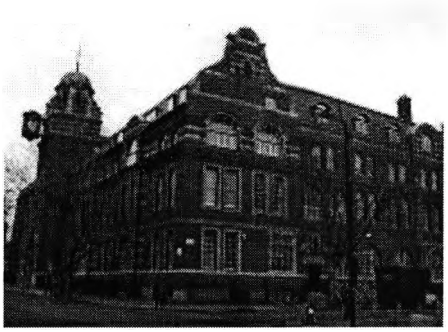
A Neural Network Linking Process

by

Harald Braun

City University, London



A Thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy,
School of Engineering and Mathematical Sciences,
City University, London

Abstract

A novel method of integrating multiple neural networks into one large network via a process referred to as *a neural network linking process* is proposed.

Neural networks are commonly trained to solve a specific problem for an encapsulated problem domain. A single network can undertake simple classification or generalisation problems. Dividing them into sub-problems, which in turn are solved by a sub-network, can disentangle more complicated classification or generalisation problems. A controller generally combines sub-network results. A controller can be, for instance, a gating network, voting system or a mathematical combiner. In each case, every sub-network is used as a separate unit and is not inter-connected to any other sub-network.

However, with the linking process a novel method for linking trained sub-networks into one large network by maintaining the knowledge of each individual sub-network is introduced. Furthermore, the linked network will utilize a stimulus process in order to distinguish the type of sub-problem to be solved, by largely retaining the accuracy of the sub-network, as well as being one step closer to the biological reality.

Original Contributions

The original contributions of this thesis may be summarised as follow:

- The creation of a framework for combination of hidden neurons by means of linking weight vectors, referred to as the *linking process*.
- Pruning of trained neural networks using the linking process.
- Introduction of domain membership that is held within neurons so that the weight vector length of each hidden neuron can be controlled.
- Study on where domain membership information can be induced into neurons to control their individual outputs and subsequently their contribution to the output of the overall network.
- Development of a stimuli network that generates input vector domain membership information for control of hidden neurons.
- Creation of penalty function to reduce weight updates for the backpropagation training algorithm to prevent the saturation of the summed input of hidden neurons.
- Linking of entire neural network weight matrixes for sharing of information for the purpose of improving the generalisation.
- Application of neural networks to claims reservation for general insurance companies and subsequent linking for improving forecasting capability.

List of Publications

1. H Braun, LL Lai "A Neural Network Linking Process", submitted to IEEE Transactions on Neural Networks, 2003.
2. H Braun, LL Lai "Neural Network Linking for Insurance Claims Reservation", submitted to IEEE Transactions on Neural Networks, 2003.
3. E Georges, LL Lai, H Braun, FN Che "Implementation of neural networks with VLSI", 6th European Conference on Power Electronics and Applications, pp 2.261-2.265, 19-21 September 1995.
4. LL Lai, FN Che, H Braun "Applications of neural networks to predicting harmonics", 6th European Conference on Power Electronics and Applications, pp 3.533-3.538, 19-21 September 1995.
5. H Braun, "Application of Neural Networks for Weather Forecasting", Proceedings of the British Council Workshop on Intelligent Weather Forecasting, Fudan University, Shanghai, China, pp 42-54, 21 March 2003, Invited Paper.
6. H Braun, LL Lai, "Application of the Internet to Power System Monitoring and Trading", chapter 12 in "Power Systems Restructuring and Deregulation", John Wiley & Sons Ltd, September 2001.

Table of Contents

List of Figures

List of Tables

Acknowledgements

The author wishes to thank his wonderful supervisor Dr L. L. Lai for his help and guidance throughout the course of this work. His comments and advice have been invaluable. Thanks to all the people in the Energy Systems Group who have advised me throughout the years.

Thanks also to Mr. Brian Martin from A. M. Best International for the permission to use the Insight Non-Life product and the extracted insurance data for publication in this thesis.

Special thanks to Dr. Fidelis Ndeh Che who was one of the first researchers within our group in the field of artificial intelligence and who taught me the English language. Furthermore to my lecturing colleague Dr. Mathias Belz for driving our efforts in teaching students in C programming that initiated the concept of neuron linking. Furthermore, to Jon Scott for correcting my spelling and grammar, my girlfriend Jeanetta McLean for her long standing support and finally to my mother Ursula Braun to whom this thesis is dedicated.

Declaration

The author hereby grants powers of discretion to the City University Librarian to allow this thesis to be copied in whole or in part without further reference to the author. This permission covers only single copies made for study purposes, subject to normal conditions of acknowledgement.

Chapter 1

Introduction

1.1 Introduction

The term *linking*, as used throughout this thesis, has originated from the C programming language. Creating a working program in C from source code, in its simplest form, employs two-step process. Step one is the *compilation* and pre-processing of the source code into a tokenised form. Step two is to *link* references tokens within the code to existing library functions into the compiled code to create the entire program [1, 2, 3].

Already existing code is the foundation of every software program. This is because the programmer can concentrate on the business logic that is to be implemented instead of writing code to access hard disks or graphics cards. With so many different computer hardware configurations, the most efficient method of writing software is by using the computers operating system device and function libraries because all appropriate drivers are present.

A direct comparison between the program creation of the C programming language and neural networks has been the basis of the concept of the *neural network linking process*. Because basic operations are contained in pre-compiled repositories or libraries, utilisation of such component libraries is archived by linking them together

with the program source code in need of the additional functionality, as shown in figure 1.1.

