$	Normal
Hyperbolic tangent	Inverse hyperbolic tangent	Interval on $[-1, +1]$	Normal
Exponential	Log	Nonnegative	Poisson
Multiple logit	Logit	Binary	Multiple Bernoulli

Table 3.5: AutoGANN activation functions

Perform model averaging

If the user requires a more stable GANN model than the one that is found with the automated construction algorithm, model averaging can be performed. This is done in step three.

Create partial residual plots

The last step of the AutoGANN system is to create partial residual plots. Partial residual plots of the combined GANN model are produced when model averaging is used, otherwise partial residual plots of the best GANN model that was found, are created. The method that was used by Potts (2000) to create these plots is extended by adding ticks to provide insight into the distribution of function values. Figure 3.21 shows an example of this improved partial residual plot.

Now that the inner workings of the AutoGANN system has been discussed, the AutoGANN user interface, which attempts to simplify the necessary user input and produce the desired results for each experiment, will be considered.

AutoGANN user interface

In order to keep the system as simple as possible, the user can only adjust the most important parameters. The adjustable settings for model selection are shown in Figure 3.18. A typical experiment, where the SO₄ data set is connected to the AutoGANN modelling node in Enterprise Miner™, is shown in Figure 3.19. The result screen which is displayed after the successful completion of an experiment is shown in Figure 3.20. The result screen consists of seven sub-screens, each giving different information:

- Score rankings overlay: Not applicable to this study.

- **Solution path:** This section shows information about nodes that are on the path from the root node to the node that represents the best GANN model that was found.
- **Model space statistics:** This section gives information about the search space, like the total number of models that were generated and the number of duplicates that were found.
- **Output:** This is the output text file that is generated by SAS®.
- **Populated search space:** Displays information about all the nodes that were generated, sorted from the best model that was found to the worst in terms of the model selection criterion.
- **Inputs:** The order, name and description of each input and sub-architecture identifier for each input as they were determined by the best model that was found.
- **Fit statistics:** The fit statistics part shows information, like the complexity and accuracy of the best model that was found.

Partial residual plots (Figure 3.21) can be inspected by choosing the *View* option. Finally, the system also generates a text file that can be used with the *Graphviz* tree draw program to create an image of the search tree (e.g. Figure 3.22).

In the next section, an example is given to illustrate the automated construction algorithm which utilizes Haberman's Survival data set (Frank and Asuncion, 2010).

	Property	Value
<input checked="" type="checkbox"/>	Model Selection Options	
<input type="checkbox"/>	Criterion	SBC_dev
<input type="checkbox"/>	Start Architecture	Intelligent
<input type="checkbox"/>	Link Architecture	0
<input type="checkbox"/>	Search Space	0123456789
<input type="checkbox"/>	Link Search Space	2103
<input type="checkbox"/>	Partial Residual Plots	Yes
<input type="checkbox"/>	Time	1 minute

Figure 3.18: AutoGANN settings screen

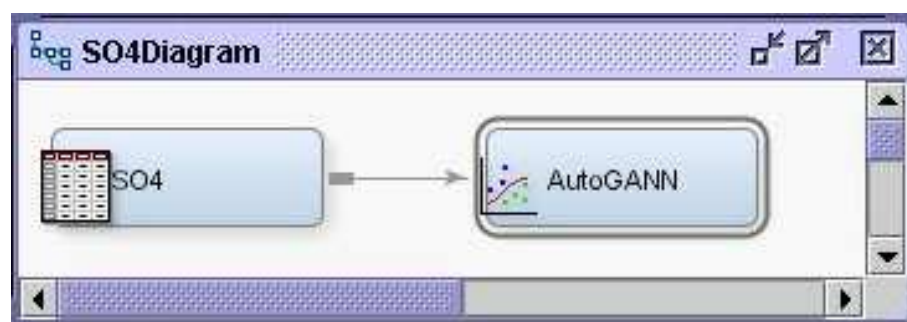


Figure 3.19: AutoGANN that is connected to the SO₄ data set

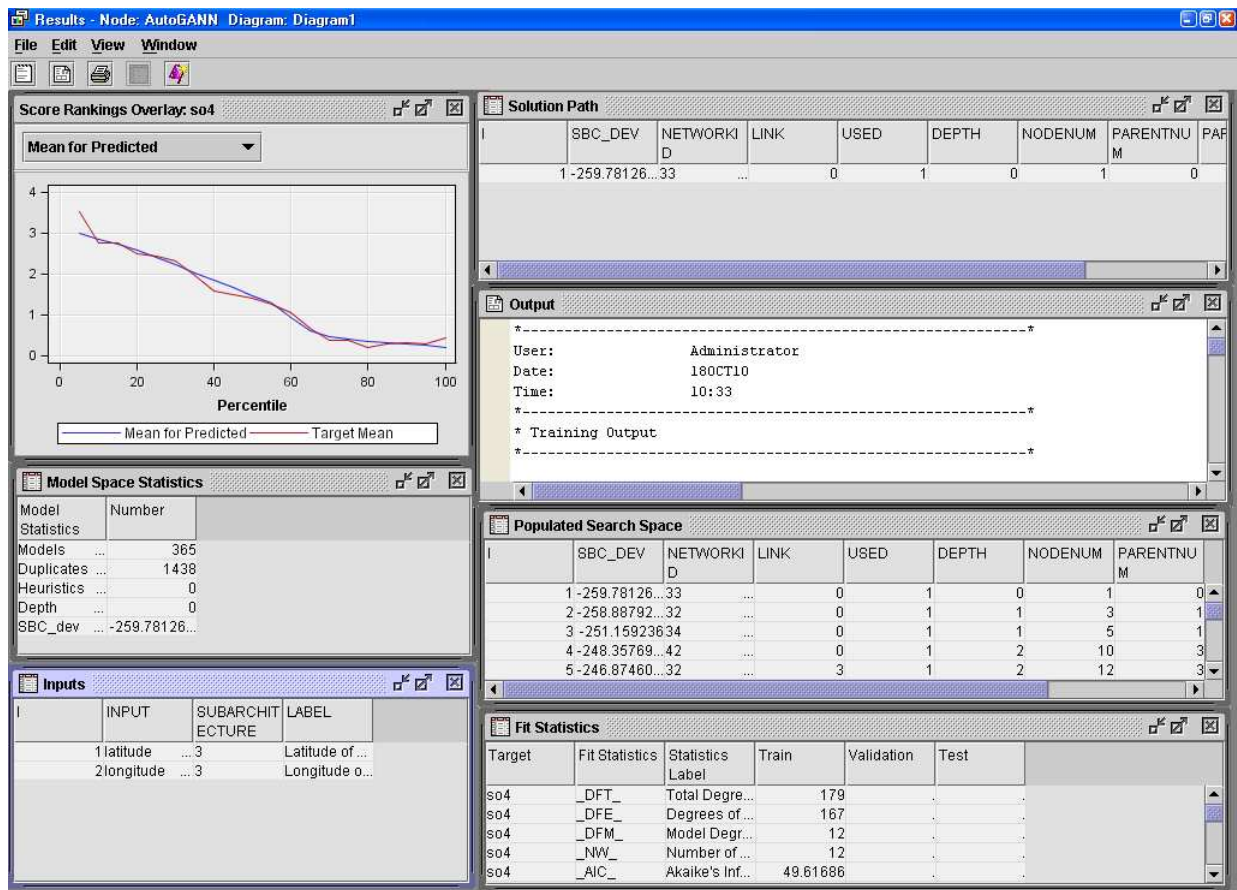


Figure 3.20: AutoGANN result screen

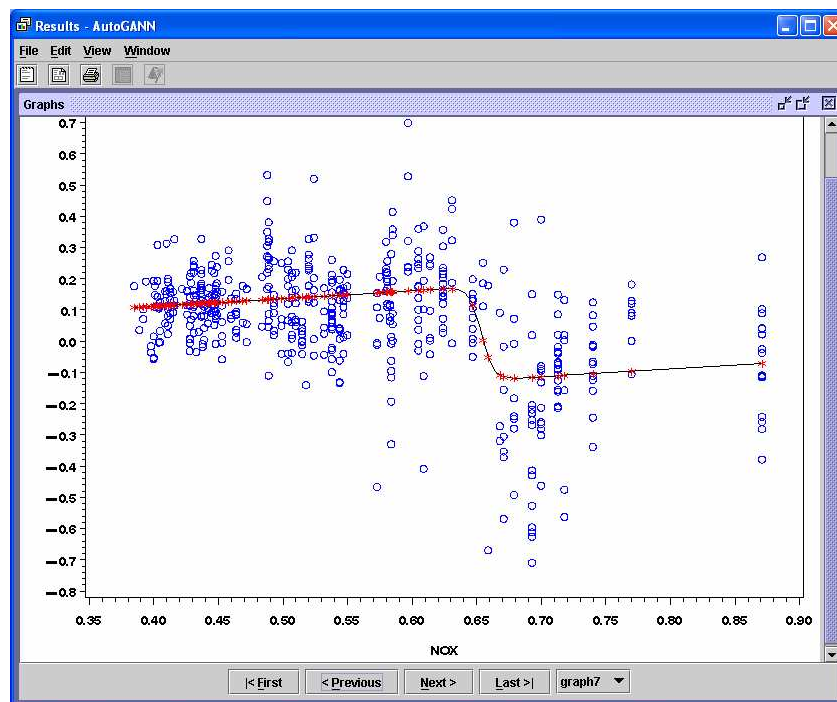


Figure 3.21: AutoGANN partial residual plot screen

3.5.5 Example

The instances of this data set represent cases from a study that was conducted between 1958 and 1970 at the University of Chicago's Billings Hospital. The study focussed on the survival of patients who had undergone surgery for breast cancer. The data set consists of 306 instances and has 4 attributes. These attributes are described in Table 3.6. This is a binary classification problem and the objective is to predict whether a patient will survive for 5 years or longer, or will die within 5 years.

Attribute	Description	Attribute scale
<i>Age</i>	The age of the patient when the operation took place	Interval
<i>Year</i>	The year when the operation took place (year-1900)	Interval
<i>Axillary</i>	The number of positive axillary nodes that were discovered	Interval
<i>Status</i>	Survival status of the patient. The status is 0 if the patient survived for 5 years or longer and 1 if the patient died within 5 years	Binary

Table 3.6: Haberman's Survival data set attributes

Methodology

The GANN sub-architecture space is limited to $\{0,1,2,3,4,5\}$ for this example, the SBC is used as model selection criterion and search time is restricted to 5 seconds. The GANN architecture is represented by $[x_1, x_2, x_3]$, where x_1 , x_2 and x_3 represents *Age*, *Year* and *Axillary* respectively.

In steps 1 to 3 of the automated construction algorithm, a GANN model is generated by using the intelligent start technique. This model is evaluated with the SBC criterion which produces a value of -516.093. The model has a GANN architecture of $[0,0,2]$. It is then set as the root of the search tree and represented by node number 1 in Figure 3.22. The *expanded* flag for this node is set to *false*.

The root of the search tree ($[0,0,2]$ model) is identified as the best unexpanded model that was created up to this point in step 4 and this node is denoted as node m . To indicate that this node is (being) expanded, the *expanded* flag of m is set to *true*.

Each input of model m is pruned by one sub-architecture level in step 5. Since the first two variables cannot be pruned any more from the architecture that was obtained from the intelligent start technique ($[0,0,2]$), the first child node, n , is created with an $[0,0,1]$ architecture. To determine whether node n has already been placed in the tree, a check is performed. Since this check is negative (node n does not already exist in the tree), the model is evaluated by the SBC criterion. The SBC value for this model is -515.308. The node's *expanded* flag is set to *false* and the model is added as node 2 to the tree (Figure 3.22).

Each input of model m ($[0,0,2]$) is grown by one sub-architecture level in step 6. This results in the creation of three new child nodes ($[1,0,2]$, $[0,1,2]$ and $[0,0,3]$). These nodes are added to the tree and numbered nodes 3, 4 and 5 respectively. Each of these nodes' *expanded* flag is set to *false*.

In step 7, the algorithm returns to step 4 to identify the best unexpanded node. This node is set to node m . In this example, this is node 3 ([1,0,2]), with an SBC value of -516.456. Step 5 (prune) and step 6 (grow) are repeated for each input of node m . Step 5 creates only one new child node [1,0,1], since [0,0,2] already exists in the tree. Step 6 creates three new child nodes, namely [2,0,2], [1,1,2] and [1,0,3].

The automated construction algorithm continues until the search space has been exhausted or the time limit has been reached. For this example, there are 215 models in the search space (6 possible sub-architectures and 3 inputs, $6^3 - 1 = 215$)¹. The AutoGANN system created 30 models and was stopped by the time limit of 5 seconds.

Results

The best model that was found, was reported as the third model that was created ([1,0,2]) with an SBC value of -516.456. The accuracy, as measured by the MSE value of this model, is 0.180485. The full search tree that was created in the time limit of 5 seconds is shown in Figure 3.22. The partial residual plots for variables x_1 (Age) and x_3 (Axillary) are shown in Figures 3.23 and 3.24 respectively.

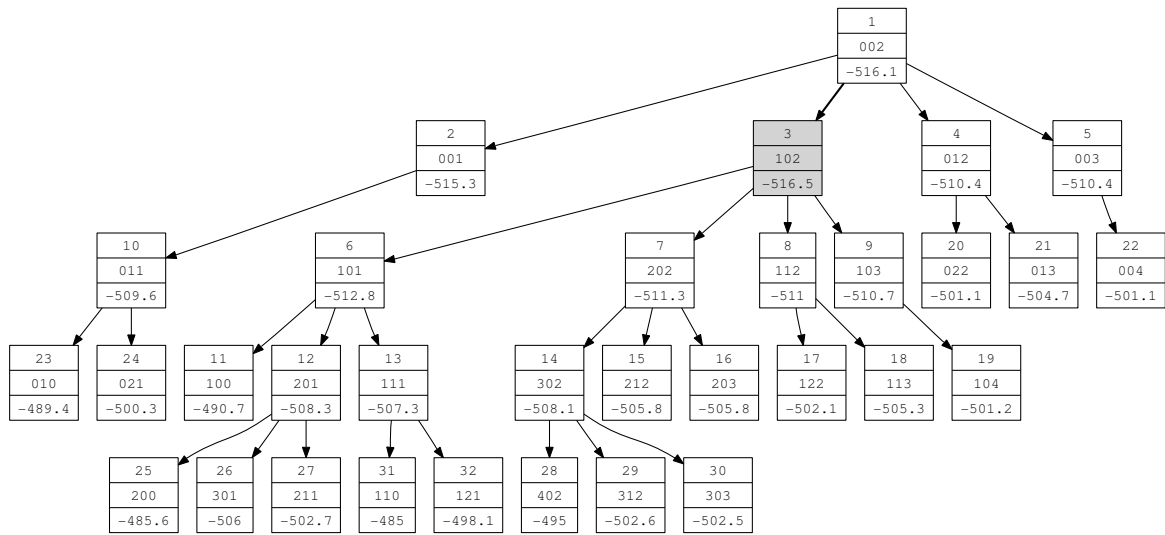


Figure 3.22: AutoGANN search tree for Haberman's Survival data set

Conclusions

This algorithm is efficient and fast. In the time limit of 5 seconds, the algorithm created and evaluated 30 models. This would be an impossible task to achieve by hand in 5 seconds by even the best modeller. The results are also not subjective to human judgement and are objectively obtained by evaluating the SBC values.

In the next section, a conclusion to this chapter is presented.

¹Note that the [0,0,0] architecture is not allowed.

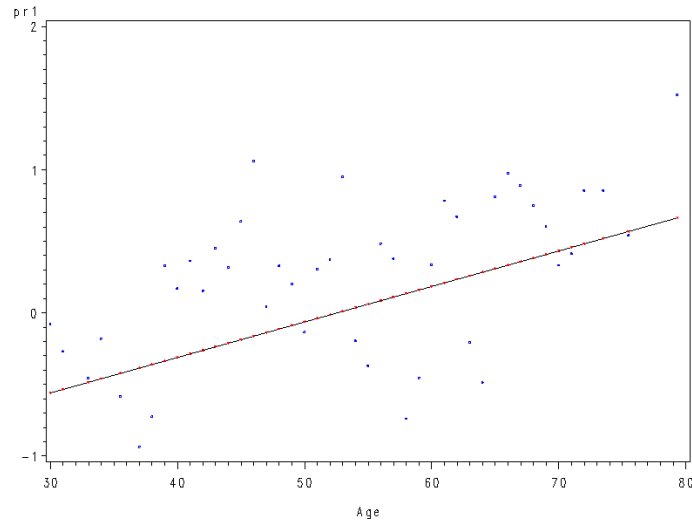


Figure 3.23: Partial residual plot of *Age*

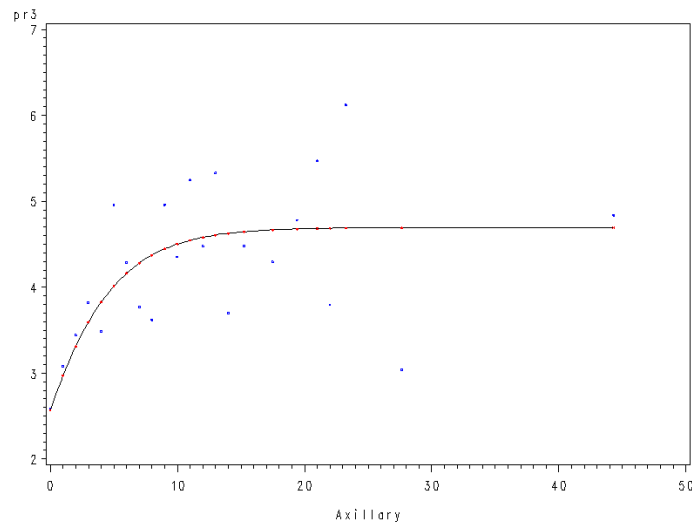


Figure 3.24: Partial residual plot of *Axillary*

3.6 Conclusion

Presently, statistical and prediction packages are providing more nonlinear modelling procedures, such as neural networks (Du Toit, 2006). However, linear and near-linear models, such as GANNs, are in general easier to understand and interpret than nonlinear models and should not be discarded as ineffective and useless when it comes to solving complex multi-dimensional problems. Nonlinear models have their own set of difficulties that may not be straightforward to solve, such as the curse of dimensionality, overparametrization and difficulty to train.

The black box perception of neural networks with respect to interpretation is relieved by the use of the GANN architecture, since graphical methods can be used to interpret the effect of each input variable on the fitted model.

Human judgement is, however, needed to interpret the partial residual plots when GANNs are constructed interactively. This can become a time consuming and daunting task for a large number of variables. Since

human judgement is subjective, the interactive construction algorithm may also result in the creation of suboptimal models.

The difficulties of the interactive construction algorithm is solved with the automated construction algorithm by incorporating a model selection criterion to guide the search for the best GANN model. Consequently, no human interaction is needed during the execution of the algorithm. The data analyst must merely set the parameters of the algorithm before the search for a good GANN model is started and then interpret the results after the search has been completed. The automated construction algorithm is also able to perform in-sample model selection and cross-validation. Given adequate time to evaluate candidate models, the best-first search technique, implemented by the automated construction algorithm, is complete and optimal.