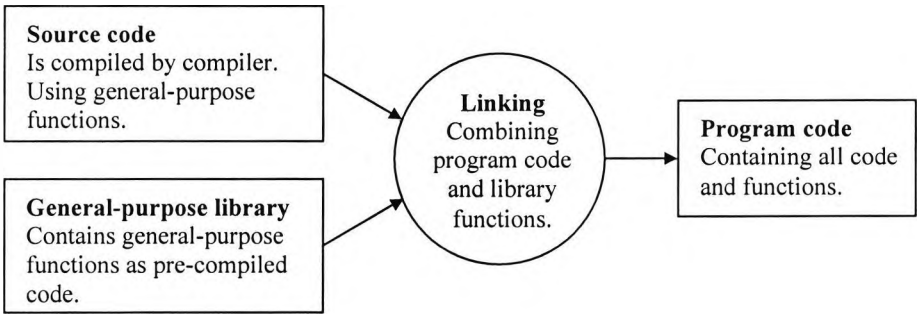


Figure 1.1 Linking source code and functions from a general-purpose library.

Most general-purpose functions contained in function libraries are used in programming source code to simplify the task for the developer and to speed up product development. For the reason that the functions contained in libraries can be called from different locations in the programming source code they are referred to as being reusable. With this, programming time and the size of the completed program can be considerably reduced. General-purpose libraries are a substantial part of every programming language. They can be shared, bought or sold on the open market, permitting developers to extend their programming environment easily.

In this thesis, the analogy of sharing and re-using libraries in programming languages has been successfully applied to the development of neural network structures. To date, neural networks are commonly trained to solve one specific problem with data from one problem domain. As a consequence, many researchers are continuously training new neural networks for data modelling, classification or interpolation issues. Once a neural network has been trained it is generally used in isolation without further integration of knowledge from other sources. Some attempts have been made to include symbolic knowledge into neural networks [4-9]. Every neural network is able to gain knowledge from different domains by learning from examples [10, 11]. But

they are generally trained to solve one particular problem for a single problem domain.

Complex problems which belong to more than one problem domain, can be simplified by using the divide-and-conquer principle in which a set of specialized sub-networks can be combined to form the final network [12-14].

The divide step of the divide and conquer principle can be done either by designing different networks for each problem domain or by purely probabilistic methods [15-17]. The conquer step can be a specific function of the outputs of one or more sub-networks or the output of the sub-network with the best performance [18, 19].

This thesis will introduce the utilisation of two different knowledge domains to train sub-neural networks which will be combined via a linking process into a single network. This follows the analogy of creating an application in a high level programming language, where the building blocks are firstly compiled (training of sub-networks) and then linked (network linking process).

The combination of neural network outputs to obtain a better result has been well researched and documented in the past [20-23]. The most common systems separate the training data by input space, train several networks for each problem domain and recombine the individual results. Such systems called Mixtures of Experts (ME), break one problem into sub-tasks, train a neural network and then combine the results with an output function or voting scheme [24-27].

The methodology that combines the outputs can be based on a linear or non-linear mathematical function (mathematical combiner) [28, 29] or a neural network itself (gating network) [30, 31]. Possible voting schemes between experts are Winner Takes All (WTA) or democratic systems [32-35]. The general concept of a ME system with gating network [36] is that a single expert is responsible for the output of a region within the input space. Whereby the decision of which network(s) to choose for that region lies with the implementation of the network controlling the gate.

Figure 1.2 shows the underlying principles of problem separation, expert training and expert recombination in one illustration. Problem separation can be used to divide the

training data into several domains. In order to find a suitable representation of the data for domain separation, linear transformation such as Principal Component Analysis (PCA) [37, 38], Factor Analysis (FA) [39, 40], Projection Pursuit (PP) [40] and Independent Component Analysis (ICA) [41] can be used. Most of these methods reduce the dimensionality so that clusters within the data can be found. Such clusters can then be used for the creation of separate knowledge domains and to train domain experts.

After separation of the training data into domains, individual neural network experts can be trained for optimal performance within their domain. Depending on the distribution of data within the domains input space and the location of the desired generalisation in hyperspace, the generalisation can be divided into interpolation or extrapolation [42].

Expert recombination amalgamates the outputs of the individual domain experts into one output. The most recent research topic that involves expert recombination currently is Hierarchical Mixture of Experts (HME) [43-49].

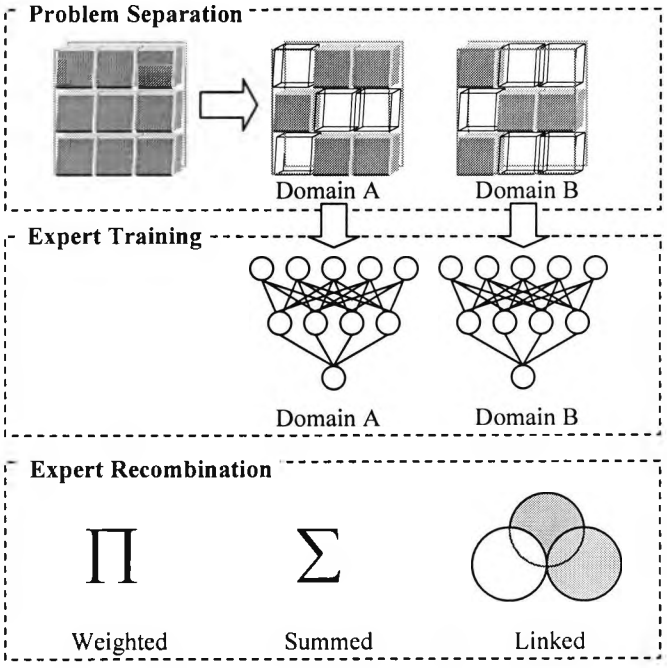


Figure 1.2 Linking can be categorised as an expert recombination method.

1.2 Problem Separation

Breaking up the input space into piecewise solutions will reduce the complexity of their interpretation [50-52]. In particular, the role of individual parameters and individual sub models (domain experts) is easily discerned. This is not the case for global models, where it is more difficult to ascertain the role of domain experts [53]. The advantage of domain experts is that they can be discarded or re-trained if their performance is substandard. Other advantages are rapid incremental learning of domain experts, fast cross-validation and no major local minima issues [54].