The automated construction algorithm is implemented in the SAS® Macro Language and called *AutoGANN*. The AutoGANN system has a simple, user-friendly and intuitive user interface with default parameter settings that are ready to construct and interpret a relatively good model. The output of the AutoGANN system can also help in guiding the data analyst to gain insight into the models that were developed.

In the next chapter the AutoGANN system is used to search for good GANN models on five different data sets. The models that are found, are compared to MLP models. Good MLP models for the five data sets are found by using a modified version of the N2C2S algorithm. These MLP models are also compared to baseline MLP models that are created by using a brute force method.

“Computer Science is no more about computers than astronomy is about telescopes.”

Professor Edsger Dijkstra

4

Experimental Results

In order to compare multilayer perceptrons (MLPs) and generalized additive neural networks (GANNs), a literature study had to be performed first to understand these two types of neural networks better. This has been accomplished in Chapters 2 and 3. The next step is to perform experiments on different data sets by using MLP and GANN models. These experiments must be well defined and must cover a broad range of tests to provide insight into the manner in which these models compare.

Five different data sets were chosen for this study as a basis for the comparison of the two types of neural networks. These data sets include the Adult (Frank and Asuncion, 2010), Boston Housing (Frank and Asuncion, 2010), Ozone (Breiman and Friedman, 1985), SO_4 (Xiang, 2001) and the Spambase (Frank and Asuncion, 2010) data sets and are all publicly available. The prediction task that was conducted on each of these data sets can be divided into one of two categories:

1. *Classification*: In classification prediction tasks, the target is a set of classes (Berry and Linoff, 1997). Characteristics of these tasks are a well-defined definition of the classes and preclassified examples that make up the training data set. The accuracy of classification tasks in this study is measured by the percentage of instances that are classified correctly.
2. *Regression*: With regression prediction tasks, the target is a continuous value (Berry and Linoff, 1997). The prediction task is to estimate the continuous value as closely as possible. The accuracy of regression tasks in the study is measured in terms of the mean squared error (MSE) value.

In this chapter, a description of the different experiments that were performed, is given in Section 4.1. The Adult data set is considered in Section 4.2 and the results of the experiments that were performed on this data

set are also presented and discussed. In Section 4.3, the Boston Housing data set and its experimental results are considered. The Ozone data set and its experimental results are discussed in Section 4.4 and in Section 4.5 the SO₄ data set and the experiments that were conducted on it are considered. The final data set, Spambase, and experiments that were conducted on it, are discussed in Section 4.6. Finally, a conclusion to this chapter is presented in Section 4.7.

4.1 Experimental design

All the experiments were conducted on a personal computer with an Intel® Core™ 2 Quad processor, operating at 2.66 GHz (per core) with 4 GB of RAM (only 3.5 GB usable with a 32 bit operating system), running Microsoft Windows XP 32 bit. The custom-built MLP construction program was implemented in Base SAS® 9.1 and the AutoGANN system in SAS® Enterprise Miner™ 5.3.

Each data set was used in MLP and GANN experiments. For the GANN experiments, the AutoGANN system, as discussed in Section 3.5.4, was used to search for the best model for the specific data set. The custom-built program for MLP construction, as discussed in Section 2.6.2 (the program code is given in Appendix A), was used to find the best MLP architecture for each data set.

Key features that are considered in order to compare MLPs and GANNs are the following:

- Model complexity: The complexity of a model (degrees of freedom) is measured in terms of the number of parameters.
- Predictive accuracy: For this study, two measurements are used to report the predictive accuracy of a model:

1. Percentage of events that are classified correctly. This measurement is used for classification tasks and is defined as

$$\text{Percentage of events classified correctly} = \frac{X}{N}, \quad (4.1)$$

where X is the number of events that are classified correctly and N is the number of instances. To report the percentage of events that are predicted correctly for K -fold cross-validation, the former is averaged over the K -folds.

2. Mean squared error (MSE). The MSE is utilized to report predictive accuracy for regression tasks, and is defined by Zhang et al. (1998) as

$$\text{MSE} = \frac{\sum_{i=1}^N (y_i - \hat{y}_i)^2}{N}, \quad (4.2)$$

where N is the number of instances, y_i is the target value and \hat{y}_i is the predicted target value. The MSE for K -fold cross-validation is determined by averaging the MSE values over the K -folds.

- Time: The time that is taken to find a good model.

In the next section, the experiments that were performed with GANNs by using the AutoGANN system are discussed.

4.1.1 GANN experiments

Two GANN experiments were conducted on each data set by using the AutoGANN system. For the first experiment, the AutoGANN system was set to run for 12 hours, performing in-sample model selection (the whole data set was used for training and validation). Since no guideline was provided by Du Toit (2006) on how long to search for a good model, a relatively prolonged time was chosen. With this experiment, the AutoGANN system used the SBC model selection criterion to search for a good GANN model. For the second experiment, the AutoGANN system was also set to run for 12 hours, but out-of-sample model selection was performed (70% of the data set was used for training and 30% for validation). The average validation error (VAVERR) was utilized as model selection criterion for this experiment to search for a good GANN model. Both of these experiments used the intelligent start method to determine the starting architecture. The search space for all the GANN experiments was set to $\{0,1,2,3,4,5\}$. To reduce the size of the search space, Du Toit (2006) suggested that a subset of the available sub-architectures should be employed.

In the next section, the MLP experiments are described.

4.1.2 MLP experiments

The MLP construction program that had been developed for this study (Appendix A) was used to conduct the MLP experiments. Six experiments that involved MLPs were conducted on each data set. Three experiments used the modified N2C2S algorithm. The first performed in-sample model selection (the whole data set was used for training and validation), the second performed out-of-sample model selection with hold-out cross-validation (70% of the data set was used for training and 30% of the data set for validation) and the third also performed out-of-sample model selection, but with K -fold cross-validation ($K = 10$). The experiment that used in-sample model selection used the SBC value as model selection criterion and the other two experiments used the average validation error (VAVERR) value as model selection criterion for out-of-sample model selection.

The next three MLP experiments were then performed by using the brute force method of the custom-built MLP construction program for iterating through a number of MLP architectures. The latter technique evaluates a range of MLP architectures to gain insight into the number of hidden neurons that were chosen by the modified N2C2S algorithm. In other words, the brute force method provides a baseline for the modified N2C2S algorithm. For these experiments, a minimum number of 1 hidden node was chosen, along with a maximum number of hidden nodes. All the MLP architectures, ranging from 1 to the maximum number of hidden nodes, were then evaluated. The first brute force method experiment used the whole data set for training and validation, the second brute force method experiment used 70% of the data set for training and 30% of the data set for validation, and the third brute force method experiment performed K -fold cross-validation with $K = 10$. For the Adult and Spambase data sets, the maximum number of hidden neurons were set to 30 and for the remaining three data sets they were set to 15. This was done since the Adult and Spambase data sets are the largest data sets (in terms of dimensions) that were used in this study and may require a more complex MLP than the smaller data sets. Since these three experiments used a brute force approach, no model selection criterion was necessary.

In the next section, the naming of the experiments is discussed.

4.1.3 Experiment identification

In total, there were eight experiments conducted on each data set. To identify these experiments, a unique identification string (ID) is given to each experiment, as shown in Table 4.1. Note that K -fold cross-validation is not implemented by the AutoGANN system and was therefore omitted by experiments conducted with the system. This model selection technique was performed by the MLP construction program to gain more insight into the results that were obtained by the system. Since the hold-out (cross-validation) model selection technique is based on a single sample from the data, more stable results can be obtained by performing K -fold cross-validation.

Experiment description	Model selection	ID
AutoGANN system, using 100% of the data set for training and validation	In-sample	AG100
AutoGANN system, using 70% of the data set for training and 30% of the data set for validation	Out-of-sample	AG70
Modified N2C2S algorithm, using 100% of the data set for training and validation	In-sample	NCS100
Modified N2C2S algorithm, using 70% of the data set for training and 30% of the data set for validation	Out-of-sample	NCS70
Modified N2C2S algorithm, performing 10-fold cross-validation	Out-of-sample	NCS10
Brute force approach, using 100% of the data set for training and validation	n/a	BRUTE100
Brute force approach, using 70% of the data set for training and 30% of the data set for validation	n/a	BRUTE70
Brute force approach, performing 10-fold cross-validation	n/a	BRUTE10

Table 4.1: Experiment identification strings

Next, the Adult data set and the experiments that were conducted on it will be considered.

4.2 The Adult data set

The Adult data set (Frank and Asuncion, 2010) contains data that was extracted from the 1994 U.S. census database. The prediction task for this data set is to determine whether a person makes more than fifty thousand U.S. dollars a year. The data set has 48 842 instances and 16 attributes, and contains instances with missing values. These instances were removed, which left the data set with 45 222 instances. The attributes are information about the persons that are represented by the instances. Information about the attributes is given in Table 4.2.

Attribute	Description	Attribute scale
<i>Age</i>	The age of the person in years	Interval
<i>Workclass</i>	The sector in which the person works	Nominal
<i>Fnlwgt</i>	Final weight	Interval
<i>Education</i>	The education level of the person	Nominal
<i>Education-num</i>	The education level of the person	Interval
<i>Marital-status</i>	The marital status of the person	Nominal
<i>Occupation</i>	The occupation of the person	Nominal
<i>Relationship</i>	The relationship status of the person	Nominal
<i>Race</i>	The race of the person	Nominal
<i>Sex</i>	The sex of the person	Binary
<i>Capital-gain</i>	The amount of capital the person has gained	Interval
<i>Capital-loss</i>	The amount of capital the person has lost	Interval
<i>Hours-per-week</i>	How many hours the person works per week	Interval
<i>Native-country</i>	The country where the person originally comes from	Nominal
<i>Income</i>	Indicates whether the person has an income less or equal to \$50 000, or more than \$50 000	Binary
<i>Status</i>	Indicates the income status, where 0 is for less or equal to \$50 000 and 1 for more than \$50 000	Binary

Table 4.2: Adult data set attributes

The *Education-num* attribute is an interval attribute that is derived from the nominal *Education* attribute and as a result, the *Education* attribute is omitted from the experiments. The *Income* attribute is also omitted, since the *Status* attribute is a binary attribute that is derived from *Income*. The AutoGANN system is designed to handle only numerical binary target attributes with values of 0 and 1. This necessitated the replacement of the class target attribute *Income* with the numerical binary target *Status*. There are thus 14 attributes used in the experiments, 13 as inputs and 1 as the target.

The results from the experiments, using GANNs, are considered next.

4.2.1 GANN results

AG100 and AG70 experiments

In the AG100 and AG70 experiments, the AutoGANN system evaluated 554 and 815 models respectively. The number of models that were generated in the AG70 experiment are greater than the number of models that were created in the AG100 experiment. It takes less time to train the individual models on 70% of the data set and to evaluate the model on the remaining 30% of the data set. As a result, more models can be generated in the allowed time in the AG70 experiment. The best models that were found in the AG100 and AG70 experiments,

according to the SBC and VAVERR values respectively, are shown in Table 4.3. The accuracies of these models, in terms of the percentage of events that were correctly classified, are shown in Table 4.4.

Experiment	Model selection criterion	Model selection criterion value	Parameters	Time
AG100	SBC	-104 254.49	36	12h
AG70	VAVERR	0.307387	45	12h

Table 4.3: Best AG100 and AG70 GANN models on the Adult data set

Experiment	Data role	Data set size	False positive	False negative	True positive	True negative	Accuracy (%)
AG100	Training and validation	45 222	2 209	4 219	6 989	31 805	85.79
AG70	Validation	13 568	1 255	677	2 108	9 528	85.76

Table 4.4: Best AG100 and AG70 GANN models' accuracies on the Adult data set

Table 4.5 shows the GANN sub-architecture of each input of the best GANN model that was found in the AG100 experiment. From this table, it can be seen that 11 out of the 13 input attributes were used in the best model. The *Race* and *Native-country* attributes were removed from the model. The *Workclass*, *Education-num*, *Marital-status*, *Occupation*, *Relationship* and *Sex* attributes all had a linear relationship with the target. The *Age*, *Fnlwgt*, *Capital-gain*, *Capital-loss* and *Hours-per-week* attributes all had a nonlinear relationship with the target.

The GANN sub-architectures of each input of the best model that were found in the AG70 experiment are shown in Table 4.6. It can be seen from this table that 12 out of the 13 input attributes were used in this model. Only the *Native-country* attribute was removed from the model. The *Workclass*, *Education-num*, *Marital-status*, *Occupation*, *Relationship*, *Race* and *Sex* attributes all had a linear relationship with the target. The *Age*, *Fnlwgt*, *Capital-gain*, *Capital-loss* and *Hours-per-week* attributes all had a nonlinear relationship with the target.

The root nodes of the search trees for the AG100 and AG70 experiments were created by using the intelligent start method. The root node of the AG100 experiment was ranked 384th out of the 554 models and had an SBC value of -103 440.65. The GANN architecture of this model was [2,1,0,1,0,1,1,0,0,5,1,2,0]. The root node of the AG70 experiment was ranked 590th out of the 815 models and had a VAVERR value of 0.313473. The GANN architecture of this model was [2,1,0,1,0,1,1,0,0,5,1,2,0]¹.

¹Note that although the training data sets for the AG100 and AG70 experiments were different, the intelligent start algorithm created the same GANN architecture for the root nodes of the two search trees.

Input	GANN sub-architecture
<i>Age</i>	4
<i>Workclass</i>	1
<i>Fnlwgt</i>	2
<i>Education-num</i>	1
<i>Marital-status</i>	1
<i>Occupation</i>	1
<i>Relationship</i>	1
<i>Race</i>	0
<i>Sex</i>	1
<i>Capital-gain</i>	2
<i>Capital-loss</i>	3
<i>Hours-per-week</i>	2
<i>Native-country</i>	0

Table 4.5: Best AG100 GANN model on the Adult data set

Input	GANN sub-architecture
<i>Age</i>	5
<i>Workclass</i>	1
<i>Fnlwgt</i>	2
<i>Education-num</i>	1
<i>Marital-status</i>	1
<i>Occupation</i>	1
<i>Relationship</i>	1
<i>Race</i>	1
<i>Sex</i>	1
<i>Capital-gain</i>	4
<i>Capital-loss</i>	2
<i>Hours-per-week</i>	5
<i>Native-country</i>	0

Table 4.6: Best AG70 GANN model on the Adult data set

In the next section, the results from the GANN experiment that was conducted on the Adult data set are discussed.

Discussion of GANN results

The accuracies of the two models (the best model from the AG100 experiment and the best model from the AG70 experiment), in terms of events that were correctly predicted, are very similar. The model from the AG100 experiment is, however, less complex than that of the AG70 experiment.

In the next section, the experimental results from the MLP experiments on the Adult data set are considered.

4.2.2 MLP results

NCS100, NCS70 and NCS10 experiments

The best MLP models that were found with the modified N2C2S algorithm by using model selection criteria in the NCS100, NCS70 and NCS10 experiments are shown in Table 4.7. The accuracies (percentage events that were correctly classified) of these models are shown in Table 4.8. As seen from this table, the accuracies only differs with less than one percent in these three experiments, but the models from the NCS70 and NCS10 experiments are much more complex, as seen in Table 4.7.

Experiment	Model selection criterion	Model selection criterion value	Hidden neurons	Parameters	Time
NCS100	SBC	-586 444.08	2	31	11m 6s 97ms
NCS70	VAVERR	0.309054	5	76	15m 4s 42ms
NCS10	VAVERR	0.309417	6	91	3h 47m 59s

Table 4.7: Best NCS100, NCS70 and NCS10 MLP models on the Adult data set

Experiment	Data role	Data set size	False positive	False negative	True positive	True negative	Accuracy (%)
NCS100	Training and validation	45 222	2 505	4 172	7 036	31 509	85.24
NCS70	Validation	13 566	714	1 231	2 152	9 469	85.66
NCS10	Validation	4 522	238	415	706	3 163	85.56

Table 4.8: Best NCS100, NCS70 and NCS10 MLP models' accuracies on the Adult data set

In the next section, the BRUTE100, BRUTE70 and BRUTE10 experimental results are considered.

BRUTE100, BRUTE70 and BRUTE10 experiments

For these experiments, the brute force method was performed from an MLP with 1 hidden neuron through to an MLP with 30 hidden neurons. The time that was taken to complete the brute force experiments is shown in Table 4.9. The best MLP models that were found by manually inspecting the results, according to the SBC (BRUTE100) and VAVERR (BRUTE70 and BRUTE10) values, are shown in Table 4.10.

Experiment	Time
BRUTE100	4h 10m 56s 45ms
BRUTE70	3h 0m 58s 81ms
BRUTE10	1d 12h 50m 18s 10ms

Table 4.9: BRUTE100, BRUTE70 and BRUTE10 completion time on the Adult data set

Experiment	Model selection criterion	Model selection criterion value	Hidden neurons	Parameters
BRUTE100	SBC	-586 516.43	2	31
BRUTE70	VAVERR	0.304095	26	391
BRUTE10	VAVERR	0.308925	22	331

Table 4.10: Best BRUTE100, BRUTE70 and BRUTE10 MLP models on the Adult data set

The accuracies of these MLP models are shown in Table 4.11. As seen in this table, the accuracies only differ less than one percent in the three experiments.

Experiment	Data role	Data set size	False positive	False negative	True positive	True negative	Accuracy (%)
BRUTE100	Training and validation	45 222	2 364	4 090	7 118	31 650	85.73
BRUTE70	Validation	13 566	687	1 265	2 101	9 513	85.61
BRUTE10	Validation	4 522	237	409	711	3 165	85.70

Table 4.11: Best BRUTE100, BRUTE70 and BRUTE10 MLP models' accuracies on the Adult data set

The full results of the BRUTE100, BRUTE70 and BRUTE10 experiments are shown in Appendix B. In the next section, the results that were obtained from the performed experiments on the Adult data set, using MLPs, are discussed.

Discussion of MLP results

The modified N2C2S algorithm performed well on this data set and produced MLPs with good accuracies. When the best models that were found in the NCS100, NCS70 and NCS10 experiments are compared to the models from the brute force search method, it can be seen that the modified N2C2S algorithm successfully found a good model without overfitting the data.

In the next section, the results from the MLP and GANN experiments that were conducted on the Adult data set are compared.

4.2.3 Comparison of MLP and GANN results

A comparison between the AG100 and the NCS100 experiments is shown in Table 4.12. The table shows that the accuracies of these two models are very similar. The MLP model is, however, less complex than the GANN model and the time that was taken to finish the search for a good MLP model is far less than the time that was taken to search for a good GANN model.