1.2.1 Self Organising Maps

There are several possible methods of partitioning the input space. For example, SOM's have the ability to perform an optimal partitioning of the input space. Their hierarchical decomposition of the input space yielded good results in many applications [55]. SOM's are beneficial for input space partitioning because of several reasons. They are less prone to local minima during the optimisation of the cluster centres, clustering results in Voronio tessellation (tiling of space without major gaps), the probability density of the inputs is preserved and the map is aligned to the largest principle components in the data set, i.e. it performs PCA.

1.2.1 Multi-Layer Perceptron

Other neural networks such as the standard Multi-Layer Perceptron (MLP) network or the Radial Basis Function (RBF) neural network, too, have the ability of partitioning multidimensional data [56-59]. The basic functioning of a MLP neural network is that the first hidden layer uses hyperplanes to partition the input space into a number of sections by way of dividing space with decision boundaries. The shape or the decision boundaries is depending on the neuron activation function e.g. linear, non-linear or radial. Decision boundaries are based on the fact that boundaries are located at the

point in input space where a small change of an input value causes a large change in output value [60-62].

1.2.3 Classic Methods

Whatever the partitioning method, it is imperative to consider the scalability of the proposed method. Some classic methods for partitioning, for instance, Delaunay tessellation [63] is known to optimally partition the input space, yet costs increase almost quadratic if the number of samples increases for high-dimensional functions [55]. This means that the method is prohibitive for problems where many patterns are expected. For this reason, many researchers use neural network based methods for input space partitioning especially SOM's [55]. Additionally, there are other more traditional methods for input space separations such as Fourier transform and polynomial approximation.

1.2.4 Avoiding Need for Problem Separation

One of the most uncomplicated way of partitioning the training data input space into domains is to prevent mixing domain information in the first place. If data originated from different sources that have nothing or very little in common, it can be assumed that the extracted data originated from different domains [64, 65]. Therefore if the data is collected independently or can be separated accurately, the need for complex input space separation can be avoided.

1.3 Expert Training

Using multiple experts requires that the problem domain can be separated into different partitions and that each partition can be solved by a domain expert that can be either rule based, a neural network, a fuzzy system, a genetic algorithm or others.

Because this thesis is in relation to linking of neural network weight matrixes, only neural networks will be used as domain experts.

The quality of expert training and the resulting approximation of the output generated depend for the most part on the partitioning of the input space and on the complexity of the objective function [66]. If the partitioning algorithm awkwardly chose boundaries, continuous partitioning of the input space can further reduce the complexity of an objective function to simplify expert training.

Some disadvantages of training multiple experts are poor generalisation performance since other domains are not present in training data of a single expert and increased memory and disk space since every expert needs to be treated in separation. But the linking process eliminates some of the disadvantages caused by multiple experts such as inadequate generalisation and increased network size.

1.3.1 Rule Based Systems

Rule based systems are algorithms in form of a computer program that emulates a human expert in a well-bounded domain of knowledge [67-69]. Characteristically, a rule based expert system consists of three major components as shown in figure 1.3. The first one being the dialog structure, second one the inference engine and the last one the knowledge base [70]. The dialog structure is the interface between the user and the system. User interfaces are designed to verbally explain their reasoning, much like a human expert would. The inference engine "drives" the computer to perform search strategies that arrive at various conclusions. The knowledge base is the set of facts and rules (heuristics) about the specific task at hand. Fore more information on rule based systems, please refer to the following publications and textbooks [71, 72].

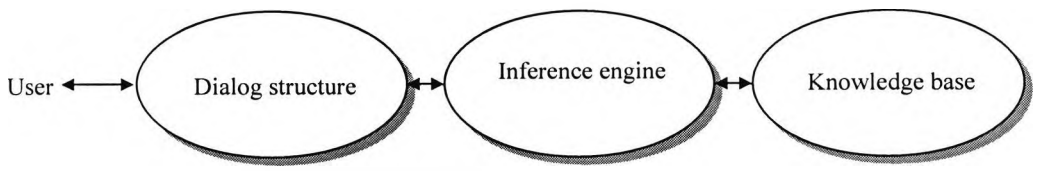


Figure 1.3 Three major components of a rule based system.

1.3.2 Neural Networks

The inspiration of neural networks originated form the biological nervous system [73-78]. A neural network consist of a collection of interconnected processing units referred to as neurons that have the ability of performing many computational tasks such as classification, generalisation and optimisation [79]. One important feature is the capability to adjust the internal representation of the model to the data that is used for training. A process referred to as learning or training accomplishes the flexibility of a neural network. A neural network consists of two major building blocks, the neurons and their connection weights.

Neurons, as used commonly, can have multiple inputs but only one output. The responsibility of the neuron is to sum up all incoming connections and process the sum by its activation function. This thesis has used sigmoidal-based activation functions such as the symmetrical and non-symmetrical sigmoid [80].

Weights determine the strength of connection between neurons in the network. This quantity is analogous to the biological synapse that determines the amplitude and sign of the signal affecting each neuron. The weights used in this thesis were continuous numbers with accuracy of 6 decimal places.

1.3.3 Fuzzy Systems

Fuzzy systems are related to rule based systems introduced in 1.3.1 and distinguish themselves mainly by the inference engines. Fuzzy systems are used where a system is difficult to model exactly i.e. limited or no data is available as it might be the case in a manually operated process.

Such processes are in general controlled by a human operator or expert because ambiguity or vagueness is common and key decisions must be made. A typical fuzzy system consists of fuzzyfication and defuzzyfication interfaces, a rule base and an inference engine as shown in figure 1.4.