	AG100	NCS100
Parameters	36	31
Accuracy (%)	85.79	85.24
Time	12h	11m 6s 97ms

Table 4.12: AG100 and NCS100 experimental results comparison on the Adult data set

A comparison between the AG70 and the NCS70 experiments is shown in Table 4.13. The table shows that the accuracies of these two models are, again, very similar, but the MLP model is more complex than the GANN model. The time that was taken to finish the search for a good MLP model is, again, far less than the time that was taken to search for a good GANN model. Since the AutoGANN system does not support K -fold cross-validation, the NCS10 experiment cannot be compared directly to a GANN experiment. The NCS10 experiment did, however, produce an MLP model with an accuracy which is very similar to that of the NCS70 experiment, but the time that was taken to find this model, as well as the model complexity, increased.

	AG70	NCS70
Parameters	45	76
Accuracy (%)	85.79	85.66
Time	12h	15m 4s 42ms

Table 4.13: AG70 and NCS70 experimental results comparison on the Adult data set

In the next section, the Boston Housing data set as well as the results that were obtained from the experiments that had been conducted on it, are considered.

4.3 The Boston Housing data set

The Boston Housing data set (Frank and Asuncion, 2010) is used to predict average house prices in the suburbs of Boston. This data set was originally utilized by Harrison and Rubinfeld (1978) to generate quantitative estimates of the willingness of people to pay for better air quality. The data set consists of 506 instances and has 14 attributes. Each instance represent a certain suburb area of Boston. The attributes represent information about the suburb. The goal is to predict the median value (*Medv*) of owner-occupied houses in these suburbs in U.S.\$1000s. Since *Medv* is an interval attribute, this can be classified as a regression problem. Information about the attributes is presented in Table 4.14.

Attribute	Description	Attribute scale
<i>Crim</i>	Crime rate per capita	Interval
<i>Zn</i>	The proportion of land that is zoned for residential lots which are larger than 25 000 square feet	Interval
<i>Indus</i>	The ratio of acres that consists of non-retail businesses	Interval
<i>Chas</i>	The Charles River dummy variable. If the piece of land borders on the river, then this value is 1, otherwise it is 0	Binary
<i>Nox</i>	The nitric oxides concentration that is measured in parts per 10 million	Interval
<i>Rm</i>	The average number of rooms per home	Interval
<i>Age</i>	The ratio of homes that were built prior to 1940 and are occupied by the owner	Interval
<i>Dis</i>	The weighted distances from five employment centres in Boston	Interval
<i>Rad</i>	An index of accessibility to highways	Interval
<i>Tax</i>	The full value property tax rate per U.S.\$10 000	Interval
<i>Ptatio</i>	The ratio between pupils and teachers	Interval
<i>B</i>	A value that was computed with the following formula: $1000(B_k - 0.63)^2$, where B_k represents the ratio of blacks in the town	Interval
<i>Lstat</i>	The percentage lower status of the population	Interval
<i>Medv</i>	The median value of homes that are occupied by their owners in U.S.\$1000s	Interval

Table 4.14: Boston Housing data set attributes

In the next section, the experiments, using GANNs on this data set, are considered.

4.3.1 GANN results

AG100 and AG70 experiments

The AutoGANN system was set to run for 12 hours in both the AG100 and the AG70 experiments. In this allowed time, the AutoGANN system created 8 602 different GANN models for the AG100 experiment and 8 878 GANN models for the AG70 experiment². As in the GANN experiments with the Adult data set, more models were evaluated in the AG70 experiment than in the AG100 experiment. The best models that were found according to the model selection criteria in the AG100 and AG70 experiments are shown in Table 4.15. The accuracies of these models, in terms of the MSE values, are shown in Table 4.16.

²For an unknown reason, the AutoGANN system created less models in the AG70 experiment than would be expected when compared to the number of models that were created in the AG100 experiment.

Experiment	Model selection criterion	Model selection criterion value	Parameters	Time
AG100	SBC	1 260.11	38	12h
AG70	VAVERR	0.434168	53	12h

Table 4.15: Best AG100 and AG70 GANN models on the Boston Housing data set

Experiment	Data role	Data set size	Accuracy (MSE)
AG100	Training and validation	506	9.659861
AG70	Validation	354	11.423594

Table 4.16: Best AG100 and AG70 GANN models' accuracies on the Boston Housing data set

The GANN architecture of the best model that was found in the AG100 experiment is shown in Table 4.17. From this table, it is clear that 10 out of the 13 input attributes were used in the best model. The *Zn*, *Age* and *Rad* attributes were removed from the model. The *Indus*, *Chas*, *Pt ratio*, *B* and *Lstat* attributes all had a linear relationship with the target. The *Crim*, *Nox*, *Rm*, *Dis* and *Tax* attributes all had a nonlinear relationship with the target.

Input	GANN sub-architecture
<i>Crim</i>	5
<i>Zn</i>	0
<i>Indus</i>	1
<i>Chas</i>	1
<i>Nox</i>	2
<i>Rm</i>	2
<i>Age</i>	0
<i>Dis</i>	4
<i>Rad</i>	0
<i>Tax</i>	2
<i>Pt ratio</i>	1
<i>B</i>	1
<i>Lstat</i>	1

Table 4.17: Best AG100 GANN model on the Boston Housing data set

The best model that was found with the AG70 experiment only used 9 out of the 13 input attributes (Table 4.18). The *Zn*, *Indus*, *Chas* and *Age* attributes were removed from the model. The *Rad* and *Tax* attributes had a linear relationship with the target. The *Crim*, *Nox*, *Rm*, *Dis*, *Ptatio*, *B* and *Lstat* attributes all had a nonlinear relationship with the target.

Input	GANN sub-architecture
<i>Crim</i>	2
<i>Zn</i>	0
<i>Indus</i>	0
<i>Chas</i>	0
<i>Nox</i>	3
<i>Rm</i>	5
<i>Age</i>	0
<i>Dis</i>	5
<i>Rad</i>	1
<i>Tax</i>	1
<i>Ptatio</i>	5
<i>B</i>	5
<i>Lstat</i>	4

Table 4.18: Best AG70 GANN model on the Boston Housing data set

The intelligent start method of the AutoGANN system was utilized to create the root nodes of the search trees for the AG100 and AG70 experiments. The starting GANN model of the AG100 experiment was ranked 6 788th out of the 8 602 models and had a SBC value of 1 371.96. The GANN architecture of this model was [3,0,0,1,1,5,2,5,1,2,1,3,3]. The starting GANN model of the AG70 experiment had an architecture of [1,0,0,1,1,5,2,5,1,1,1,0,1], was ranked 8 102th out of the 8 878 models and had a VAVERR value of 0.621064.

In the next section, the GANN experimental results from the Boston Housing data set are discussed.

Discussion of GANN results

In both of the GANN experiments, the AutoGANN system searched through a large number of GANN models. The number of models that were generated for the Boston Housing data set are far greater than the number of models that were constructed for the Adult data set. This can be attributed to the fact that the Boston Housing data set has much less instances, which causes the training time per model to be reduced. The best GANN model that was found with the AG100 experiment is more accurate in terms of the MSE value than that of the AG70 experiment. This model is also less complex than the one that was found in the AG70 experiment.

The experiments that were conducted by using MLPs are considered next.

4.3.2 MLP results

NCS100, NCS70 and NCS10 experiments

The best MLP models that were found on the Boston Housing data set, using the modified N2C2S algorithm, are shown in Table 4.19. The accuracies of these models are presented in Table 4.20. As seen in this table, the accuracies (MSE) differ substantially in these three experiments.

Experiment	Model selection criterion	Model selection criterion value	Hidden neurons	Parameters	Time
NCS100	SBC	-1 982.71	3	46	6s 42ms
NCS70	VAVERR	0.614418	4	61	8s 89ms
NCS10	VAVERR	0.489649	3	46	55s 61ms

Table 4.19: Best NCS100, NCS70 and NCS10 MLP models on the Boston Housing data set

Experiment	Data role	Data set size	Accuracy (MSE)
NCS100	Training and validation	506	5.709240
NCS70	Validation	354	16.291630
NCS10	Validation	455	11.843650

Table 4.20: Best NCS100, NCS70 and NCS10 MLP models' accuracies on the Boston Housing data set

In the next section, the BRUTE100, BRUTE70 and BRUTE10 experimental results are considered.

BRUTE100, BRUTE70 and BRUTE10 experiments

For these experiments, the brute force method was performed from an MLP with 1 hidden neuron through to an MLP with 15 hidden neurons. The time that was taken to complete these experiments are shown in Table 4.21.

Experiment	Time
BRUTE100	51s 42ms
BRUTE70	36s 20ms
BRUTE10	6m 59s 84ms

Table 4.21: BRUTE100, BRUTE70 and BRUTE10 completion time on the Boston Housing data set

The best MLP models that were found in the BRUTE100, BRUTE70 and BRUTE10 experiments, according to the SBC (BRUTE100) and VAVERR (BRUTE70 and BRUTE10) values, are shown in Table 4.22.

Experiment	Model selection criterion	Model selection criterion value	Hidden neurons	Parameters
BRUTE100	SBC	-1 977.08	3	46
BRUTE70	VAVERR	0.594371	7	106
BRUTE10	VAVERR	0.486868	5	76

Table 4.22: Best BRUTE100, BRUTE70 and BRUTE10 MLP models on the Boston Housing data set

The accuracies (MSE) of these MLP models are shown in Table 4.23. As seen in this table and Table 4.22, the model from the BRUTE70 experiment has a higher complexity and is less accurate than that of the BRUTE10 experiment.

Experiment	Data role	Data set size	Accuracy (MSE)
BRUTE100	Training and validation	506	5.773066
BRUTE70	Validation	354	15.918820
BRUTE10	Validation	455	11.050678

Table 4.23: Best BRUTE100, BRUTE70 and BRUTE10 MLP models' accuracies on the Boston Housing data set

The full results of the BRUTE100, BRUTE70 and BRUTE10 experiments are shown in Appendix B. The MLP experimental results on the Boston Housing data set are discussed in the next section.

Discussion of MLP results

The MSEs of the best MLP models that were found with NCS100, NCS70 and NCS10 experiments differ substantially. The MSE of the NCS70 experiment is about 3 times higher than that of the NCS100 experiment. The modified N2C2S algorithm performed well overall when compared to the baseline BRUTE100, BRUTE70 and BRUTE10 experiments.

In the next section, the results from the GANN and MLP experiments are compared.

4.3.3 Comparison of MLP and GANN results

In Table 4.24, a comparison between the AG100 and the NCS100 experiments is shown. This table shows that the MLP model is more complex, but was found in a much shorter time than the GANN model. The MLP model also performs considerably better than the GANN model in terms of predictive accuracy.

A comparison between the AG70 and the NCS70 experiments is shown in Table 4.25. This table shows that the accuracy of the GANN model is better and that the model is less complex than the best MLP model. However, it is worth noting that the more stable model that was found with the NCS10 experiment is nearly as accurate ($MSE = 11.843650$) as the model that was found in the AG70 experiment and is also less complex (46 parameters).

	AG100	NCS100
Parameters	38	46
Accuracy (MSE)	9.659861	5.709240
Time	12h	6s 42ms

Table 4.24: AG100 and NCS100 experimental results comparison on the Boston Housing data set

	AG70	NCS70
Parameters	58	61
Accuracy (MSE)	11.423594	16.291630
Time	12h	8s 89ms

Table 4.25: AG70 and NCS70 experimental results comparison on the Boston Housing data set

In the next section, the Ozone data set as well as the results that were obtained from the experiments that had been conducted on it, are considered.

4.4 The Ozone data set

The Ozone data set contains meteorological data about the amount of ground level ozone in the Los Angeles metropolis over the course of a year (Breiman and Friedman, 1985). They used this data set to estimate the optimal transformations for multiple regression and correlation. The data set consists of 330 observations and 10 attributes. The attributes represent various information about surface conditions and the objective is to predict the ground level ozone as a pollutant. The target attribute (*Ozone*) is an interval variable and, as a result, the prediction problem is classified as a regression problem. Information about the attributes are presented in Table 4.26.

Attribute	Description	Attribute scale
<i>Vh</i>	The altitude at which the pressure is 500 millibars	Interval
<i>Wind</i>	The wind speed in miles per hour	Interval
<i>Humid</i>	The percentage humidity	Interval
<i>Temp</i>	The temperature in degrees Fahrenheit	Interval
<i>Ibh</i>	The inversion base height in feet	Interval
<i>Dpg</i>	The pressure gradient	Interval
<i>Ibt</i>	The inversion base temperature in degrees Fahrenheit	Interval
<i>Vis</i>	The visibility in miles	Interval
<i>Doy</i>	The day of year	Interval
<i>Ozone</i>	Ground level ozone as a pollutant	Interval

Table 4.26: Ozone data set attributes

In the next section, the experiments, using GANNs on this data set, are considered.

4.4.1 GANN results

AG100 and AG70 experiments

With the AG100 experiment on the Ozone data set, the AutoGANN system evaluated 7 857 different GANN models, but with the AG70 experiment, only 4 636 GANN models were created by the AutoGANN system³. The best models that were found (according to the model selection criteria) in the AG100 and AG70 experiments are shown in Table 4.27 and the accuracies of these models in terms of the MSE value are shown in Table 4.28.

Experiment	Model selection criterion	Model selection criterion value	Parameters	Time
AG100	SBC	898.08	23	12h
AG70	VAVERR	0.810853	52	12h

Table 4.27: Best AG100 and AG70 GANN models on the Ozone data set

³For an unknown reason, the AutoGANN system generated less models in the AG70 experiment than in the AG100 experiment. This is in contrast to the results of the experiments that were conducted on the other data sets and may be as a result of unknown factors that could have influenced the computer's performance at the time of the experiment. The experiment was repeated and the same phenomenon was observed.

Experiment	Data role	Data set size	Accuracy (MSE)
AG100	Training and validation	330	11.276206
AG70	Validation	231	11.381018

Table 4.28: Best AG100 and AG70 GANN models' accuracies on the Ozone data set

In Table 4.29, the GANN sub-architecture of each input of the best GANN model that was found in the AG100 experiment is shown. From this table it can be seen that 6 out of the 9 attributes were used in the best model. The *Wind*, *Temp* and *Ibh* attributes were removed from the model. The *Vh* and *Vis* attributes had a linear relationship with the target. The *Humid*, *Dpg*, *Ibt* and *Doy* attributes all had a nonlinear relationship with the target.

Input	GANN sub-architecture
<i>Vh</i>	1
<i>Wind</i>	0
<i>Humid</i>	2
<i>Temp</i>	0
<i>Ibh</i>	0
<i>Dpg</i>	3
<i>Ibt</i>	2
<i>Vis</i>	1
<i>Doy</i>	3

Table 4.29: Best AG100 GANN model on the Ozone data set

The GANN architecture of the best model that was found in the AG70 experiment is shown in Table 4.30. From this table, it is clear that 8 out of the 9 attributes were used in the best model. Only the *Ibt* attribute was removed from the model and there were no attributes that had a linear relationship with the target. The *Vh*, *Wind*, *Humid*, *Temp*, *Ibh*, *Dpg*, *Vis* and *Doy* attributes all had a nonlinear relationship with the target.

The root nodes of the search trees for both the AG100 and the AG70 experiments were created by using the intelligent start method. The AG100 experiment's starting GANN model was ranked 3 954th out of the 7 857 models and had an SBC value of 940.70. The GANN architecture of this model was [0,0,1,1,0,3,2,1,1]. The root node of the AG70 experiment was ranked 4 627th out of the 4 636 models and had a VAVERR value of 1.644113. The GANN architecture of this model was [0,0,0,1,2,0,0,0,0].

Input	GANN sub-architecture
<i>Vh</i>	2
<i>Wind</i>	3
<i>Humid</i>	5
<i>Temp</i>	4
<i>Ibh</i>	5
<i>Dpg</i>	4
<i>Ibt</i>	0
<i>Vis</i>	3
<i>Doy</i>	4

Table 4.30: Best AG70 GANN model on the Ozone data set

In the next section the results from the GANN experiment conducted on the Ozone data set are discussed.

Discussion of GANN results

The AutoGANN system chose a good starting GANN model with the intelligent start method in the AG100 experiment, as the starting GANN model was ranked better than 3 903 other models. The starting GANN model of the AG70 experiment did, however, not perform as well and only proved to be better than 9 other models. The results show that the best model that was found in the AG70 experiment is, in terms of the MSE value, slightly less accurate than, but more than twice as complex as, the best model that was found in the AG100 experiment.

In the next section, the results from the MLP experiments are considered.

4.4.2 MLP results

NCS100, NCS70 and NCS10 experiments

The best MLP models, according to the model selection criteria, are shown in Table 4.31. These models were found by the modified N2C2S algorithm. The accuracies of these models in terms of the MSE values are shown in Table 4.32. As seen in the table, the accuracies differ substantially in these three experiments.

Experiment	Model selection criterion	Model selection criterion value	Hidden neurons	Parameters	Time
NCS100	SBC	-1 073.19	3	34	3s 17ms
NCS70	VAVERR	1.383407	2	23	1s 69ms
NCS10	VAVERR	1.063660	3	34	27s 89ms

Table 4.31: Best NCS100, NCS70 and NCS10 MLP models on the Ozone data set

Experiment	Data role	Data set size	Accuracy (MSE)
NCS100	Training and validation	330	10.391040
NCS70	Validation	231	17.657960
NCS10	Validation	297	14.357012

Table 4.32: Best NCS100, NCS70 and NCS10 MLP models' accuracies on the Ozone data set

In the next section, the BRUTE100, BRUTE70 and BRUTE10 experimental results are considered.

BRUTE100, BRUTE70 and BRUTE10 experiments

For the Ozone data set, the brute force method was set to reach a maximum of 15 hidden neurons, starting at 1. The time that was taken to complete each of the brute force experiments on this data set is shown in Table 4.33.

Experiment	Time
BRUTE100	30s 94ms
BRUTE70	25s 03ms
BRUTE10	4m 31s 48ms

Table 4.33: BRUTE100, BRUTE70 and BRUTE10 completion time on the Ozone data set

The results were manually analyzed and compared in terms of the SBC (BRUTE100) and VAVERR (BRUTE70 and BRUTE10) values to determine the best model according to the model selection criteria. The best MLP models that were found are shown in Table 4.34 and the accuracies (MSE) of these MLP models are shown in Table 4.35.

The full results of the BRUTE100, BRUTE70 and BRUTE10 experiments are shown in Appendix B. These tables can be used as a baseline for the results of the NCS100, NCS70 and NCS10 experiments respectively.

Experiment	Model selection criterion	Model selection criterion value	Hidden neurons	Parameters
BRUTE100	SBC	-946.11	3	34
BRUTE70	VAVERR	1.082693	2	23
BRUTE10	VAVERR	1.139797	3	34

Table 4.34: Best BRUTE100, BRUTE70 and BRUTE10 MLP models on the Ozone data set

Experiment	Data role	Data set size	Accuracy (MSE)
BRUTE100	Training and validation	330	10.325550
BRUTE70	Validation	231	14.557900
BRUTE10	Validation	297	15.106363

Table 4.35: Best BRUTE100, BRUTE70 and BRUTE10 MLP models' accuracies on the Ozone data set

The MLP experimental results on the Ozone data set are discussed in the next section.

Discussion of MLP results

The MLP models that were found by the modified N2C2S algorithm have the same architectures and thus the same model complexity as the best models that were determined by the brute force method. These models also did not overfit the data, since the number of parameters are not too high. This indicates that the modified N2C2S algorithm performed as expected.

The results from the MLP and GANN experiments are considered in the next section.

4.4.3 Comparison of MLP and GANN results

A comparison between the AG100 and the NCS100 experiments is shown in Table 4.36. This table indicates that the MLP model is more complex, but was found in a much shorter time than the GANN model. The MLP model also performs better than the GANN model in terms of the MSE value.

	AG100	NCS100
Parameters	23	34
Accuracy (MSE)	11.276206	10.391040
Time	12h	3s 17ms

Table 4.36: AG100 and NCS100 experimental results comparison on the Ozone data set

A comparison between the AG70 and the NCS70 experiments is shown in Table 4.37. This table shows that the accuracy of the GANN model is better than that of the MLP model, but that the MLP model was found in far less time and is less complex. The MLP model from the NCS10 experiment is more complex than that of the NCS70 experiment, but performs better.

	AG70	NCS70
Parameters	52	23
Accuracy (MSE)	11.381018	17.657960
Time	12h	1s 69ms

Table 4.37: AG70 and NCS70 experimental results comparison on the Ozone data set

In the next section, the SO_4 data set as well as the results that were obtained from the experiments that were conducted on it, are considered.

4.5 The SO_4 data set

The SO_4 data set (Xiang, 2001) is relatively small, with only 179 instances that contain data of deposits of sulphate (SO_4) over 48 neighbouring states and 179 sites in the US in 1990. The prediction objective is to use the coordinates (latitude and longitude variables) to predict the amount of sulphate at that site. The target variable (SO_4) is an interval variable and this prediction problem can thus be classified as a regression problem. There are 3 attributes as described in Table 4.38.

Attribute	Description	Attribute scale
<i>Latitude</i>	Latitude coordinate of the SO_4 site	Interval
<i>Longitude</i>	Longitude coordinate of the SO_4 site	Interval
<i>SO₄</i>	Amount of sulphate at the site, measured in grams per square meter	Interval

Table 4.38: SO_4 data set attributes

In the next section, the experiments, using GANNs on this data set, are considered.

4.5.1 GANN results

AG100 and AG70 experiments

The AutoGANN system was set to run for 12 hours for the AG100 and AG70 experiments on the SO_4 data set. With only two inputs, the problem has a small search space. As a result, the AutoGANN system generated all possible GANN models (35) in both the AG100 and the AG70 experiments. The best models that were found

Experiment	Model selection criterion	Model selection criterion value	Parameters	Time
AG100	SBC	-259.78	11	31s 73ms
AG70	VAVERR	0.267996	15	32s 50ms

Table 4.39: Best AG100 and AG70 GANN models on the SO₄ data set

in these experiments are shown in Table 4.39 and the accuracies in terms of the MSE value are shown in Table 4.40.

Experiment	Data role	Data set size	Accuracy (MSE)
AG100	Training and validation	179	0.180485
AG70	Validation	125	0.267996

Table 4.40: Best AG100 and AG70 GANN models' accuracies on the SO₄ data set

The GANN sub-architecture for each input of the best model that were found in the AG100 experiment is shown in Table 4.41. From this table, it can be seen that both of the input variables were used in the best model and that both had a nonlinear relationship with the target.

Input	GANN sub-architecture
<i>Latitude</i>	3
<i>Longitude</i>	3

Table 4.41: Best AG100 GANN model on the SO₄ data set

Table 4.42 shows the GANN sub-architecture for each input of the best GANN model that was found in the AG70 experiment. From this table, it can be seen that both of the input variables were used in the best model and that both had a nonlinear relationship with the target.

Input	GANN sub-architecture
<i>Latitude</i>	4
<i>Longitude</i>	4

Table 4.42: Best AG70 GANN model on the SO₄ data set

Both of the root nodes of the search trees for the AG100 and AG70 experiments were created by using the intelligent start method. The root node of the AG100 experiment was ranked 18th out of the 35 models and had an SBC value of -165.18. The GANN architecture of this model was [1,2]. The root node of the AG70 experiment was ranked 11th out of the 35 models and had a VAVERR of 0.241324. The GANN architecture of this model was [1,3].

In the next section, the results from the GANN experiment that was conducted on the SO₄ data set are discussed.

Discussion of GANN results

The results show that the intelligent start method of the AutoGANN system created a good starting GANN model in both of the GANN experiments. The time that was taken to find the best models was also short, since the search space contained only 35 models. The best GANN model that was found in the AG100 experiment is more accurate than the one that was found in the AG70 experiment in terms of the MSE value. This model is also less complex than the best model found in the AG70 experiment.

In the next section, the results from the MLP experiments are considered.

4.5.2 MLP results

NCS100, NCS70 and NCS10 experiments

The best MLP models that were found with the modified N2C2S algorithm are shown in Table 4.43. The accuracies of these models are shown in Table 4.44. As seen in this table, the accuracies differ significantly in these three experiments.

Experiment	Model selection criterion	Model selection criterion value	Hidden neurons	Parameters	Time
NCS100	SBC	-1 284.77	4	17	2s 03ms
NCS70	VAVERR	0.110612	1	5	1s 00ms
NCS10	VAVERR	0.064134	5	21	23s 94ms

Table 4.43: Best NCS100, NCS70 and NCS10 MLP models on the SO₄ data set

In the next section, the BRUTE100, BRUTE70 and BRUTE10 experimental results are considered.

BRUTE100, BRUTE70 and BRUTE10 experiments

For these experiments, the brute force method was performed from an MLP with 1 hidden neuron to an MLP with 15 hidden neurons. The time that was taken to complete the brute force experiments is shown in Table 4.45. The best MLP models that were found according to the SBC (BRUTE100) and VAVERR (BRUTE70 and BRUTE10) values are shown in Table 4.46 and the accuracies of these MLP models are shown in Table 4.47.

Experiment	Data role	Data set size	Accuracy (MSE)
NCS100	Training and validation	179	0.083514
NCS70	Validation	125	0.129118
NCS10	Validation	161	0.107660

Table 4.44: Best NCS100, NCS70 and NCS10 MLP models' accuracies on the SO₄ data set

Experiment	Time
BRUTE100	8s 69ms
BRUTE70	7s 53ms
BRUTE10	1m 18s 80ms

Table 4.45: BRUTE100, BRUTE70 and BRUTE10 completion time on the SO₄ data set

Experiment	Model selection criterion	Model selection criterion value	Hidden neurons	Parameters
BRUTE100	SBC	-1 287.82	3	13
BRUTE70	VAVERR	0.054677	4	17
BRUTE10	VAVERR	0.062345	5	21

Table 4.46: Best BRUTE100, BRUTE70 and BRUTE10 MLP models on the SO₄ data set

Experiment	Data role	Data set size	Accuracy (MSE)
BRUTE100	Training and validation	179	0.092193
BRUTE70	Validation	125	0.092898
BRUTE10	Validation	161	0.107044

Table 4.47: Best BRUTE100, BRUTE70 and BRUTE10 MLP models' accuracies on the SO₄ data set

The full results of the BRUTE100, BRUTE70 and BRUTE10 experiments are shown in Appendix B. These tables can be used as a baseline for the results of the NCS100, NCS70 and NCS10 experiments respectively. The MLP experimental results on the SO₄ data set are discussed in the next section.

Discussion of MLP results

The MLP models that were created in the NCS100, NCS70 and NCS10 experiments performed well when compared to those created in the BRUTE100, BRUTE70 and BRUTE10 experiments. This indicate that the modified N2C2S algorithm succeeded in creating good MLP models on this data set.

In the next section, the results from the GANN and MLP experiments are compared.

4.5.3 Comparison of MLP and GANN results

A comparison between the AG100 and the NCS100 experiments is shown in Table 4.48. The table shows that the MLP model is more complex, but was found in a far shorter time than the GANN model. The MLP model also performed better than the GANN model.

	AG100	NCS100
Parameters	11	17
Accuracy (MSE)	0.180485	0.083514
Time	31s 73ms	2s 03ms

Table 4.48: AG100 and NCS100 experimental results comparison on the SO₄ data set

A comparison between the AG70 and the NCS70 experiments is shown in Table 4.49. The table shows that the accuracy of the MLP model is better, the model is less complex and it was found in less time than the GANN model.

	AG70	NCS70
Parameters	15	5
Accuracy (MSE)	0.267996	0.129118
Time	32s50ms	1s

Table 4.49: AG70 and NCS70 experimental results comparison on the SO₄ data set

In the next section, the Spambase data set as well as the results that were obtained from the experiments that had been conducted on it, are considered.

4.6 The Spambase data set

The Spambase data set (Frank and Asuncion, 2010) contains words and characters that were extracted from either e-mails that were known to be spam or normal e-mails. Each instance in the Spambase data set represents an e-mail and each instance is classified as spam or normal by the *Spam* attribute. Since the target (*Spam*) is a binary variable, this can be regarded as a classification problem. The data set consists of 4601 instances and have 58 attributes. The attributes are presented in Table 4.50.

Attribute	Description	Attribute scale
<i>Word_freq_WORD</i>	48 attributes that represent the frequency (percentage) with which a specific <i>WORD</i> occurs in the e-mail	Interval
<i>Char_freq_CHAR</i>	6 attributes that represent the frequency (percentage) with which a specific <i>CHAR</i> occurs in the e-mail	Interval
<i>Capital_run_length_average</i>	The average length of an uninterrupted sequence of capital letters	Interval
<i>Capital_run_length_longest</i>	The length of the longest uninterrupted sequence of capital letters	Interval
<i>Capital_run_length_total</i>	The total number of capital letters	Interval
<i>Spam</i>	Classifies the instance as spam (1) or normal e-mail (0)	Binary

Table 4.50: Spambase data set attributes

The results from the GANN experiments on this data set are considered next.

4.6.1 GANN results

AG100 and AG70 experiments

The AutoGANN system evaluated 1 203 and 1 728 different GANN architectures in the AG100 and AG70 experiments respectively. The results of the best models that were found in the AG100 and AG70 experiments according to the model selection criteria are shown in Table 4.51 and the accuracies of these models in terms of events that were correctly classified are shown in Table 4.52.

Experiment	Model selection criterion	Model selection criterion value	Parameters	Time
AG100	SBC	-14 500.12	142	12h
AG70	VAVERR	0.166957	137	12h

Table 4.51: Best AG100 and AG70 GANN models on the Spambase data set

Experiment	Data role	Data set size	False positive	False negative	True positive	True negative	Accuracy (%)
AG100	Training and validation	4 601	101	109	1 704	2 687	95.44
AG70	Validation	1 382	46	36	509	791	94.07

Table 4.52: Best AG100 and AG70 GANN models' accuracies on the Spambase data set

The GANN architecture of the best model that was created in the AG100 experiment is shown in Table 4.53. From this table, it can be seen that 47 out of the 57 attributes were used in this model. From these, 10 input attributes were removed from the model, 30 input attributes had a linear relationship with the target and 17 input attributes had a nonlinear relationship with the target.

Input	GANN sub-architecture	Input	GANN sub-architecture	Input	GANN sub-architecture
<i>Make</i>	1	<i>Credit</i>	1	<i>Pm</i>	1
<i>Address</i>	0	<i>Your</i>	3	<i>Direct</i>	1
<i>All</i>	1	<i>Font</i>	1	<i>Cs</i>	1
<i>Word_3d</i>	1	<i>Word_000</i>	1	<i>Meeting</i>	3
<i>Our</i>	3	<i>Money</i>	3	<i>Original</i>	0
<i>Over</i>	1	<i>Hp</i>	3	<i>Project</i>	1
<i>Remove</i>	3	<i>Hpl</i>	0	<i>Re</i>	1
<i>Internet</i>	2	<i>George</i>	1	<i>Edu</i>	3
<i>Order</i>	1	<i>Word_650</i>	1	<i>Table</i>	0
<i>Mail</i>	0	<i>Lab</i>	1	<i>Conference</i>	1
<i>Receive</i>	1	<i>Labs</i>	0	<i>Char_1</i>	1
<i>Will</i>	2	<i>Telnet</i>	0	<i>Char_2</i>	1
<i>People</i>	1	<i>Word_857</i>	1	<i>Char_3</i>	0
<i>Report</i>	2	<i>Data</i>	1	<i>Char_4</i>	2
<i>Addresses</i>	1	<i>Word_415</i>	1	<i>Char_5</i>	3
<i>Free</i>	3	<i>Word_85</i>	1	<i>Char_6</i>	1
<i>Business</i>	1	<i>Technology</i>	3	<i>Length_average</i>	3
<i>Email</i>	0	<i>Word_1999</i>	3	<i>Length_longest</i>	0
<i>You</i>	1	<i>Parts</i>	1	<i>Length_total</i>	3

Table 4.53: Best AG100 GANN model on the Spambase data set

The architecture of the best GANN model from the AG70 experiment is shown in Table 4.54. From this table, it can be seen that 40 out of the 57 attributes were used in this model. From these, 17 input attributes were removed from the model, 18 input attributes had a linear relationship with the target and 22 input attributes had a nonlinear relationship with the target.

The root nodes of the search trees for the AG100 and the AG70 experiments were created by using the intelligent start method. The root node of the AG100 experiment was ranked 1 184th out of the 1 203 models and had an SBC value of -13 480.67. The GANN architecture of this model was

[0,0,0,0,2,0,1,1,0,0,0,1,0,0,0,2,0,0,0,0,2,0,0,2,2,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,2,2,0,0,0,0,0,3,2,0,2,0,2].

Input	GANN sub-architecture	Input	GANN sub-architecture	Input	GANN sub-architecture
<i>Make</i>	3	<i>Credit</i>	1	<i>Pm</i>	1
<i>Address</i>	1	<i>Your</i>	1	<i>Direct</i>	0
<i>All</i>	1	<i>Font</i>	2	<i>Cs</i>	0
<i>Word_3d</i>	1	<i>Word_000</i>	3	<i>Meeting</i>	2
<i>Our</i>	2	<i>Money</i>	2	<i>Original</i>	1
<i>Over</i>	3	<i>Hp</i>	2	<i>Project</i>	0
<i>Remove</i>	1	<i>Hpl</i>	1	<i>Re</i>	3
<i>Internet</i>	1	<i>George</i>	1	<i>Edu</i>	5
<i>Order</i>	0	<i>Word_650</i>	0	<i>Table</i>	0
<i>Mail</i>	1	<i>Lab</i>	1	<i>Conference</i>	2
<i>Receive</i>	0	<i>Labs</i>	0	<i>Char_1</i>	1
<i>Will</i>	3	<i>Telnet</i>	1	<i>Char_2</i>	2
<i>People</i>	3	<i>Word_857</i>	0	<i>Char_3</i>	0
<i>Report</i>	1	<i>Data</i>	2	<i>Char_4</i>	2
<i>Addresses</i>	0	<i>Word_415</i>	0	<i>Char_5</i>	2
<i>Free</i>	3	<i>Word_85</i>	0	<i>Char_6</i>	1
<i>Business</i>	0	<i>Technology</i>	2	<i>Length_average</i>	0
<i>Email</i>	0	<i>Word_1999</i>	3	<i>Length_longest</i>	2
<i>You</i>	1	<i>Parts</i>	0	<i>Length_total</i>	2

Table 4.54: Best AG70 GANN model on the Spambase data set

The root node of the AG70 experiment was ranked 1 694th out of the 1 728 models and had a VAVERR value of 0.263859. The GANN architecture of this model was

[2,0,0,0,2,2,1,1,0,0,0,2,2,0,0,3,0,0,0,0,2,2,2,2,2,0,0,0,0,0,0,2,0,0,2,2,0,0,0,0,2,0,0,3,5,0,2,1,2,0,3,2,0,0,2,2].

In the next section, the results from the GANN experiment that were conducted on the Spambase data set are discussed.

Discussion of GANN results

The AG70 experiment's best model, with 137 parameters, is less complex than the AG100 experiment's model with 142 parameters and the accuracies of these two models differ with less than two percent, with 95.44% for the AG100 experiment's model and 94.07% for the AG70 experiment's model. However, it can be assumed that it is more important not to classify a normal e-mail as spam and, as a result, the classification accuracy of non-events (normal e-mails) must be taken into account. The predictive accuracy of non-events is shown in

(4.3) and (4.4).

$$\text{Predictive accuracy of non-events (AG100)} = \frac{2\ 687}{2\ 687 + 109} = 96.10\%. \quad (4.3)$$

$$\text{Predictive accuracy of non-events (AG70)} = \frac{791}{791 + 36} = 95.64\%. \quad (4.4)$$

In the next section, the MLP experimental results are considered.

4.6.2 MLP results

NCS100, NCS70 and NCS10 experiments

The best MLP models that were found with the modified N2C2S algorithm in the NCS100, NCS70 and NCS10 experiments are shown in Table 4.55.

Experiment	Model selection criterion	Model selection criterion value	Hidden neurons	Parameters	Time
NCS100	SBC	-59 708.79	8	473	5m 53s 61ms
NCS70	VAVERR	0.202062	2	119	49s 70ms
NCS10	VAVERR	0.197001	2	119	9m 34s 94ms

Table 4.55: Best NCS100, NCS70 and NCS10 MLP models on the Spambase data set

The accuracies of these models are shown in Table 4.56. As seen in this table, the accuracy of the best model that was found in the NCS100 experiment is very high, which may be the result of overfitting.

Experiment	Data role	Data set size	False positive	False negative	True positive	True negative	Accuracy (%)
NCS100	Training and validation	4 601	11	16	1 797	2 777	99.41
NCS70	Validation	1 380	34	60	467	819	93.19
NCS10	Validation	460	13	17	164	266	93.49

Table 4.56: Best NCS100, NCS70 and NCS10 MLP models' accuracies on the Spambase data set

In the next section, the BRUTE100, BRUTE70 and BRUTE10 experimental results are considered.

BRUTE100, BRUTE70 and BRUTE10 experiments

For these experiments, the brute force method was performed from an MLP with 1 hidden neuron to an MLP with 30 hidden neurons. The time that was taken to complete the brute force experiments is shown in Table 4.57.

Experiment	Time
BRUTE100	2h 0m 8s 41ms
BRUTE70	1h 16m 13s 59ms
BRUTE10	21h 13m 51s 75ms

Table 4.57: BRUTE100, BRUTE70 and BRUTE10 completion time on the Spambase data set

The best MLP models that was found according to the SBC (BRUTE100) and VAVERR (BRUTE70 and BRUTE10) values are shown in Table 4.58. The accuracies of these MLP models are shown in Table 4.59.