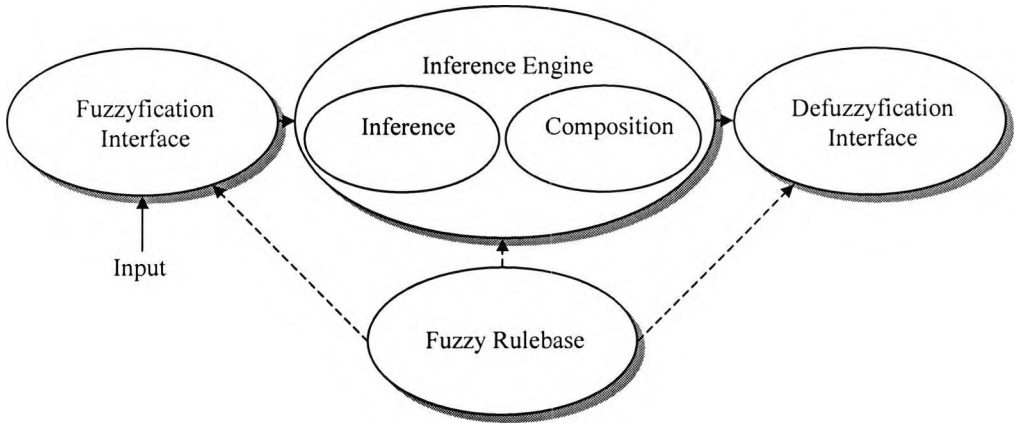


Figure 1.4 Major components of a typical fuzzy system.

The fuzzyfication interface the membership functions defined for the input variables are applied to determine the degree of membership for each premise. During the inference process, each premises truth-value is computed that is used for the conclusion part of each rule.

There are basically two inferences, Min or Product. If Min inference is used, the output membership function is clipped at a height that corresponds to the truth-value. If Product inference is used, the truth-value is used to scale down the output membership function.

During composition all domain memberships are combined to form a single truth-value for the entire fuzzy subset. As with the inference stage, there are two possible methods of composition, Max or Sum. If Max composition is used, the output fuzzy subset is constructed by using the largest output of all fuzzy subsets. If Sum composition is used, all fuzzy subset outputs are summed up to form a single output.

The final stage is the defuzzyfication stage. Its purpose is convert the output of the fuzzy subsets into a numeric value. The most common defuzzyfication functions are Centre of Gravity (COG) or Mean of Maxima (MOM). If COG is used, the centre of gravity of a membership function is used and if MOM is used the mean value of all points where the membership function has its highest value.

For more information on fuzzy systems, please refer to the following publications and textbooks [81-84].

1.3.4 Genetic Algorithms

Genetic Algorithms (GAs) are adaptive multi-dimensional search techniques, which were introduced by Holland [85-87] and are founded on the genetic processes of biological organisms based on Darwinian evolution theory. They have been applied successfully to solve many kinds of problems on search, optimisation, and machine learning.

Genetic algorithms and their extension to genetic programming make liberal use of vocabulary and concepts borrowed from the study of genetics. In particular, the existence of a genetic code, and the variation resulting from recombination and mutation during the reproductive process are fundamental to these methodologies. It is important to note however, that an extremely simplified version of human genetics is employed as the basis for genetic algorithms.

A GA operates on a problem that is specified in terms of a number of parameters that need to be optimised. One of the main features of GAs is that they hold a population of such parameters, so that many points in the problem space are sampled simultaneously. The population is generated by some heuristic and referred to as a vector or more traditionally as a string. Such strings contain coded data that are usually bit strings representing numbers or binary information with regards to a possible solution.

The search for the best strings starts by rating each strings solution against some form of test performance. A new population is then generated, by choosing the best strings preferentially. The simplest technique of doing this is to allocate children in proportion to their test performance. With this, the result is that the best string increases in number exponentially, and hence rapidly takes over the whole population. Other ways are to choose two members of the new population at random, or depending on their performance, and produce new offspring by mixing parameters

from the parents by means of crossover. As a result the strings ABCD and EFGH might be crossed to produce AFCD and EBGH.

There are numerous surveys of the GA field such as Goldberg [88] gives a survey of the field within a textbook on genetic algorithms, Davis [89-91] who contributed numerous research papers and on a more practical matter [92] a handbook of GA and Michalewicz [93] gives an overview of genetic algorithms and their applications.

1.3.5 Decision Trees

Decision trees are graphical representations of actions that categorise the decisions made. Decision trees have been successfully applied in fields where multiple decisions based on an analysis lead to an outcome e.g. medicine and law. In medicine, a decision tree can be used to analyse a patient's condition and suggest a course of treatment [94-97]. In law, a decision tree can be used to search existing cases by keywords to find similarities [98, 99].

A decision tree consists of a root node, branches and leaves. It represents a series of tests where each test is imposed at every step along the tree to form a complete analysis. Therefore, representing a functional map of domain knowledge starting from the root node, along its branches that are processed in order to reach a conclusion that is located in a leaf. Each path from the tree root to a leaf corresponds to a combination of tests and the tree itself corresponds to a collection of tests.

In more detail, classifications of instances are made by sorting them along the tree starting from the root node until a leaf is reached. Leaves provide classifications of the instance that is to be analysed. Every node within the tree denotes a test of some attribute and each branch descending from that node matches to one of the available values for this attribute. Each instance starts at the root node of the decision tree and is moved towards the tree branches whiles testing the attribute specified by each node, repeating this procedure until a leaf is reached. For a more complete and accurate treatment of the subject, the reader is encouraged to consult one of the many standard texts on decision trees [100-102].

1.4 Expert Recombination

The identification of the conditions under which the combination of an ensemble of experts yields improved performance compared to the individual expert is the underlying objective of expert recombination.

The creation of experts with de-correlated errors can be accomplished by a variety of methods. Such methods include the training of different types of experts, same experts with different topologies or structures [103-106], training of expert systems with different initial starting points. All these methods can be used to create distinct experts on the same data without the need for input space separation.