Experiment	Model selection criterion	Model selection criterion values	Hidden neurons	Parameters
BRUTE100	SBC	-59 347.61	8	273
BRUTE70	VAVERR	0.190246	2	119
BRUTE10	VAVERR	0.198754	1	60

Table 4.58: Best BRUTE100, BRUTE70 and BRUTE10 MLP models on the Spambase data set

Experiment	Data role	Data set size	False positive	False negative	True positive	True negative	Accuracy (%)
BRUTE100	Training and validation	4 601	10	16	1 797	2 778	99.43
BRUTE70	Validation	1 380	24	63	524	769	93.70
BRUTE10	Validation	460	16	14	168	262	93.44

Table 4.59: Best BRUTE100, BRUTE70 and BRUTE10 MLP models accuracies on the Spambase data set

The full results of the BRUTE100, BRUTE70 and BRUTE10 experiments are shown in Appendix B. In the next section, the MLP experimental results are discussed.

Discussion of MLP results

When the best models that were found in the NCS100, NCS70 and NCS10 experiments are compared to the base line brute force method experiments, it can be seen that the modified N2C2S algorithm performed well by finding good MLP models. The high accuracy and high complexity of the model that was found in the NCS100 experiment may, however, indicate that this model has overfitted.

The accuracy of predicting a normal e-mail correctly is more important, as was discussed earlier. This normal e-mail classification accuracy for experiments NCS100, NCS70 and NCS10 are shown in (4.5), (4.6) and (4.7) respectively.

$$\text{Predictive accuracy of non-events (NCS100)} = \frac{2\,777}{2\,777 + 16} = 99.44\%. \quad (4.5)$$

$$\text{Predictive accuracy of non-events (NCS70)} = \frac{819}{819 + 60} = 93.17\%. \quad (4.6)$$

$$\text{Predictive accuracy of non-events (NCS10)} = \frac{266}{266 + 17} = 93.99\%. \quad (4.7)$$

In the next section, the results that were obtained from the MLP and GANN experiments that had been conducted on the Spambase data set are considered.

4.6.3 Comparison of MLP and GANN results

The comparison between the AG100 and the NCS100 experiments is shown in Table 4.60. The table shows that the MLP model is much more complex, but was found in a much shorter time than the GANN model. The MLP model also performs much better than the GANN model.

	AG100	NCS100
Parameters	142	473
Accuracy (%)	95.44	99.41
Time	12h	5m 53s 61ms

Table 4.60: AG100 and NCS100 experimental results comparison on the Spambase data set

The comparison between the AG70 and the NCS70 experiments is shown in Table 4.61. The table shows that the accuracy of the GANN model is better than that of the MLP model, but is more complex and took much longer to find. The accuracy of predicting normal e-mail correctly is shown in Table 4.62. As can be seen in this table, the NCS100 experiment's model almost has a perfect score, but this, as mentioned earlier, may be a result of overfitting. The AG70 experiment's model performed better than the best models of the NCS70 and NCS10 experiments.

	AG70	NCS70
Parameters	137	119
Accuracy (%)	94.07	93.19
Time	12h	49s 70ms

Table 4.61: AG70 and NCS70 experimental results comparison on the Spambase data set

Experiment	Accuracy (%)
AG100	96.10
NCS100	99.44
AG70	95.64
NCS70	93.17
NCS10	93.99

Table 4.62: Normal e-mail prediction accuracies comparison on the Spambase data set

In the next section, conclusions to this chapter is presented.

4.7 Conclusion

In this chapter, eight different experiments were defined and named. All eight experiments were conducted on each of the five data sets (Adult data set, Boston Housing data set, Ozone data set, SO_4 data set and Spambase data set). From the eight experiments, two were experiments that were conducted with GANN models (one experiment used in-sample model selection and the other one utilized out-of-sample model selection) and the remaining six experiments were conducted by using MLP models (one experiment utilized in-sample model selection, two experiments used out-of-sample model selection and the remaining three experiments were conducted by using a brute force method with no model selection technique). The AutoGANN system and the MLP construction program were utilized to construct the GANN and MLP models respectively.

A higher level comparison between MLPs and GANNs with regard to accuracy, complexity, comprehensibility, ease of construction, and utility is given in the next chapter.

“Statistics: The only science that enables different experts using the same figures to draw different conclusions.”

Evan Esar

5

Comparative discussion on MLPs and GANNs

A literature study on multilayer perceptrons (MLPs) and generalized additive neural networks (GANNs) has been conducted in Chapters 2 and 3 respectively and results have been obtained from the experiments that were performed on five publicly available data sets with each of these neural networks, as discussed in Chapter 4. It is now possible to compare MLPs with GANNs in terms of predictive accuracy (Section 5.1), model complexity (Section 5.2), comprehensibility (Section 5.3), ease of construction (Section 5.4), and utility (Section 5.5). A conclusion to this chapter is presented in Section 5.6.

5.1 Predictive accuracy

To compare the predictive accuracies of the MLPs and GANNs, the experiments that were conducted on each data set can be divided into two groups: First, the experiments that were performed where in-sample model selection was utilized (the full data set was used for training and validation) with the AG100 and NCS100 experiments. Second, those experiments that were performed with out-of-sample model selection (hold-out and 10-fold cross-validation) with the AG70, NCS70 and NCS10 experiments. For the experiments that were performed with in-sample model selection, the SBC value was used as the model selection criterion, while the average validation error (VAVERR) value was used as model selection criterion for the experiments that were performed with out-of-sample model selection. To gain further insight into the best model that was selected for the classification problems, the events classification information (percentage correct event prediction) was also considered. The best models that were found for regression tasks are presented in terms of the mean squared error (MSE).

In Table 5.1 the accuracy results that were obtained by the best MLP models that had been constructed, are summarized. The accuracy results that were obtained by the best GANN models that had been constructed, are summarized in Table 5.2. Table 5.3 shows the best models that were built in terms of accuracy when in-sample model selection and out-of-sample model selection were performed on the five data sets. This table is obtained by comparing Tables 5.1 and 5.2.

MLPs	In-sample model selection	Out-of-sample model selection (cross-validation)	Out-of-sample model selection (10-fold cross-validation)
Adult (%)	85.24	85.66	85.56
Boston Housing (MSE)	5.709240	16.291630	11.843650
Ozone (MSE)	10.391040	17.657960	14.357012
SO ₄ (MSE)	0.083514	0.129118	0.107660
Spambase (%)	99.41	93.19	93.49

Table 5.1: Accuracy of the best MLP models that were obtained

GANNs	In-sample model selection	Out-of-sample model selection (cross-validation)	Out-of-sample model selection (10-fold cross-validation)
Adult (%)	85.79	85.76	n/a
Boston Housing (MSE)	9.659861	11.423594	n/a
Ozone (MSE)	11.276206	11.381018	n/a
SO ₄ (MSE)	0.180485	0.267996	n/a
Spambase (%)	95.44	94.07	n/a

Table 5.2: Accuracy of the best GANN models that were obtained

GANNs vs MLPs	In-sample model selection	Out-of-sample model selection (cross-validation)
Adult	GANN	GANN
Boston Housing	MLP	GANN
Ozone	MLP	GANN
SO ₄	MLP	MLP
Spambase	MLP	GANN

Table 5.3: Best models that were obtained in terms of accuracy

For the two classification tasks (Adult data set and Spambase data set), the accuracies are reported in terms of percentage events that were predicted correctly. The accuracies of the three regression tasks (Boston Housing

data set, Ozone data set and SO₄ data set) are presented in terms of the MSE. With in-sample model selection, both types of neural networks performed well on the Adult data set, Ozone data set and the Spambase data set. When in-sample model selection was performed (Table 5.3), the MLPs outperformed the GANNs in terms of predictive accuracy in four of the five data sets (Boston Housing data set, Ozone data set, SO₄ data set and Spambase data set). Note that the accuracy of the MLP model on the Spambase data set that was created by using in-sample model selection is very high (99.41%). This may be as a result of overfitting.

When out-of-sample model selection (hold-out cross-validation) was performed, the GANNs performed better than the MLPs in four of the five data sets (Adult data set, Boston Housing data set, Ozone data set and Spambase data set), as can be seen in Table 5.3. The last column of Table 5.1 shows the results that were obtained by 10-fold cross-validation that was performed on the MLPs. These results are more stable when compared to the hold-out cross-validation method, as the latter is based on a single sample that was taken from the data. Unfortunately, 10-fold cross-validation was not implemented in the AutoGANN system, making a comparison infeasible. Four of the five MLP results (Boston Housing data set, Ozone data set, SO₄ data set and Spambase data set) showed that the models that had been created with in-sample model selection were more accurate than those that had been created by using out-of-sample model selection (hold-out cross-validation and 10-fold cross-validation). The same observation was made with the GANN experiments, where the in-sample model selection models performed better than the out-of-sample model selection models in all five data sets. This may also be as a result of overfitting.

It is clear from Table 5.3 that no single type of neural network always outperforms the other in terms of predictive accuracy. This would suggest that the type of neural network model that is used, is highly dependent on the problem.

In their simplest form, additive models (GANNs) are unable to model interactions (De Waal and Du Toit, 2011). A possible remedy is to add explicit interaction terms or variables to the set of independent variables. These interaction variables are then treated as normal variables and the model is estimated in the usual manner. Also, since the interactions have been made explicit, the contribution of the interaction terms can be analyzed by using partial residual plots. This would give added insight into the model, which is not possible with an MLP model, as the interactions in the MLP are intertwined with the contributions of the inputs. Interactions could have an influence on the accuracy and should be further investigated.

Sometimes, the modified N2C2S algorithm may overfit the data, which suggests that further research into this matter should be conducted. Since the SBC model selection criterion was utilized for in-sample model selection, experiments with other types of criteria should also be performed to determine if this choice of criterion caused the model to overfit.

When choosing between GANN and MLP models, the following guidelines can now be given, following the accuracy results of the models:

- MLP models may perform better than GANN models in terms of accuracy when in-sample model selection with the SBC criterion is used and may thus be suggested for problems where in-sample model selection is used.

- GANN models may perform better than MLP models in terms of accuracy when out-of-sample model selection with the average validation error is used and may thus be suggested for problems where out-of-sample model selection is used.

In the next section, the complexity of these models will be considered.

5.2 Model complexity

Model complexity, as discussed in Section 3.5.2, has a direct affect upon the generalization capability of a model. The former is controlled by both the modified N2C2S algorithm for MLPs and the automated construction algorithm for GANNs by means of an in-sample model selection criterion when in-sample model selection is performed and cross-validation when out-of-sample model selection is done. To determine the appropriate level of complexity, the principle of parsimony is followed by both algorithms.

Model complexity is measured by the number of parameters of the model (degrees of freedom). Tables 5.4 and 5.5 show the number of parameters of the best MLP and GANN models that were constructed respectively. Zhang et al. (1998) describes a rule-of-thumb, stating that at least 10 records are needed to estimate each parameter in a model accurately. For the in-sample model selection experiments (where the full data sets were used for training and validation), this heuristic gives a maximum of 4522 parameters for the Adult problem (the data set has 45 222 records), 51 parameters for the Boston Housing problem (the data set has 506 records), 33 parameters for the Ozone problem (the data set has 330 records), 18 parameters for the SO₄ problem (the data set has 179 records) and 460 parameters for the Spambase problem (the data set has 4 601 records). For the experiments where the hold-out method was used (70% of the data set for training and 30% for validation), the rule-of-thumb gives a maximum of 3 166 parameters for the Adult data set (the training data set has 31 656 records), 35 parameters for the Boston Housing data set (the training data set has 354 records), 23 parameters for the Ozone data set (the training data set has 231 records), 13 parameters for the SO₄ data set (the training data set has 125 records) and 322 parameters for the Spambase problem (the data set has 3 221 records). Finally, for the experiments where 10-fold cross-validation were used (for each fold, 90% of the data set is used for training and 10% for validation), the heuristic gives a maximum of 4 070 parameters for the Adult data set (for each fold the training data set has 40 700 records), 46 parameters for the Boston Housing problem (the data set has 455 records), 30 parameters for the Ozone problem (the data set has 297 records), 16 parameters for the SO₄ problem (the data set has 161 records) and 414 parameters for the Spambase problem (the data set has 4 141 records). In Tables 5.4 and 5.5 all models are within these bounds, except those that are marked with *, which may indicate models that are too complex.

When the total number of parameters of the in-sample model selection experiments are considered (Tables 5.4 and 5.5), it clearly shows that GANNs are less complex with 250 parameters, compared to the 601 parameters of the MLPs. However, the MLP from the NCS100 experiment on the Spambase data set has a very large number of parameters (473). This may indicate overfitting, since as discussed earlier, the accuracy of this model is very high (99.41%). The total number of parameters in the out-of-sample model selection experiments

show that MLPs are slightly less complex than GANNs in these experiments.

MLPs	In-sample model selection	Out-of-sample model selection (cross-validation)	Out-of-sample model selection (10-fold cross-validation)
Adult	31	76	91
Boston Housing	46	61*	46
Ozone	34*	23	34*
SO ₄	17	5	21*
Spambase	473*	119	119
Total	601	284	311

Table 5.4: Number of parameters of best MLP models that were obtained

GANNs	In-sample model selection	Out-of-sample model selection (cross-validation)	Out-of-sample model selection (10-fold cross-validation)
Adult	36	45	n/a
Boston Housing	38	53*	n/a
Ozone	23	52*	n/a
SO ₄	11	15*	n/a
Spambase	142	137	n/a
Total	250	302	n/a

Table 5.5: Number of parameters of best GANN models that were obtained

In general, the GANN models that were selected by the automated construction algorithm may be more parsimonious, as it allows finer control over the number of parameters in the model than is possible with an MLP (De Waal and Du Toit, 2011). With an MLP, all the inputs are connected to all the neurons in the hidden layer. For example, if the given problem has 7 inputs, the hidden layer has 10 neurons and the output layer has 1 neuron, the number of parameters increase or decrease in multiples of 9 when a neuron is added or removed from the hidden layer (7 inputs connected to the neuron, a bias and a connection to the output layer). The best MLP model may be a model with a different number of parameters from those that are described above.

From the discussion on the complexity of the models, the following guidelines can now be given when choosing between GANN and MLP models:

- When in-sample model selection with the SBC criterion is performed, GANN models may tend to be less complex than MLP models.
- When out-of-sample model selection with the average validation error criterion is performed, MLP models may tend to be slightly less complex.

In the next section, the comprehensibility of the constructed models is discussed.

5.3 Comprehensibility

In many real-world problems, the need to interpret the results by understanding the relationships between input attributes and the target is just as important as the predictive accuracy of the model. MLP models are considered to be black boxes, as results that are obtained by these models are difficult to interpret. On the other hand, GANN models were developed, in part, to overcome this problem that MLPs have. Results from GANNs can be interpreted with partial residual plots which show the relationship between inputs and the target. It can thus be suggested that GANN models should be used with problems where the understanding of the relationships between input attributes and the target is important.

In the next section, the ease with which the models can be constructed is considered.

5.4 Ease of construction

Another important feature that must be considered when comparing GANNs and MLPs is the relative ease with which the models are built. In this study, the AutoGANN system was used to construct the GANN models and an implementation of the modified N2C2S algorithm was used to build MLP models. Both programs were implemented in the SAS® Macro Language. Before search commences, only a few parameters must be set. Both the systems then search automatically for the best model, without the need for input from the user while the search is taking place.

Constructing GANNs with the AutoGANN system is much easier, since it has a user-friendly graphical interface. The MLP construction program does not have a graphical user interface and it is required of the user to change the settings directly in the code of the program.

With the AutoGANN system, the time that is allowed to search for a good GANN model must be set beforehand. Unfortunately, there is no guideline on how long it will take the system to find such a good GANN model and as a result, a relatively long time is usually chosen (12 hours in this study). In contrast, as the results that were obtained in Chapter 4 indicate, the time that is taken to search for a good MLP model with the modified N2C2S algorithm is sometimes far less than the time that is needed to find a good GANN model.

In the next section, the usefulness of these two types of neural networks and the programs that are used to create them are discussed.

5.5 Utility

Both of these types of neural networks are applicable to predictive problems and are relatively easy to use. In both the cases of MLPs and GANNs, the problem lies in selecting the best neural network architecture for the problem at hand. The AutoGANN system that was used to search for the best GANN model is very advanced and incorporates many heuristic features that decreases the time it takes to find the best GANN model. The automated construction algorithm, implemented in the AutoGANN system, consists of a complete search strategy. Consequently, if not stopped by a time limit, the automated construction algorithm will search through

all possible models in the search space.

The MLP model selection program that was developed for this study may find good MLP models faster than the AutoGANN system can find good GANN models, as shown in Chapter 4.

Both of these construction programs were developed in the SAS® Macro Language. The AutoGANN system is implemented as a model node in the SAS® Enterprise Miner™ package. Since both the AutoGANN system and the MLP construction program are developed in SAS®, they both require a SAS® software licence. This can be problematic, since SAS® software is relatively expensive.

In the next section, a conclusion to this chapter is presented.

5.6 Conclusion

In this chapter the results that were obtained with MLPs and GANNs on the five chosen data sets (Chapter 4) and the programs that were used to create the MLP and GANN models were compared with regard to predictive accuracy, model complexity, comprehensibility, ease of construction, and utility.

In the next and final chapter, the conclusion of this study is presented.

“Insanity: doing the same thing over and over again and expecting different results.”

Albert Einstein

6

Conclusion

The purpose of this study was to investigate and compare GANN and MLP models as prediction techniques. Since theory provides little guidance on the selection of the appropriate architecture a priori and the architectures of artificial neural networks may differ for each different data set, a search had to be performed to find a good model for a specific data set. As a result, an automated construction algorithm for GANNs, implemented by the AutoGANN system, was used to search for good GANN models while a modified version of the N2C2S algorithm, implemented by a custom-built program, was used to search for good MLP models. These two systems were also investigated and compared.

In this chapter, a summary of findings is presented in Section 6.1. Section 6.2 focuses on a summary of the contributions of this study and suggestions for future work are presented in Section 6.3. The chapter is concluded in Section 6.4.

6.1 Summary of findings

The following discoveries were made in this study:

- Predictive accuracy: When in-sample model selection is performed with the SBC model selection criterion, the MLP models seem to be more accurate than GANN models. When out-of-sample model selection is done with the average validation error model selection criterion, GANN models, on the other hand, seem to be more accurate than MLP models. Overall, no single type of neural network always outperforms the other in terms of predictive accuracy. This may suggest that the type of neural network

model that was used is highly dependent on the problem.

- **Complexity:** Overall, MLP and GANN models seem to be very similar in complexity, with the exception of the in-sample model selection experiment on the Spambase data set. In this experiment, the modified N2C2S algorithm selected a much more complex MLP model. Although very similar in complexity, there were, however, slight differences which may suggest that GANN models are less complex than MLP models when in-sample model selection is used and when out-of-sample model selection is used, MLP models seem to be the lesser complex model.
- **Comprehensibility:** The MLP models are considered to be black boxes in terms of understandability and interpretability, but the GANN models overcome this problem with the use of partial residual plots, which show the relationships between input attributes and the target graphically. As a result, the use of GANN models may be suggested when understanding of the relationships between input attributes and the target is important.
- **Ease of construction:** Both programs that were used to search for a good GANN and MLP model respectively make it easy to construct the model, but the AutoGANN system is far more advanced and offers a graphical user interface, whereas the MLP construction program does not. This makes it easy for the AutoGANN user to change the settings and select the appropriate experiment, whereas the MLP construction program user would have to do this in the code itself.