Other methods that can be used for the same type of experts are training on separate or partly overlapping data, experts with different training algorithms using different input signals [107] or dissimilar feature representations of the input [108].

It is well understood that combining several experts is often superior to using the expert that performed best after training. Generally, each expert can consist of any model as illustrated in the previous sections. In its simplest form, experts can be combined linearly as a mean average a weighted sum or voting schemes.

There are two distinct types of expert recombination schemes, static and dynamic. With the static scheme, experts are evaluated on their performance and connected into a scheme of fixed weighting [109]. With the dynamic scheme, the certainty of an expert in its output or its domain membership towards the input vector is determined and the contributions of each expert are evaluated during runtime [110].

The dynamic scheme, that calculates the contribution of each expert, requires some means of discriminatory quantity. Mostly the input vector that is applied to the experts is evaluated by some implementation of a gating network that is used to weight the output of the experts. This is arguably the preferred method because the weights connecting the experts are not static but instead depend on the input vector.

1.4.1 Mathematical Combiners

The process of expert recombination requires a means of combining the outputs of experts. Outputs of multiple experts can be combined by means of sum, product, weighted sum, or more complex functions that may or may not be dynamic.

One of the most common methods of recombination of experts is the combination of their individual outputs via the mean (mean-rule), where the outputs of the individual classifiers are averaged. Several studies have been undertaken and shown that mean-rule combination reduces the mean-squared error when independent experts produce independent errors [111, 112].

Developments in a common theoretical framework for combining classifiers which use distinct pattern representations outlined a number of possible combination schemes such as product, sum, min, max, and majority vote rules [113]. The results demonstrate that the sum rule outperformed the other classifier combination schemes. One explanation given was that the sensitivity of various schemes to estimation errors and that the sum rule is the most resilient to estimation errors, so almost certainly explaining its superior performance.

1.4.2 Gating Network

The purpose of a gating network is to predict the probability that the output of specific domain experts is accurate. It is part of the expert recombination process insofar that the gating network decides the amount of contribution from each expert to some means of mathematical combiner that will produce the overall network output.

Its decision to chose one or more experts from an ensemble of experts could be based on many factors. Gating networks are generally connected to the same inputs as the experts are but this is not imperative. Gating networks can receive a subset or entirely different inputs to the experts located in the ensemble.

Gating networks can consist of SOM, MLP, Hopfield or any other networks. They are usually trained to classify input vectors into domain memberships. Subsequently, they

can be using the same training data as the experts but do not require their targets since classification, not generalisation on the data is the objective. Alternatively, gating networks can be trained on separate data that contains expert membership information generated by i.e. feature extraction or PCA.

During the training of an ensemble with a gating network output and the output of experts, each network's output influences the whole ensemble in a different way. Whereas the experts contribute their outputs to the combiner, that forms the output of the entire system, the output of the gating network is connected to the combiner. The combiner then processes the outputs of the experts and the gating network to generate the output(s) of the entire ensemble. With this, the gating network controls the propagation of each expert's output to a greater or lesser extent.

During training the expert networks compete to let their output determine the system's output. Each expert network's weights are changed according to the back-propagation learning rule, which is applied in order to minimise the error of the whole system. The gating network's learning algorithm differentiates between winner and loser expert networks. It rewards the winner network by increasing its influence over the output of the whole system. This is achieved by increasing a factor for each winner that operates within the mathematical combiner. The gating network punishes the losers respectively by reducing the factor. The ready trained network is then able to rely on the specific task knowledge of the expert networks, as the gating network allows that network most suitable for a specific task to influence the output of the whole system to the greatest extent [114].

1.4.3 Democratic Systems

Another way of combining experts in an ensemble is the democratic approach of classifier voting. There is no controlling network, such as a gating network, in a democratic system. Each expert is analysing the incoming data and the output response and generate a means of best performance. This is followed by a voting system that will make a decision based on each expert's performance. There are

several voting schemes that have been researched [24, 25] but the two most popular ones are the "maximum votes" and the "majority" schemes. With the "maximum votes" scheme the system returns the classification suggested by the most experts [115]. With the "majority" scheme the system returns a classification if one is suggested by more than half of the modules [116].

1.4.4 Boosting and Bagging

Another method of creating multiple neural networks to solve a complex problem is the use of network ensemble methods. Ensemble methods differ from ME models in so far that all networks contribute towards the combined result. An ensemble consists of a set of individually trained networks whose predictions are combined when classifying unseen data for generalisation. Many researchers have proved that combination of individual network results in an ensemble improves generalisation. The two most popular techniques for creating networks that form part of an ensemble are Bagging and Boosting [106, 195, 196].

Bagging creates new training sets of the size of the original training set for each individual in the ensemble by randomly redistributing training records. As a result some training vectors may be repeated or omitted. This has the effect that each network is trained with different training data, therefore separating the input space. It is most effective on data with a high sensitivity where small changes in the inputs are related to large changes at the output or where similar input vectors have large differences in the output value [200].

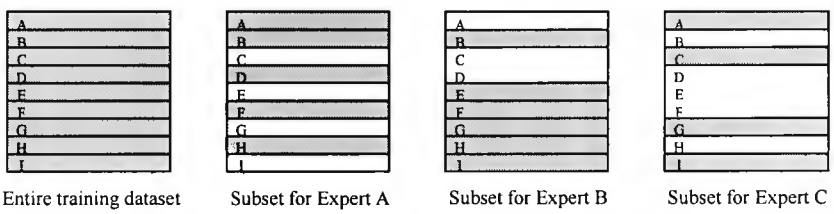


Figure 1.5 Bagging is a bootstrap method where every NN receives different training data.

Boosting is the process of creating a series of classifiers, which are using training data based on the performance of the previous classifier. Boosting has received its name because it is boosting data samples that have been predicted poorly by the predecessor of the current classifier by repeating the most erroneous data sample in the training set. Therefore boosting is creating new classifiers in each generation that are better for prediction than the current classifier.

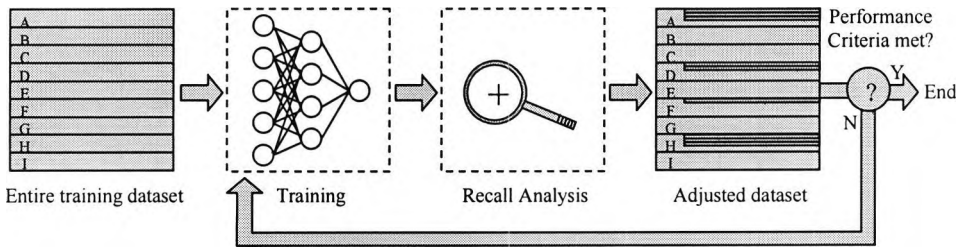


Figure 1.6 With boosting poorly performing records are duplicated to boost performance.

1.5 Neural Network Pruning

Real world problems can be solved with highly structured neural networks of large size. One of the issues arising with large network sizes is to minimise the size of the network whilst maintaining good performance. Neural networks that have been optimised in size are less likely to overfit and may thus generalise better on unseen data. In principle, there are two approaches to find the optimal size of a neural network, network growing and network pruning.

Network growing is a process where the network training has started with a relative small number of neurons and neurons are added during training to archive the required accuracy [201-203]. Since the linking process is a network reduction technique, network growing will not be discussed in great detail.

Network pruning is a process where weights or neurons are removed from a network that has started with a relative large number of neurons.

Pruning is based on evaluation of the performance of a trained network by means of determining weights or neurons that do not affect the accuracy to any large extent and removing those weights or neurons [120, 204]. The most widely used approach to find non-contributing neurons is by means of sensitivity analysis [60-62, 119]. In this method, a weight or all weights of a neuron are set to zero and the effect on the overall network performance is evaluated. Weights or neurons that have little effect can be identified and removed [138, 139, 205].

With the removal of non-contributing weights or neurons, pruning techniques can be split into three major categories, magnitude based, optimum brain damage and optimum brain surgeon.

1.5.1 Magnitude Based Pruning

The Magnitude Based pruning technique (MB) is based on the removal of weights that have the least affect on the training error and therefore a small saliency. The simplest approach of locating weights that can be removed is where saliencies are assumed to be corresponding to the magnitude (weight) value of the neuron connections. With this, weights that have small values are thought to have minor effect on the overall network output.

The basic approach is to start with an oversized neural network that contains too many weights for the objective function. After a reasonable amount of training, training is stopped (early stopping) and the weights are analysed. Weights are removed if their value lies outside the threshold. This removal process is referred to as weight elimination [211]. This is generally followed by a continuation of the network training. This process is then continued until no more weights below the threshold can be found or the training and generalisation error reach satisfactory readings.

Given that neural network training is permanently updating weights into positive and negative directions, it cannot be guarantee that once weight values are outside the threshold limits that they will fall back into the limits. Therefore a method of "weight decay" has been used that will reduce weights by a certain amount during each

training iteration [197-199]. The simplicity of the basic decay function, shown in equation (1.1) makes it a popular choice.

$$w^{new} = w^{old}(1-\varepsilon) \tag{1.1}$$

With ε being the decay parameter chosen between 0 and 1 [197]. In this way, weights that are not required for reducing the error of the objective function become smaller and smaller until they fall within the threshold limits so that they can be removed altogether.

Weights that are required for the training to match the objective function cannot be decayed indefinitely otherwise the decay may offset the weight update and the network may start to oscillate. Furthermore, there is a necessity for determining how important additional requirements are in relation to the error of fit. With early stopping, it is important to find the best stopping point [206, 207].

The value used for the decay parameter ε can be static e.g. 0.1 or dynamic. It is important for static weight decay parameters to be so small that they do not reduce the generalisation capabilities of the network significantly. They should also be large enough in order to prevent the training algorithm from being trapped in local minima. Dynamic weight decay values e.g. the quadratic weight decay function uses the sum of all vector lengths as a penalty term as shown in equation (1.2) [209].

$$\varepsilon(w) = \|w\|^2 = \sum_{i=N} w_i^2 \tag{1.2}$$

With N as referring to all weights within the neural network. This procedure operates during training by forcing some of the weights in the network to take values close to zero, while permitting other weights to retain their relatively large values. With this, all the weights in the network are treated equally.

Weights can be treated independently if an additional term is added to the update rule that takes the current weight into account. The simplest case uses a term that is

proportional to the present size of the weight that is subtracted from the standard backpropagation weight change. Although usually better than no penalty at all, the problem with weight decay for non-linear networks is that it makes it hard for the network to develop significant non linearities since these can only be achieved through large weights [210].

1.5.2 Optimum Brain Damage

The Optimum Brain Damage pruning algorithm (OBD) aims to iteratively delete weights whose deletion will result in the least increase of the recall error of the neural network for the purpose of reducing and optimising the architecture of a neural network [212].

The OBD algorithm is estimating the importance of the weights for the recall error (training error) and ranks the weights accordingly to their saliency. Saliencies can be estimated by a second order expansion of the training error around its minimum. For weight w_i, the saliency is given by equation (1.3).

$$s_i = \left(\frac{\varepsilon_i}{N_{train}} + \frac{1}{2} H_{ii} \right) \hat{w}_{ii}^2 \tag{1.3}$$

With H_{ii} as the i-th diagonal element of the Hessian matrix of the un-regularized cost function and ε as the weight decay associated with w_i. For reasons of simplification of the OBD algorithm, the following assumptions have been made:

- Terms of third and higher orders for the deleted weights can be neglected.
- The off-diagonal terms in the Hessian can be neglected (if more than one weight is pruned).