The AutoGANN system, however, has one difficulty. There is no guideline to help in the selection of the time that is needed to find a good GANN model. The system will thus continue to search for a better GANN model until a specified time has passed or the search space has been exhausted. The modified N2C2S algorithm that was implemented in the MLP construction program, on the other hand, will stop as soon as a good MLP model (according to the model selection criterion) has been found.

- **Utility:** Both the MLP and GANN models are applicable to prediction problems and are relatively easy to use. Moreover, the AutoGANN system and the MLP construction program that were used in this study were very useful in finding a good model, since both programs search automatically for a good model by using a model selection criterion. Search commences after initial parameters have been set by the user. These programs may, however, be expensive to obtain, since both require the SAS® software which is relatively expensive.

6.2 Summary of contributions

The following contributions were made by this study:

- The study provides a literature study on MLP and GANN models.
- A program was developed in the SAS® Macro Language to search for a good MLP model on the Adult, Boston Housing, Ozone, SO₄ and Spambase data sets. This program incorporated a modified version of

the N2C2S algorithm and a brute force method.

- Experimental results were obtained from various experiments that were conducted by using MLP and GANN models on the five data sets with the AutoGANN system and the custom-built MLP construction program.
- The results were analyzed and conclusions were drawn, which resulted in useful information about the performance of MLP and GANN models as well as information about the construction of MLP and GANN models. This information that was gained, resulted in some guidelines that can be used when choosing between an MLP and a GANN model.

6.3 Suggestions for future work

For future work, to gain more insight, more experiments are suggested on different data sets by using MLP and GANN models. In order to understand the differences and similarities between these two models better, the data sets that are chosen should also include regression tasks as well as classification tasks. The size of these data sets in terms of records and dimensionality should also vary from very small to very large data sets.

Additional research should be performed on the improvement of the modified N2C2S search algorithm to ensure that a good MLP model is found each time, without overfitting the data. The original N2C2S algorithm has measures that are incorporated to curb the overfitting effect.

Since the GANN architecture allows for the removal of unimportant input attributes from the model, research should be performed on the incorporation of a method for removing unimportant input attributes from the MLP models in order to reduce the complexity of MLP models.

Additive models (GANNs) in their simplest form are unable to model interactions (De Waal and Du Toit, 2011). Further research should be conducted on the use of explicit interaction terms or variables to give added insight into the model.

6.4 Conclusion

The objectives of this study were to investigate MLP and GANN models and to compare these two models by performing a number of experiments with them. The experimental results were then used to draw meaningful conclusions regarding these two models. These objectives have been reached by conducting a literature study on MLPs (Chapter 2) and GANNs (Chapter 3), performing experiments (Chapter 4) by using the AutoGANN system and a custom-built MLP program (Appendix A), and reaching significant conclusions that resulted in some guidelines for choosing the appropriate model (MLP or GANN) for a specific experiment (Chapter 5).

“Computers are famous for being able to do complicated things starting from simple programs.”

Seth Lloyd



MLP construction program code

Note that the following program code has been edited for printing purposes. Lines that begin with * are for comment purposes only.

```
options nosource nonotes nodate;
options pagesize=32767 mvarsize=max;
ods listing close;

*List of global variables used in the program
%global ac1 ac2 activation algorithm;
%global catalog count count2 criterion;
%global data set DataSelection Dir dset;
%global err errorfunc extype;
%global flag flag2;
%global hidnodes hMax;
%global i inputs inputclass inputvar;
%global k;
%global Library;
%global netoption nobs nomvar nvars;
%global params prelim;
%global r2 records;
%global scorerecords SplitTrain SplitVal;
```

```

%global target targettype time ttime tdelay;
%global u_aic u_ase u_correct u_correctperc u_sbc u_sse;

*Main program macro
%macro main;

/*.....USER SETTINGS.....*/
/*.....*/

    *data set Selection: Adult,House,Ozone,SO4,Spam;
    %let DataSelection = SO4;
    *Specify the directory of the data set;
    %let Dir = 'C:MLP Research';
    *Number of K-Fold cross-validation to be used;
    *(1 = user defined split, 0 = complete data set for training and scoring)
    %let k = 10;
    *Split size of training data set;
    %let SplitTrain = 70;
    *Split size of testing data set;
    %let SplitVal = 30;
    *Number of preliminary runs for random weight selection
    %let prelim = 10;
    *The netoptions to be used (like/dev);
    %let netoption = dev;
    *Criterion for selecting best model: AIC/SBC/VAVERR (when hidnodes = 0);
    %let criterion = SBC;
    *0 = use the search algorithm, >0 use the selected amount of hidden nodes
    %let hidnodes = 1;
    *Max number of nodes in the hidden layer;
    %let hMax = 15;

/*.....*/
/*.....*/

    *Run the settime macro;
    %settime;
    *Set experiment type for results file name;
    %let extype = &k._&hidnodes._&hMax;

    *Information about the Adult data set;
    %if &DataSelection = Adult %then %do;

```

```

libname Adult &Dir;
%let Library = Adult;
%let catalog = &Library..catalog;
%let data set = &Library..adult;
%let inputclass = workclass marital_status occupation
                  relationship race gender native_country;
%let inputvar = age fnlwgt educational_num
                capital_gain capital_loss hours_per_week;
%let target = status;
%let activation = Mlogistic;
%let errorfunc = mbe;
%let targettype = nom;
%let nomvar = 1;

*Creates the Adult data catalog;
proc dmdb batch data=&data set dmdbcat=&catalog;
    class &inputclass &target(DESC);
    var &inputvar;
run;

%end;

*Information about the House data set;
%else %if &DataSelection = House %then %do;
    libname House &Dir;
    %let Library = House;
    %let catalog = &Library..catalog;
    %let data set = &Library..Housing;
    %let inputclass = CHAS;
    %let inputvar = CRIM ZN INDUS NOX RM AGE DIS RAD TAX PTRAT B LSTAT;
    %let target = MEDV;
    %let activation = exp;
    %let errorfunc = poisson;
    %let targettype = int;
    %let nomvar = 1;

    *Creates the House data catalog;
    proc dmdb batch data=&data set dmdbcat=&catalog;
        class &inputclass;
        var &inputvar &target;
    run;

%end;

*Information about the Ozone data set;
%else %if &DataSelection = Ozone %then %do;

```

```

libname Ozone &Dir;
%let Library = Ozone;
%let catalog = &Library..catalog;
%let data set = &Library..Ozone;
%let inputs = VH Wind Humid Temp Ibh dpg Ibt Vis Doy;
%let target = Ozone;
%let activation = exp;
%let errorfunc = poisson;
%let targettype = int;
%let nomvar = 0;

*Creates the Ozone data catalog;
proc dmdb batch data=&data set dmdbcat=&catalog;
    var &inputs &target;
run;

%end;

*Information about the SO4 data set;
%else %if &DataSelection = SO4 %then %do;
    libname So4 &Dir;
    %let Library = So4;
    %let catalog = &Library..catalog;
    %let data set = &Library..So4;
    %let inputs = latitude longitude;
    %let target = so4;
    %let activation = exp;
    %let errorfunc = poisson;
    %let targettype = int;
    %let nomvar = 0;

    *Creates the SO4 data catalog;
    proc dmdb batch data=&data set dmdbcat=&catalog;
        var &inputs &target;
    run;

%end;

*Information about the Spam data set;
%else %if &DataSelection = Spam %then %do;
    libname Spam &Dir;
    %let Library = Spam;
    %let catalog = &Library..catalog;
    %let data set = &Library..spambase;
    %let inputs = make address all word_3d our over
        remove internet order mail;

```

```

%let inputs = &inputs receive will people
               report addresses free business email;
%let inputs = &inputs you credit your font
               word_000 money hp hpl george word_650;
%let inputs = &inputs lab labs telnet word_857
               data word_415 word_85 technology;
%let inputs = &inputs word_1999 parts pm direct
               cs meeting original project re edu;
%let inputs = &inputs table conference char_1
               char_2 char_3 char_4 char_5 char_6;
%let inputs = &inputs length_average
               length_longest length_total;

%let target = SPAM;
%let activation = Mlogistic;
%let errorfunc = mbe;
%let targettype = nom;
%let nomvar = 0;

*Creates the Spam data catalog;
proc dmdb batch data=&data set dmdbcat=&catalog;
    var &inputs;
    class &target(DESC);
run;

%end;

*Displays the options selected by the user;
%put *****;
%put ;
%put &DataSelection data set selected;
%if &hidnodes = 0 %then %do;
    %put Using search algorithm with &criteria as the search criterion;
    %put Max number of nodes = &hMax;
%end;
%else %do;
    %put Training neural network with &hidnodes to &hMax nodes;
%end;
%if &k = 0 %then %do;
    %put 100% of data set to be used for training;
%end;
%else %if &k = 1 %then %do;
    %put &SplitTrain / &SplitVal split for training/testing;
%end;

```



```

%else %if &k > 1 %then %do;
    %put &k Fold cross-validation are selected;
%end;
%put &prelim Preliminary runs are selected;
%put Netoptions set to &netoption;
%put ;
%put *****;

%if &hidnodes = 0 %then %do;

    *Variables used in the algorithm;
    %let flag = 0;
    %let flag2 = 0;
    %let count = 1;

    *Run the CreateTable macro;
    %CreateTables;

    %put Step 1;

    *Begin the algorithm by creating a neural network;
    *with random starting weights (repeated K times);
    %do i = 0 %to &k-1;
        *Run the RandomNeural macro;
        %RandomNeural;
        %put Step 2;
        %put %sysevalf(&i + 1)/&k.Fold;
        *Run the TotalTable macro;
        %TotalTable;
        *Run the average macro;
        %average;
        *Run the TotalAverageTable macro;
        %TotalAverageTable;
    %end;

    *Repeat the loop until the algorithm has finished;
    %do %until (&flag);

        %if ~(&flag2) %then %do;
            *Add an extra node to the hidden layer;
            %let count = %eval(&count + 1);
            %put Step 3;

```

```

%end;

*Repeat K times;
%do i = 0 %to &k-1;
    %if (&flag2) %then %do;
        %put Step 5b (&count nodes);
        *Run the RandomNeural macro;
        %RandomNeural;
    %end;
    %else %do;
        %put Step 4 (&count nodes);
        *Run the InestNeural macro;
        %InestNeural;
    %end;
    %put %sysevalf(&i + 1)/&k.Fold;
    *Run the TotalTable macro;
    %TotalTable;
%end;

*Run the average macro;
%average;

%if (&flag) %then %do;
    %if &criterion = SBC %then %do;
        *Get the SBC value (ac1 stays the same);
        proc sql;
            SELECT User_SBC INTO :ac2 FROM
                &Library..TempValidateResults2
                WHERE NODES = &count;
        QUIT;
    %end;
    %else %if &criterion = AIC %then %do;
        *Get the AIC value (ac1 stays the same);
        proc sql;
            SELECT User_AIC INTO :ac2 FROM
                &Library..TempValidateResults2
                WHERE NODES = &count;
        QUIT;
    %end;
    %else %if &criterion = VAVERR %then %do;
        *Get the VAVERR value (ac1 stays the same);
        proc sql;

```

```

SELECT Validate_VAVERR INTO :ac2 FROM
&Library..TempValidateResults2
WHERE NODES = &count;

QUIT;

%end;

%end;

%else %do;

%let num = %eval(&count-1);

%if &criterion = SBC %then %do;

*Get the SBC value for ac1 and ac2;

proc sql;

SELECT User_SBC INTO :ac1 FROM
&Library..TempValidateResults3
WHERE NODES = &num;

SELECT User_SBC INTO :ac2 FROM
&Library..TempValidateResults2
WHERE NODES = &count;

QUIT;

%end;

%else %if &criterion = AIC %then %do;

*Get the AIC value for ac1 and ac2;

proc sql;

SELECT User_AIC INTO :ac1 FROM
&Library..TempValidateResults3
WHERE NODES = &num;

SELECT User_AIC INTO :ac2 FROM
&Library..TempValidateResults2
WHERE NODES = &count;

QUIT;

%end;

%else %if &criterion = VAVERR %then %do;

*Get the VAVERR value for ac1 and ac2;

proc sql;

SELECT Validate_VAVERR INTO :ac1 FROM
&Library..TempValidateResults3
WHERE NODES = &num;

SELECT Validate_VAVERR INTO :ac2 FROM
&Library..TempValidateResults2
WHERE NODES = &count;

QUIT;

%end;

%end;

```

```

%put &ac2 < &ac1 ;

*Evaluate the VAVERR from the current and the previous MLP created;
%if %sysevalf(%sysevalf(&ac2) > %sysevalf(&ac1)) %then %do;
    %put False;
    %if (&flag2) %then %do;
        *Stop the algorithm if new MLP with random weights;
        *is worse than the previous MLP;
        %let flag = 1;
        %put Step 6 (Terminate Program);
    %end;
    *Run MLP with random weights;
%else %do;
    %put Step 5b (New MLP with &count nodes and
        random weights);
    %let flag2 = 1;
%end;
%end;
%else %do;
    %put True;
    *Run MLP with an extra hidden node;
    %put Step 5a (New Neural Network with &count + 1 nodes
        and initial weights set to previous values);
    *Run the TotalAverageTable macro;
    %TotalAverageTable;
    %let flag2 = 0;
    %let ac1 = &ac2;
%end;

%if %sysevalf(&count > &hMax) %then %do;
    *Stop the algorithm if hidden nodes > max nodes;
    %let flag = 1;
%end;

%end;

%end;
%else %do;
    *Run the CreateTable macro;
    %CreateTables;
    %let count2 = 0;
    %do hidnodes = &hidnodes %to &hMax;

```

```

%let count2 = %sysevalf(&count2 + 1);
%let count = 0;
%put Training Neural Network With &hidnodes Hidden Nodes;
%do i = 0 %to &k-1;
    %let count = %sysevalf(&count + 1);
    *Run the ManualRandomNeural macro;
    %ManualRandomNeural;
    %put %sysevalf(&i + 1)/&k.Fold;
    *Run the "TotalTable" macro;
    %TotalTable;
%end;
%let count = &hidnodes;
*Run the average macro;
%average;
%if &count2 = 1 %then %do;
    *Combines the Average Train- and Validation Fit statistics;
    data &Library..&Library.Results&extype;
        merge &Library..TempTrainResults2
                &Library..TempValidateResults2;
    run;
%end;
%else %do;
    *Combines the Average Train- and Validation Fit statistics;
    data &Library..Results2;
        merge &Library..TempTrainResults2
                &Library..TempValidateResults2;
    run;
    *Combines the new results with the other results;
    data &Library..&Library.Results&extype;
        set &Library..&Library.Results&extype &Library..Results2;
    run;
%end;
%end;

%end;

*Run the gettime macro;
%gettime;
*Run the showtime macro;
%showtime;

*Delete certain data sets used;
proc data sets nolist library=&Library;

```

```

        delete train0-train99 test0-test99;
        delete usertemp;
        delete logs;
        delete temptrainresults1;
        delete temptrainresults2;
        delete temptrainresults3;
        delete tempvalidateresults1;
        delete tempvalidateresults2;
        delete tempvalidateresults3;
        delete trainfit;
        delete validatefit;
        delete temp;
        delete start;
        delete _namedat;
        delete score;
        delete catalog/memtype=catalog;
        delete Results2;
run;

        *Makes a beep sound when the program has finished;
        data _null_;
call sound(400,80);
        run;

%mend main;

*Creates the testing and training data sets;
%macro CreateTable;

        *Delete previously created data sets;
        proc data sets nolist library=&Library;
                delete train0-train99 test0-test99;
run;

        *Create a Training-Testing data set split;
        %if &k = 1 %then %do;
                %partition2(&data set,&SplitTrain,&SplitVal,
                        &Library..train0,&Library..test0);
        %end;
        %else %if &k = 0 %then %do;
                data &Library..train0;
                        set &data set;

```

```

        run;
        data &Library..test0;
            set &data set;
        run;
        %let k = 1;
    %end;
    *Choose records for K data sets;
    %else %do;
        data &Library..temp;
            set &data set;
            cv = int(ranuni(0)/(1/&k));
        run;

        %do i = 0 %to &k-1;
            *Create K training data sets;
            data &Library..train&i;
                set &Library..temp;
                if cv ne &i then output;
            run;

            *Create K testing data sets;
            data &Library..test&i;
                set &Library..temp;
                if cv eq &i then output;
            run;
        %end;
    %end;

%mend CreateTable;

*Creates tables that contain the average statistics of the;
*K-fold cross-validations done in all the iterations;
%macro TotalAverageTable;

    %if &count = 1 %then %do;
        data &Library..TempValidateResults3;
            set &Library..TempValidateResults2;
        run;

        data &Library..TempTrainResults3;
            set &Library..TempTrainResults2;
        run;
    %end;

```

```

%end;

%else %do;

    data &Library..TempValidateResults3;
        set &Library..TempValidateResults3 &Library..TempValidateResults2;
    run;

    data &Library..TempTrainResults3;
        set &Library..TempTrainResults3 &Library..TempTrainResults2;
    run;

%end;

*Combines the Average Train- and Validation Fit statistics;
data &Library..&Library.Results&extype;
    merge &Library..TempTrainResults3 &Library..TempValidateResults3;
run;

%mend TotalAverageTable;

*Creates tables that contain the average statistics K-fold statistics;
*done in a single iteration of the algorithms;
%macro average;

    proc sql;

        create table

            &Library..TempValidateResults2 as

                SELECT &count as NODES,
                    AVG(_VASE_) as Validate_ASE,
                    AVG(_VAVER_) as Validate_VAVER,
                    AVG(_VDIV_) as Validate_DIV,
                    AVG(_VERR_) as Validate_ERR,
                    AVG(_VMAX_) as Validate_MAX,
                    AVG(_VMSE_) as Validate_MSE,
                    AVG(_VNOBS_) as Validate_NOBS,
                    AVG(_VRASE_) as Validate_RASE,
                    AVG(_VRMSE_) as Validate_RMSE,
                    AVG(_VSSE_) as Validate_SSE,
                    AVG(_VSUMW_) as Validate_SUMW,
                    AVG(_VMISC_) as Validate_MISC,
                    AVG(_VWRONG_) as Validate_WRONG,
                    AVG(r2) as User_R2,
                    AVG(User_ASE) as User_ASE,

```



```

        AVG(User_SSE) as User_SSE,
        AVG(User_AIC) as User_AIC,
        AVG(User_SBC) as User_SBC,
        AVG(User_CorrectPerc) as User_CorrectPerc
FROM &Library..TempValidateResults1;

```

```

create table

```

```

    &Library..TempTrainResults2 as

```

```

        SELECT &count as NODES,
            AVG(_DFT_) as Train_DFT,
            AVG(_DFE_) as Train_DFE,
            AVG(_DFM_) as Train_DFM,
            AVG(_NW_) as Train_NW,
            AVG(_AIC_) as Train_AIC,
            AVG(_SBC_) as Train_SBC,
            AVG(_ASE_) as Train_ASE,
            AVG(_MAX_) as Train_MAX,
            AVG(_DIV_) as Train_DIV,
            AVG(_NOBS_) as Train_NOBS,
            AVG(_RASE_) as Train_RASE,
            AVG(_SSE_) as Train_SSE,
            AVG(_SUMW_) as Train_SUMW,
            AVG(_FPE_) as Train_FPE,
            AVG(_MSE_) as Train_MSE,
            AVG(_RFPE_) as Train_RFPE,
            AVG(_RMSE_) as Train_RMSE,
            AVG(_VAVERR_) as Train_VAVERR,
            AVG(_ERR_) as Train_ERR
        FROM &Library..TempTrainResults1;

```

```

QUIT;

```

```

%mend average;

```

```

*Creates tables that contains the statistics of the K-fold cross-validation;

```

```

%macro TotalTable;

```

```

    *Run the calculater2 macro;

```

```

    %calculater2;

```

```

    *Run the UserCalculate macro;

```

```

    %UserCalculate;

```

```

data &Library..validatefit;
    set &Library..validatefit;
    R2 = &r2;
    User_ASE = &u_ase;
    User_SSE = &u_sse;
    User_AIC = &u_aic;
    User_SBC = &u_sbc;
    User_CorrectPerc = &u_correctperc;
run;

%if &i = 0 %then %do;
    data &Library..TempValidateResults1;
        set &Library..validatefit(firstobs=2);
    run;

    data &Library..TempTrainResults1;
        set &Library..trainfit(firstobs=2);
    run;
%end;
%else %do;
    data &Library..TempValidateResults1;
        set &Library..TempValidateResults1 &Library..validatefit(firstobs=2);
    run;

    data &Library..TempTrainResults1;
        set &Library..TempTrainResults1 &Library..trainfit(firstobs=2);
    run;
%end;

%mend TotalTable;

*Creates and run an MLP with random initial weights and;
*user specified number of hidden nodes;
%macro ManualRandomNeural;

proc neural data=&Library..train&i dmdbcat=&catalog random=0;

    %if(&nomvar) %then %do;
        *Nominal input variables;
        input &inputclass/ level=nom id=in1;
        *Integer input variables;
        input &inputvar/ level=int id=in2;

```

```

*Target variable, activation function, error function;
target &target/ level=&targettype id=out
act=&activation error=&errorfunc bias;
netoptions object=&netoption;
*Units in hidden layer;
hidden %eval(&hidnodes) / bias id=hid1;

*Connect nominal input variables with hidden layer;
connect in1 hid1;
*Connect Integer input variables with hidden layer;
connect in2 hid1;
*Connect hidden layer with output;
connect hid1 out;
*Number of preliminary runs;
prelim %eval(&prelim);
train outfit=&Library..trainfit outtest=start maxiter=10000;
score data=&Library..test&i
nodmdb out=&Library..Score
outfit=&Library..validatefit
role=validation;

%end;
%else %do;
*Integer input variables;
input &inputs/ level=int id=in;
*Target variable, activation function, error function;
target &target/ level=&targettype id=out
act=&activation error=&errorfunc bias;
netoptions object=&netoption;
*Units in hidden layer;
hidden %eval(&hidnodes) / bias id=hid1;

*Connect Integer input variables with hidden layer;
connect in hid1;
*Connect hidden layer with output;
connect hid1 out;
*Number of preliminary runs;
prelim %eval(&prelim);
train outfit=&Library..trainfit outtest=start maxiter=10000;
score data=&Library..test&i
nodmdb out=&Library..Score
outfit=&Library..validatefit
role=validation;

```

```

        %end;

run;

%mend ManualRandomNeural;

*Creates and run an MLP with random initial weights;
%macro RandomNeural;

proc neural data=&Library..train&i dmdbcat=&catalog random=0;

    %if(&nomvar) %then %do;
        *Nominal input variables;
        input &inputclass/ level=nom id=in1;
        *Integer input variables;
        input &inputvar/ level=int id=in2;
        *Target variable, activation function, error function;
        target &target/ level=&targettype id=out
        act=&activation error=&errorfunc bias;
        netoptions object=&netoption;
        *Units in hidden layer;
        hidden %eval(&count) / bias id=hid1;

        *Connect nominal input variables with hidden layer;
        connect in1 hid1;
        *Connect Integer input variables with hidden layer;
        connect in2 hid1;
        *Connect hidden layer with output;
        connect hid1 out;
        *Number of preliminary runs;
        prelim %eval(&prelim);
        train outfit=&Library..trainfit outtest=start maxiter=10000;
        score data=&Library..test&i
        nodmdb out=&Library..Score
        outfit=&Library..validatefit
        role=validation;
    %end;
    %else %do;
        *Integer input variables;
        input &inputs/ level=int id=in;
        *Target variable, activation function, error function;
        target &target/ level=&targettype id=out

```

```

        act=&activation error=&errorfunc bias;
        netoptions object=&netoption;
        *Units in hidden layer;
        hidden %eval(&count) / bias id=hid1;

        *Connect Integer input variables with hidden layer;
        connect in hid1;
        *Connect hidden layer with output;
        connect hid1 out;
        *Number of preliminary runs;
        prelim %eval(&prelim);
        train outfit=&Library..trainfit outest=start maxiter=10000;
        score data=&Library..test&i
        nodmdb out=&Library..Score
        outfit=&Library..validatefit
        role=validation;
    %end;

run;

%mend RandomNeural;

*Creates and run an MLP with initial weights set to previous best;
%macro InestNeural;

    proc neural data=&Library..train&i dmdbcat=&catalog random=0;
        %if(&nomvar) %then %do;
            *Nominal input variables;
            input &inputclass/ level=nom id=in1;
            *Integer input variables;
            input &inputvar/ level=int id=in2;
            *Target variable, activation function, error function;
            target &target/ level=&targettype id=out
            act=&activation error=&errorfunc bias;
            netoptions object=&netoption;
            *Units in hidden layer;
            hidden %eval(&count) / bias id=hid1;

            *Connect nominal input variables with hidden layer;
            connect in1 hid1;
            *Connect Integer input variables with hidden layer;
            connect in2 hid1;

```

```

        *Connect hidden layer with output;
        connect hid1 out;
        *Set the initial weights to previously saved weights;
        initial inest=start;
        *Number of preliminary runs;
        prelim %eval(&prelim);
        train outfit=&Library..trainfit outest=start maxiter=10000;
        score data=&Library..test&i
        nodmdb out=&Library..Score
        outfit=&Library..validatefit
        role=validation;
    %end;
    %else %do;
        *Integer input variables;
        input &inputs/ level=int id=in;
        *Target variable, activation function, error function;
        target &target/ level=&targettype id=out
        act=&activation error=&errorfunc bias;
        netoptions object=&netoption;
        *Units in hidden layer;
        hidden %eval(&count) / bias id=hid1;

        *Connect Integer input variables with hidden layer;
        connect in hid1;
        *Connect hidden layer with output;
        connect hid1 out;
        *Set the initial weights to previously saved weights;
        initial inest=start;
        *Number of preliminary runs;
        prelim %eval(&prelim);
        train outfit=&Library..trainfit outest=start maxiter=10000;
        score data=&Library..test&i
        nodmdb out=&Library..Score
        outfit=&Library..validatefit
        role=validation;
    %end;

run;

%mend InestNeural;

*Gets the current system time;

```

```

%macro settime;

    %let ttime = %sysfunc(datetime());

%mend settime;

*Calculates the time passed;
%macro gettime;

    %local a b c d e;
    %let tdelay = %sysevalf(%sysfunc(datetime()) - &ttime);
    %let a = &tdelay;
    %let b = %sysevalf(&a / 86400, integer);
    %let a = %sysevalf(&a - 86400 * &b);
    %let c = %sysevalf(&a / 3600, integer);
    %let a = %sysevalf(&a - 3600 * &c);
    %let d = %sysevalf(&a / 60, integer);
    %let e = %sysevalf(%sysfunc(round(&a - 60 * &d, 0.01)));
    %let time = &b:&c:&d:&e;

%mend gettime;

*Shows the time in the log;
%macro showtime;

    %put ;
    %put elapsed time = &time;
    %put ;

%mend showtime;

*Calculates the Correlation Coefficient (R2);
%macro calculater2;

    %local t;
    %let t = &target;

    proc sql noprint;
        select mean(&t.) into :mean
        from &Library..score;
    run;

```

```

data _null_;
    set &Library..score;
    retain sum1 0;
    retain sum2 0;
    sum1 + (&t. - p_&t.)**2;
    sum2 + (&t. - &mean)**2;
    call symput ('sum1', sum1);
    call symput ('sum2', sum2);
run;

%let r2 = %sysevalf(1 - (&sum1 / &sum2));

%mend calculater2;

*Gets the number of observations and;
*variables from a specified data set;
%macro obsnvars(ds,nvarsp,nobsp);

    %let dset=&ds;
    %let dsid = %sysfunc(open(&dset));

    %if &dsid %then %do;
        %let nobsp =%sysfunc(attrn(&dsid,NOBS));
        %let nvarsp=%sysfunc(attrn(&dsid,NVARS));
        %let rc = %sysfunc(close(&dsid));
    %end;
    %else
        %put Open for data set &dset failed - %sysfunc(sysmsg());
    %end;

%mend obsnvars;

*Partition a specified data set into two;
*data sets with specified percentage of;
*the original data set;
%macro partition2(source,p1,p2,set1,set2);

    %let seed = 0;

    %obsnvars(&source,nvars,nobs);

    %let cutoff1 = %sysevalf(&nobs * &p1 / 100, ceil);
    %let cutoff2 = %eval(&nobs - &cutoff1);

```



```

data &set1 &set2;
    drop _c00: ;
    set &source;
    if (ranuni(&seed) * 1000 < %sysevalf (&p1 * 10, ceil)
and _c000001 < &cutoff1) then do;
        _c000001 + 1;
        output &set1;
    end;
    else
        if _c000002 < &cutoff2 then do;
            _c000002 + 1;
            output &set2;
        end;
        else do;
            _c000001 + 1;
            output &set1;
        end;
run;

%mend partition2;

*Calculations to verify statistics;
%macro UserCalculate;

    %let p=P_;

    data &Library..UserTemp;
        set &Library..score;
        %if &targettype = nom %then %do;
            Round_err = Round(&p.&target.1,1);
            User_Err = &target - &p.&target.1;
        %end;
        %else %do;
            Round_err = Round(&p.&target,1);
            User_Err = &target - &p.&target;
        %end;
        User_SqrErr = (User_Err**2);
        if &target = Round_err then do
            Correct = 1;
        end;
    else do

```

```

        Correct = 0;

    end;

run;

proc sql;

    SELECT AVG(User_SqrErr) INTO :u_ase FROM &Library..UserTemp;
    SELECT SUM(User_SqrErr) INTO :u_sse FROM &Library..UserTemp;
    SELECT SUM(Correct) INTO :u_correct FROM &Library..UserTemp;
    SELECT _DFM_ INTO :params FROM &Library..trainfit;
    SELECT _DFT_ INTO :records FROM &Library..trainfit;

QUIT;

%let err = &u_ase;

data logs;

    LogX = LOG(&err/&records);
    LogN = LOG(&records);

run;

%local LogX;
%local LogN;

proc sql;

    SELECT LogX INTO :LogX FROM logs;
    SELECT LogN INTO :LogN FROM logs;

QUIT;

%if %sysevalf(&netoption = like) %then %do;

    %let u_aic = %sysevalf((2 * &err) + (2 * &params));
    %let u_sbc = %sysevalf((2 * &err) + (&params * &LogN));

%end;

%else %if %sysevalf(&netoption = dev) %then %do;

    %let u_aic = %sysevalf((&records * &LogX) + (2 * &params));
    %let u_sbc = %sysevalf((&records * &LogX) + (&params * &LogN));

%end;

%let dsid=%sysfunc(open(&Library..score));
%let num=%sysfunc(attrn(&dsid,nlobs));
%let rc=%sysfunc(close(&dsid));
%let scorerecords = &num;
%let u_correctperc = %sysevalf(%eval(&u_correct) /
                                %eval(&scorerecords) * 100);

%mend UserCalculate;

*Run the main macro;

%main;

```

“A man should look for what is, and not for what he thinks should be.”

Albert Einstein

B

MLP brute force method results

In this appendix, the results from the brute force MLP experiments (BRUTE100, BRUTE70 and BRUTE10) are shown in table format. These results can be used as a baseline for the results of the MLP experiments that utilized the modified N2C2S algorithm (NCS100, NCS70 and NCS10). The result tables are arranged according to the data set and the experiment that was conducted. First, the results from the experiments that were conducted with the Adult data set by using the brute force method are shown (BRUTE100, BRUTE70 and then BRUTE10), then the results from the brute force experiments with the Boston Housing data set are presented (BRUTE100, BRUTE70 and then BRUTE10). This is followed by the experimental results from the Ozone data set in which the brute force method was used (BRUTE100, BRUTE70 and then BRUTE10), then the results from the brute force experiments that were conducted with the SO₄ data set are presented (BRUTE100, BRUTE70 and then BRUTE10). Finally, the brute force experimental results with the Spambase data set are shown (BRUTE100, BRUTE70 and then BRUTE10).

Number of hidden neurons	Number of parameters	SBC	Accuracy (%)
1	16	-586 248.46	84.89
2	31	-586 516.43	85.25
3	46	-586 192.00	85.32
4	61	-586 075.60	85.69
5	76	-585 556.71	85.76
6	91	-584 861.46	85.93
7	106	-584 261.18	85.92
8	121	-583 100.14	85.86
9	136	-581 003.01	85.41
10	151	-580 131.39	85.48
11	166	-581 321.90	86.15
12	181	-578 849.42	85.68
13	196	-577 756.22	85.61
14	211	-578 467.37	86.14
15	226	-577 867.09	86.18
16	241	-576 583.95	86.20
17	256	-575 861.57	86.16
18	271	-574 813.58	86.05
19	286	-574 312.79	86.27
20	301	-573 364.29	86.22
21	316	-572 262.04	86.06
22	331	-571 471.83	86.18
23	346	-569 740.99	85.86
24	361	-569 493.44	86.13
25	376	-568 924.81	86.26
26	391	-568 148.16	86.20
27	406	-567 710.68	86.43
28	421	-566 762.18	86.34
29	436	-565 506.18	86.24
30	451	-565 321.94	86.43

Table B.1: BRUTE100 results on the Adult data set

Number of hidden neurons	Number of parameters	VAVERR	Accuracy (%)
1	16	0.331499	84.54
2	31	0.319356	84.97
3	46	0.311373	85.33
4	61	0.307451	85.52
5	76	0.309672	85.64
6	91	0.310592	85.58
7	106	0.308023	85.53
8	121	0.307989	85.49
9	136	0.309302	85.55
10	151	0.307018	85.54
11	166	0.30785	85.52
12	181	0.307723	85.60
13	196	0.306812	85.56
14	211	0.306202	85.76
15	226	0.306466	85.63
16	241	0.304596	85.69
17	256	0.308354	85.38
18	271	0.307312	85.43
19	286	0.308569	85.44
20	301	0.304105	85.68
21	316	0.307371	85.69
22	331	0.306259	85.43
23	346	0.308959	85.49
24	361	0.308886	85.52
25	376	0.310335	85.57
26	391	0.304095	85.61
27	406	0.308271	85.52
28	421	0.308054	85.51
29	436	0.309942	85.48
30	451	0.306603	85.58

Table B.2: BRUTE70 results on the Adult data set

Number of hidden neurons	Number of parameters	VAVERR	Accuracy (%)
1	16	0.327534	84.84
2	31	0.319237	85.11
3	46	0.313656	85.40
4	61	0.311689	85.46
5	76	0.311471	85.48
6	91	0.309602	85.66
7	106	0.312170	85.37
8	121	0.309848	85.57
9	136	0.310119	85.63
10	151	0.311394	85.49
11	166	0.310701	85.57
12	181	0.310618	85.50
13	196	0.310921	85.42
14	211	0.309766	85.57
15	226	0.309320	85.60
16	241	0.311408	85.57
17	256	0.309925	85.62
18	271	0.309190	85.58
19	286	0.309300	85.62
20	301	0.309015	85.62
21	316	0.310431	85.62
22	331	0.308925	85.70
23	346	0.310947	85.51
24	361	0.309784	85.56
25	376	0.310518	85.47
26	391	0.309945	85.53
27	406	0.309314	85.72
28	421	0.309845	85.67
29	436	0.310976	85.60
30	451	0.310944	85.53

Table B.3: BRUTE10 results on the Adult data set

Number of hidden neurons	Number of parameters	SBC	MSE
1	16	-1 662.31	15.555990
2	31	-1 858.93	8.769547
3	46	-1 977.08	5.773066
4	61	-1 956.36	5.000743
5	76	-1 899.36	4.653638
6	91	-1 929.77	3.643612
7	106	-1 878.56	3.352132
8	121	-1 861.06	2.885232
9	136	-1 789.30	2.764449
10	151	-1 704.41	2.718357
11	166	-1 761.07	2.020741
12	181	-1 669.45	2.013672
13	196	-1 711.17	1.541766
14	211	-1 606.65	1.576033
15	226	-1 606.15	1.311662

Table B.4: BRUTE100 results on the Boston Housing data set

Number of hidden neurons	Number of parameters	VAERR	MSE
1	16	0.781956	18.919640
2	31	1.835688	200.914400
3	46	0.902232	47.047140
4	61	1.021484	39.108760
5	76	0.601183	14.093970
6	91	5.972047	1 303.487000
7	106	0.594371	15.918820
8	121	0.670548	15.938900
9	136	0.922856	28.352260
10	151	0.840043	20.351250
11	166	1.144861	33.489850
12	181	1.231336	35.282320
13	196	0.965153	23.985820
14	211	1.496066	46.057140
15	226	1.608093	60.631880

Table B.5: BRUTE70 results on the Boston Housing data set

Number of hidden neurons	Number of parameters	VAVERR	MSE
1	16	0.707133	16.969953
2	31	0.764980	34.694599
3	46	0.496325	10.938516
4	61	0.514646	13.530875
5	76	0.486868	11.050678
6	91	0.569609	14.504891
7	106	0.539869	13.055272
8	121	0.695531	18.737204
9	136	0.553737	13.139862
10	151	0.683851	16.473891
11	166	0.740993	19.381405
12	181	0.987968	24.276840
13	196	1.095268	29.491034
14	211	0.990657	29.880732
15	226	0.939331	25.304731

Table B.6: BRUTE10 results on the Boston Housing data set

Number of hidden neurons	Number of parameters	SBC	MSE
1	12	-938.19	15.567830
2	23	-934.81	12.963540
3	34	-946.11	10.325550
4	45	-917.89	9.270305
5	56	-897.58	8.125972
6	67	-847.14	7.803900
7	78	-829.10	6.793455
8	89	-808.92	5.952597
9	100	-847.42	4.366050
10	111	-833.72	3.751110
11	122	-656.85	5.284338
12	133	-705.59	3.757401
13	144	-642.35	3.751188
14	155	-676.87	2.784722
15	166	-660.91	2.409009