By repeatedly elimination of weights with the smallest saliencies (weight values not equal to zero) and retraining the resulting network, a size-optimised network is obtained. The iterative process can be stopped when the obtained estimate of the networks generalisation error is close to a required forecasting accuracy [213].

1.5.3 Optimal Brain Surgeon

The Optimum Brain Surgeon pruning algorithm (OBS) can be considered as the slightly more complex extension of the Optimum Brain Damage (OBD) algorithm. Even though OBS and OBD are essentially founded on the same theoretical approach, OBS does not make any assumption about the form of Hessian matrix as the OBD does as described in section 1.4.5.2. This causes the OBS to be more complex but on the other hand more robust than OBD. Many researchers have found that optimum brain surgeon (OBS) is superior to magnitude based (MB) and optimal brain damage (OBD) techniques. Furthermore, OBS permits the pruning of more weights than other methods for the same error on the training set. With this, it has produced better generalisation results on test data. The disadvantage with OBS is that the inverse of the Hessian matrix has to be computed fully to judge saliency and weight change for every link. The computation of the full inverse Hessian matrix makes the OBS a complicated algorithm that is quite slow and takes much memory compared to the other methods. OBS is only mentioned for reasons of completeness and more detailed information can be found in [213-216].

1.6 Structure of the Thesis

This thesis is organised into 6 chapters with contain sections that relate to the subject of their corresponding chapters. The next paragraph denotes an executive summary of this thesis followed by more comprehensive summaries for each chapter.

Chapter 1 represents a short survey of topics that represent the background required to other chapters. *Chapter 2* shows how the equations used for linking have been derived, this chapter contains the mathematical foundation of the linking process. *Chapter 3* presents a simple application of linking to pruning. *Chapter 4* introduces the stimuli network that is used as a classifier for the linked network. *Chapter 5*

introduces the linking of two networks for inter and extrapolation. *Chapter 6* uses the linking process for a real-life data example relating to the insurance industry.
Furthermore, chapter 3 introduces a standard reporting linking results in table format. Once the table layout and contents have been understood, all other linking results can be comprehended easily.

1.6.1 Summary of Chapter 2: Derivation of the linking equation

This chapter introduces vectors and their notation in multidimensional space. Since neurons are written as weight vectors, they contain the knowledge of the neuron. Linking is a process at the heart of neurons because of that, chapter 2 describes the process of combining two neuron's knowledge into one neuron. Neuron linking manipulates the network topology since neurons are removed with the objective to improve the generalisation capability and reduce the size of a linked network. Linking can be used for knowledge optimisation (network pruning) [117, 118] or knowledge combination (mixture of experts) [43-49].
Chapter 2 shows that weight vectors pointing in similar directions express similar knowledge. It shows how a 2:2:1 network is pruned into a 2:1:1 network by linking using 4 different ways of combining the knowledge of both neurons. The first linking method uses weight averaging, where weights of two neurons are summed and divided to create the mean average. After defining two methods for measuring vector component errors, an equivalent rate of error is calculated. The second linking method adds a weighting factor into the weight combination process with the objective to derive a function that shifts the error towards smaller weights. The third linking method derives the weighting factors from vector components, where each vector dimension will be individually weighted. The fourth linking method will utilise all findings from the previous linking methods. It is additionally introducing a vector length manipulation to allow linking of two vectors that have substantial length differences but point in the same, or opposite, direction.

1.6.2 Summary of Chapter 3: Pruning of Neural Network Weight Matrixes

This chapter demonstrated the use of linking for pruning weight matrixes. It shows how neurons can be combined by linking instead of being removed by more traditional pruning algorithms [119-123]. A neural network has been trained with a mathematical function and the recall and generalisation errors have been logged for benchmarking so that the performance of the pruned network can be evaluated.

This chapter discusses the loss of recall accuracy that may occur if hidden neurons are linked that have a large angle difference. It visualises weight vectors in 3D graphics to illustrate the pruning of complete neurons. In this process, a vector length adjustment factor F is calculated for vectors that point in similar, or opposite, directions but differ in length.

After the theory of hidden neuron linking has been discussed with a simple 2:2:1 network, a numerical example with a more complex 2:20:1 network is introduced. Pruning by utilisation of the linking algorithm has reduced the network size from 20 hidden neurons to just 15.

This chapter introduces the standard tables that will be used for analysis of the linking process. Such tables are: the list of angles between vectors (table 3.4), results of vector combination (table 3.5), vector component change impact analysis (table 3.6), vector length change impact analysis (table 3.7) and benchmark comparison (table 3.8). Once the tables are understood it should be easy to follow all other analysis since the order and appearance of the tables is consistent throughout this thesis.

1.6.3 Summary of Chapter 4: The Stimuli Network

This chapter introduces the stimuli-response theory, which originated from the field of psychology but it remained unused in the field of neural networks. It describes in its simplest form how knowledge held in the brain is accessed if different stimuli information is applied [124].

Chapter 4 introduces the activation of neurons by induction of stimuli into a neural network. Because the linking process is tagging neurons depending on their domain membership(s), each neuron holds information to which domain(s) it belongs. Depending on the stimuli information, neurons are encouraged to contribute to the overall network output if they belong to the domain(s) activated by the stimuli network. This is achieved by multiplication of each domain membership factor generated by the stimuli network with weight vector adjustment factors held in each linked neuron.

Two stimuli induction points in the neuron have been analysed given that the stimuli induction can be made prior or after the activation function. For that reason, an analysis of neuron sensitivity has been included. It has been found that neurons that have large numbers at their summed inputs prior to the activation function suffer from saturation.