Table B.7: BRUTE100 results on the Ozone data set

Number of hidden neurons	Number of parameters	VAVERR	MSE
1	12	1.244689	16.734990
2	23	1.082693	14.557900
3	34	1.130124	15.256380
4	45	1.183423	16.722250
5	56	1.710985	25.560680
6	67	1.460380	22.779450
7	78	2.551725	42.313030
8	89	2.239165	37.606780
9	100	2.394974	41.911400
10	111	2.801250	48.464590
11	122	3.372007	67.772670
12	133	5.261488	115.724500
13	144	4.711515	99.501110
14	155	5.123945	120.854700
15	166	3.990240	69.250170

Table B.8: BRUTE70 results on the Ozone data set

Number of hidden neurons	Number of parameters	VAVERR	MSE
1	12	1.346970	16.879764
2	23	1.273780	16.614138
3	34	1.139797	15.106363
4	45	1.229552	16.172386
5	56	1.333431	17.873299
6	67	1.405833	18.781528
7	78	1.824317	23.881712
8	89	1.730560	31.552813
9	100	2.440511	41.056634
10	111	1.968626	29.122909
11	122	2.542599	40.499850
12	133	2.577597	43.656292
13	144	2.815467	43.622020
14	155	3.988830	139.881048
15	166	3.195163	50.825107

Table B.9: BRUTE10 results on the Ozone data set

Number of hidden neurons	Number of parameters	SBC	MSE
1	5	-1 105.41	0.322067
2	9	-1 192.88	0.175952
3	13	-1 287.82	0.092193
4	17	-1 284.33	0.083718
5	21	-1 271.91	0.079911
6	25	-1 263.69	0.074511
7	29	-1 244.87	0.073712
8	33	-1 247.63	0.064640
9	37	-1 201.55	0.074465
10	41	-1 175.45	0.076724
11	45	-1 199.89	0.059605
12	49	-1 185.35	0.057575
13	53	-1 173.11	0.054901
14	57	-1 153.22	0.054637
15	61	-1 141.79	0.051866

Table B.10: BRUTE100 results on the SO₄ data set

Number of hidden neurons	Number of parameters	VAVERR	MSE
1	5	0.147243	0.256515
2	9	0.121748	0.209630
3	13	0.080277	0.104616
4	17	0.054677	0.092898
5	21	0.055323	0.099312
6	25	0.061801	0.102103
7	29	0.081793	0.146781
8	33	0.086488	0.135128
9	37	0.081401	0.144120
10	41	0.120920	0.229561
11	45	0.076118	0.146338
12	49	0.130656	0.176114
13	53	0.077771	0.147034
14	57	0.076056	0.125791
15	61	0.073424	0.125603

Table B.11: BRUTE70 results on the SO₄ data set

Number of hidden neurons	Number of parameters	VAVERR	MSE
1	5	0.189700	0.335896
2	9	0.126817	0.197753
3	13	0.079668	0.109535
4	17	0.063151	0.101856
5	21	0.062345	0.107044
6	25	0.076598	0.126063
7	29	0.074230	0.131305
8	33	0.075888	0.127790
9	37	0.082921	0.130947
10	41	0.079019	0.135428
11	45	0.073381	0.123816
12	49	0.079877	0.130903
13	53	0.091622	0.129548
14	57	0.089610	0.141690
15	61	0.084496	0.130659

Table B.12: BRUTE10 results on the SO₄ data set

Number of hidden neurons	Number of parameters	SBC	Accuracy (%)
1	60	-52 556.83	94.31
2	119	-52 684.04	94.87
3	178	-53 547.87	96.26
4	237	-55 064.12	97.44
5	296	-55 043.63	97.91
6	355	-54 700.62	98.00
7	414	-57 478.46	99.11
8	473	-59 347.61	99.43
9	532	-55 409.83	98.80
10	591	-56 964.27	99.20
11	650	-56 178.18	99.17
12	709	-57 526.04	99.43
13	768	-56 632.28	99.30
14	827	-55 963.06	99.39
15	886	-57 676.69	99.65
16	945	-56 407.95	99.50
17	1 004	-56 539.76	99.61
18	1 063	-56 500.42	99.65
19	1 122	-55 794.38	99.59
20	1 181	-55 357.97	99.65
21	1 240	-55 344.85	99.63
22	1 299	-54 213.22	99.59
23	1 358	-54 294.42	99.67
24	1 417	-50 841.59	99.37
25	1 476	-53 272.06	99.67
26	1 535	-52 662.19	99.67
27	1 594	-52 377.14	99.65
28	1 653	-52 148.23	99.67
29	1 712	-51 401.25	99.67
30	1 771	-51 940.25	99.70

Table B.13: BRUTE100 results on the Spambase data set

Number of hidden neurons	Number of parameters	VAVERR	Accuracy (%)
1	60	0.190574	93.77
2	119	0.190246	93.70
3	178	0.219910	93.70
4	237	0.239873	93.12
5	296	0.731697	92.25
6	355	0.471034	92.10
7	414	0.608570	91.38
8	473	0.893028	92.46
9	532	0.579423	91.67
10	591	0.526023	92.32
11	650	0.440628	93.55
12	709	0.399910	92.39
13	768	0.367853	93.19
14	827	0.354482	93.91
15	886	0.391671	92.90
16	945	0.354328	93.77
17	1 004	0.439693	92.46
18	1 063	0.375745	93.26
19	1 122	0.397949	93.77
20	1 181	0.399680	92.83
21	1 240	0.428946	94.20
22	1 299	0.351746	94.49
23	1 358	0.395222	93.91
24	1 417	0.374867	93.26
25	1 476	0.355396	94.13
26	1 535	0.326509	94.06
27	1 594	0.280652	94.28
28	1 653	0.384702	93.04
29	1 712	0.344357	94.35
30	1 771	0.415189	93.19

Table B.14: BRUTE70 results on the Spambase data set

Number of hidden neurons	Number of parameters	VAVERR	Accuracy (%)
1	60	0.198754	93.44
2	119	0.205768	93.39
3	178	0.238180	93.36
4	237	0.251255	93.41
5	296	0.392936	92.74
6	355	0.629614	92.44
7	414	0.862891	91.86
8	473	1.034209	91.71
9	532	0.432075	92.82
10	591	0.454756	93.06
11	650	0.469864	92.70
12	709	0.471734	93.06
13	768	0.465625	92.97
14	827	0.489211	92.73
15	886	0.482830	93.00
16	945	0.472985	93.22
17	1 004	0.507718	93.11
18	1 063	0.468729	93.07
19	1 122	0.431768	93.56
20	1 181	0.421669	93.23
21	1 240	0.451414	93.24
22	1 299	0.425484	93.75
23	1 358	0.452078	93.45
24	1 417	0.423939	93.44
25	1 476	0.430619	93.40
26	1 535	0.428210	93.45
27	1 594	0.419633	93.30
28	1 653	0.463558	93.61
29	1 712	0.440156	93.26
30	1 771	0.406101	93.28

Table B.15: BRUTE10 results on the Spambase data set

Bibliography

- Akaike, H. (1969), 'Fitting autoregressive models for prediction', *Annals of the Institute of Statistical Mathematics* **21**, 243–247.
- Akaike, H. (1974), 'A new look at the statistical model identification', *IEEE Transactions on Automatic Control* **AC-19**, 716–723.
- Akaike, H. (1978), 'A bayesian analysis of the minimum aic procedure', *Annals of the Institute of Statistical Mathematics* **30, Part A**, 9–14.
- Allen, D. M. (1974), 'The relationship between variable selection and data augmentation and a method for prediction', *Technometrics* **16**, 125–127.
- Anders, U. and Korn, O. (1999), 'Model selection in neural networks', *Neural Networks* **12**, 309–323.
- Ash, T. (1989), Dynamic node creation in back-propagation networks, Technical Report 8901, Institute for Cognitive Science, UCSD, La Jolla.
- Basheer, I. A. and Hajmeer, M. (2000), 'Artificial neural networks: fundamentals, computing, design, and application', *Journal of Microbiological Methods* **43**, 3–31.
- Bell, D., Walker, J., O'Connor, G., Orrel, J. and Tibshirani, R. J. (1989), 'Spinal deformation following multi-level thoracic and lumbar laminectomy in children', Submitted for publication.
- Bellman, R. E. (1961), *Adaptive Control Processes: A Guided Tour*, Princeton University Press, Princeton, NJ.
- Berk, K. N. and Booth, D. E. (1995), 'Seeing a curve in multiple regression', *Technometrics* **37**, 385–398.
- Berry, M. J. A. and Linoff, G. (1997), *Data Mining Techniques for Marketing, Sales, and Customer Support*, John Wiley & Sons, Inc., New York.
- Bhansali, R. J. and Downham, D. Y. (1977), 'Some properties of the order of an autoregressive model selected by a generalization of akaike's epf criterion', *Biometrika* **64**, 547–551.
- Blum, A. L. and Langey, P. (1997), 'Selection of relevant features and examples in machine learning', *Artificial Intelligence* **97**, 245–271.

- Bose, N. K. and Garga, A. K. (1993), 'Neural network design using voronoi diagrams', *IEEE Transactions on Neural Networks* **4**, 778–787.
- Breiman, L. and Friedman, J. H. (1985), 'Estimating optimal transformations for multiple regression and correlation', *Journal of the American Statistical Association* **80**, 580–619.
- Burnham, K. P. and Anderson, D. R. (2002), *Model Selection and Multi-model Inference: A Practical Information-Theoretic Approach*, 2nd edn, Springer, New York.
- Cai, Z. and Tsai, C. (1999), 'Diagnostics for nonlinearity in generalized linear models', *Computational Statistics and Data Analysis* **29**, 445–469.
- Campher, E. S. (2008), Comparing generalised additive neural networks with decision trees and alternating conditional expectations, Master's thesis, North-West University, Potchefstroom Campus, South Africa.
- Carpenter, A. (2004), *Carpenter's Complete Guide to the SAS® Macro Language*, 2nd edn, SAS Institute Inc., Cary, NC.
- Coppin, B. (2004), *Artificial Intelligence Illuminated*, Jones and Bartlett, Sudbury, MA.
- DARPA (1988), *Defence Advanced Research Projects Agency: Neural Network Study*, AFCEA International Press, Fairfax, VA.
- De Waal, D. A. and Du Toit, J. V. (2011), 'Automation of generalized additive neural networks for predictive data mining', Submitted for publication.
- Du Toit, J. V. (2006), Automated Construction of Generalized Additive Neural Networks for Predictive Data Mining, PhD thesis, North-West University, Potchefstroom Campus, South Africa.
- Ezekiel, M. (1924), 'A method for handling curvilinear correlation for any number of variables', *Journal of the American Statistical Association* **19**, 431–453.
- Fahlman, S. E. and Lebiere, C. (1990), The cascade-correlation architecture, in D. Touretzky, ed., 'Advances in Neural Information Processing Systems (Denver, 1989) (2)', Morgan Kaufmann, San Mateo, pp. 524–532.
- Faraway, J. J. (1992), 'On the cost of data analysis', *Journal of Computational and Graphical Statistics* **1**, 213–229.
- Frank, A. and Asuncion, A. (2010), 'UCI machine learning repository', <http://archive.ics.uci.edu/ml>, Date of access: 10 April 2011.
- Frean, M. (1990), 'The upstart algorithm: a method for constructing and training feedforward neural networks', *Neural Computation* **2**, 198–209.
- Friedman, J. H. and Stuetzle, W. (1981), 'Projection pursuit regression', *Journal of the American Statistical Association* **76**, 817–823.

- Geweke, J. and Meese, R. (1981), 'Estimating regression models of finite but unknown order', *International Economic Review* **22**, 55–70.
- Guyon, I. and Elisseeff, A. (2003), 'An introduction to variable and feature selection', *Journal of Machine Learning Research* **3**, 1157–1182.
- Hagan, M. T., Demuth, H. B. and Beale, M. (1996), *Neural Network Design*, PWS Publishing Company, Boston, MA.
- Hannan, E. J. and Quinn, B. G. (1979), 'The determination of the order of an autoregression', *Journal of the Royal Statistical Society* **41**, 190–195.
- Hanson, S. J. (1990), Meiosis networks, in D. Touretzky, ed., 'Advances in Neural Information Processing Systems (Denver, 1989) (2)', Morgan Kaufmann, San Mateo, pp. 533–541.
- Harrison, D. and Rubinfeld, D. L. (1978), 'Hedonic housing prices and the demand for clean air', *Journal of Environmental Economics and Management* **5**, 81–102.
- Hastie, T. J. and Tibshirani, R. J. (1986), 'Generalized additive models', *Statistical Science* **1**, 297–318.
- Hastie, T. J. and Tibshirani, R. J. (1987), 'Generalized additive models: some applications', *Journal of the American Statistical Association* **82**, 371–386.
- Hastie, T. J. and Tibshirani, R. J. (1990), *Generalized Additive Models*, Vol. 43 of *Monographs on Statistics and Applied Probability*, Chapman and Hall, London.
- Haughton, D. (1989), 'Size of the error in the choice of a model to fit data from an exponential family', *Sankhyā, Series A* **51**, 45–58.
- Hebb, D. O. (1949), *The Organization of Behaviour*, Wiley, New York.
- Hopfield, J. J. (1982), 'Neural networks and physical systems with emergent collective computational abilities', *Proceedings of the National Academy of Sciences* **79**, 2554–2558.
- Hurvich, C. M. and Tsai, C. (1989), 'Regression and time series model selection in small samples', *Biometrika* **76**, 297–307.
- Jiao, L. and Li, H. (2010), 'Qspr studies on the aqueous solubility of pcdd/fs by using artificial neural network combined with stepwise regression', *Chemometrics and Intelligent Laboratory Systems* **103**, 90–95.
- Kang, S. (1991), An Investigation of the Use of Feedforward Neural Networks for Forecasting, PhD thesis, Kent State University.
- Kullback, S. and Leibler, R. A. (1951), 'On information and sufficiency', *The Annals of Mathematical Statistics* **22**, 79–86.

- Larsen, W. A. and McCleary, S. J. (1972), 'The use of partial residual plots in regression analysis', *Technometrics* **14**, 781–790.
- Lippmann, R. P. (1987), 'An introduction to computing with neural nets', *IEEE ASSP Magazine* pp. 4–22.
- Luger, G. F. (2005), *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, 5th edn, Addison-Wesley, London.
- Mallows, C. L. (1973), 'Some comments on cp', *Technometrics* **15**, 661–675.
- Marchand, M., Golea, M. and Ruján, P. (1990), 'A convergence theorem for sequential learning in two-layer perceptrons', *Europhysics Letters* **11**, 487–492.
- McCullagh, P. and Nelder, J. A. (1989), *Generalized Linear Models*, Vol. 37 of *Monographs on Statistics and Applied Probability*, 2nd edn, Chapman and Hall, London.
- McCulloch, W. S. and Pitts, W. H. (1943), 'A logical calculus of the ideas immanent in nervous activity', *Bulletin of Mathematical Biophysics* **5**, 115–133.
- Mézard, M. and Nadal, J. P. (1989), 'Learning in feedforward layered networks: the tiling algorithm', *Journal of Physics A*, 2191–2203.
- Minsky, M. and Papert, S. (1969), *Perceptrons*, MIT Press, Cambridge, MA.
- Murtagh, F. (1991), 'Multilayer perceptrons for classification and regression', *Neurocomputing* **2**, 183–197.
- Negnevitsky, M. (2005), *Artificial Intelligence - A Guide to Intelligent Systems*, 2nd edn, Pearson Education Inc., Essex, England.
- Potts, W. J. E. (1999), Generalized additive neural networks, in 'Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', pp. 194–200.
- Potts, W. J. E. (2000), *Neural Network Modeling Course Notes*, SAS Institute Inc., Cary, NC.
- Reed, R. D. and Marks II, R. J. (1999), *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*, MIT Press, Cambridge, MA.
- Rich, E. and Knight, K. (1991), *Artificial Intelligence*, 2nd edn, McGraw-Hill, Inc., New York.
- Ripley, B. D. (1996), *Pattern Recognition and Neural Networks*, Cambridge University Press, Cambridge, United Kingdom.
- Rissanen, J. (1978), 'Modelling by shortest data description', *Automatica* **14**, 465–471.
- Rosenblatt, F. (1958), 'The perceptron: a probabilistic model for information storage and organization in the brain', *Psychological Review* **65**, 386–408.
- Rosenblatt, F. (1962), *Principles of Neurodynamics*, Spartan Books, New York.

- Rumelhart, D. E. and McClelland, J. L. (1986), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, MIT Press, Cambridge, MA.
- Russell, S. and Norvig, P. (2010), *Artificial Intelligence - A Modern Approach*, 3rd edn, Pearson Education Inc., Upper Saddle River, New Jersey.
- SAS Institute Inc. (2005), 'Fact sheet', <http://www.sas.com/technologies/analytics/datamining/miner/factsheet.pdf>, Date of access: 06 January 2011.
- Schwarz, G. (1978), 'Estimating the dimension of a model', *The Annals of Statistics* **6**, 461–464.
- Setiono, R. (2001), 'Feedforward neural network construction using cross-validation', *Neural Computation* **13**, 2865–2877.
- Shibata, R. (1980), 'Asymptotically efficient selection of the order of the model for estimating parameters of a linear process', *The Annals of Statistics* **8**, 147–164.
- Socket, E. B., Daneman, D., Clarson, C. and Ehrich, R. M. (1987), 'Factors affecting and patterns of residual insulin secretion during the first year of type 1 (insulin dependent) diabetes mellitus in children', *Diabetologia* **30**, 453–459.
- Sugiura, N. (1978), 'Further analysis of the data by akaike's information criterion and the finite corrections', *Communications in Statistics - Theory and Methods* **7**, 13–26.
- Tang, Z. and Fishwick, P. A. (1993), 'Feedforward neural nets as models for time series forecasting', *ORSA Journal on Computing* **5**, 374–385.
- Winston, P. H. (1992), *Artificial Intelligence*, 3rd edn, Addison-Wesley, Massachusetts.
- Witten, I. H. and Frank, E. (2005), *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd edn, Morgan Kaufmann Publishers, San Francisco.
- Wong, F. S. (1991), 'Time series forecasting using backpropagation neural networks', *Neurocomputing* **2**, 147–159.
- Wood, S. N. (2006), *Generalized Additive Models: An Introduction with R*, Texts in Statistical Science, Chapman & Hall/CRC, London.
- Wynne-Jones, M. (1992), Node splitting: A constructive algorithm for feed-forward neural networks, in J. Moody, S. Hanson and R. Lippmann, eds, 'Advances in Neural Information Processing Systems (4)', Morgan Kaufmann, San Mateo, pp. 1072–1079.
- Xiang, D. (2001), Fitting generalized additive models with the gam procedure, in 'SUGI26 Conference Proceedings', SAS Institute Inc., Cary, NC.
- Zhang, G., Patuwo, B. E. and Hu, M. Y. (1998), 'Forecasting with artificial neural networks: the state of the art', *International Journal of Forecasting* **14**, 35–62.