To avoid neuron saturation, a modified backpropagation training update algorithm has been developed which uses a penalty function to reduce the weight updates of neurons that are close to saturation. With reduced weight updates on saturated neurons, weight updates of neurons that are not saturated change as usual to create a more uniform weight distribution throughout the entire weight matrix during training.

At the end of this chapter, linking of numerical experiments has been carried out with combinations of different stimuli induction points, saturated and unsaturated networks. Linking equations are calculated in detail including all explicit calculations and sub-totals to permit easy understanding. All linking results are subsequently compared with results obtained by utilisation of a gating network and a linear combiner. The chapter concludes with a direct comparison of recall error between all networks and ensemble methods discussed.

1.6.4 Summary of Chapter 5: Linking of Neural Network Weight Matrixes

This chapter introduces linking for the purpose of combining domain knowledge from separate neural network weight matrixes. It extends the framework of linking single neuron to the linking of entire weight matrixes for the formation of a single entity. Such entities will consist of linked and non-linked neurons that have the ability to share common information between domains.

For controlling the neurons, a stimuli network has been deployed in order to categorise the domain membership of input vectors. Each neuron contains an internal table of domain memberships that determine to which domains it belongs. Once a domain classification of the input vector has been made, each neuron's output will be depending on a match between the stimuli classification and the neuron's internal classification table.

Furthermore, this chapter introduces the advantage network linking can make to a changing environment. Linking is beneficial for the reusability of already trained matrixes. This means that already trained networks can be linked with new or changing networks without the need to re-train networks that have already been successfully trained. With this, a collection of neural network weight matrixes can be stored, e.g. in a database and whenever a problem contained by a single domain needs solving, all networks that have been trained on sub-domains can be linked to form a new network that contains knowledge of the entire problem domain.

This chapter is training two networks with functions describing a path through 3-dimensional space. After training, the input space is analysed to ensure both networks are sharing some input space sections so that the linked network is dealing with intersecting domains. Following the training, every single step required for linking of entire weight matrixes is presented. After linking, an error analysis on the linked weights is shown with tables that present an impact analysis for each linked neuron.

At the end of chapter 5, a numerical experiment is included that studies the influence that linking can have on interpolation and extrapolation. For that reason, training and testing data has been created that uses different sections of the input space, which has been spit into quadrants. Each quadrant of the input space represents interpolation or extrapolation depending on its position. Several neural networks have been trained with different quadrants in the training and testing data to create networks that are interpolating or extrapolating.

1.6.5 Summary of Chapter 6: Claims Reservation

This chapter focuses on a real-life application of linking neural networks for claims reservation for the insurance industry. Claims reservation is a very important issue for non-life Property and Casualty (P&C) insurance companies [125]. Insurance companies need to generate financial reserves for many reasons including liability management. If claims become payable, financial reserves that were taken from the money earned by premiums, need to be used. But how much money is reserved for this purpose depends strongly on history data for a particular risk group.

So far, no attempts have been found in the literature where neural networks were used for insurance claims reservation. Therefore a detailed analysis of available insurance data has been undertaken for the purpose of training data preparation. The result of this analysis includes the presentation of claims figures in triangular form and the extraction of training and testing data.

Two numerical examples are present in chapter 6. The first one uses data from only one insurance company and the second example uses data from two insurance companies. In each example, the data has been split into two domains and neural networks have been trained for each domain, which were subsequently linked after training.

There are already many existing numerical methods of calculating claims reservation. The most common one, the basic Chain Ladder Method (CLM) is introduced and

discussed in detail. The results of the CLM will be used as a benchmark to compare the results obtained with the neural networks.

In the first example, training data from one company has been normalised and split into two domains, one domain contains information about development years (x-direction) and the other domain contains information about years of origin (y-direction). Following this, two networks have been trained for each of the domains and linked. During each stage, each neural network has been tested on its ability to forecast and all forecasting results have been summarised for comparison.

In the second example, training data from two companies has been normalised and split into two domains. Two training and testing data files, consisting of data from one company each, were used for training and testing. After training, the networks have been evaluated for their forecasting capabilities and linked. The objective of this numerical experiment was to attempt to transfer knowledge obtained by one company to the other company. For this purpose, both networks have been re-evaluated after linking and compared with their forecasting performance prior to linking.

Chapter 2

Derivation of the Neural Network Linking Equation

2.1 Introduction

Vectors are defined as being complete if the magnitude and the direction are given. A vector represents a quantity that has a direction as well as magnitude. A two-dimensional vector is shown in figure 2.1, which introduces the notation used in this thesis. The magnitude of a Vector is denoted graphically by its length and mathematically by its absolute value $|\mathbf{w}|$. The vector direction is represented by its angle θ between a point of reference and the vector itself. To fully describe a two-dimensional vector, only two numeric quantities ($|\mathbf{w}|$, θ) are required [126].

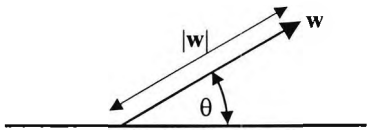
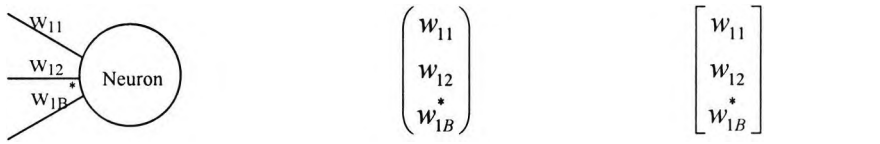


Figure 2.1 Vectors are defined by direction and magnitude.

Vector representation can be used for displaying many physical quantities in a graphical manner such as force or speed. Vectors can be used for graphical representation of neuron weights, as long as their dimensions do not exceed three. The most common coordinate system for representing vectors is the Cartesian coordinate system where every dimension is perpendicular to each other. This is also applicable to dimensions exceeding the all to familiar three-dimensional coordinate system. A coordinate system exceeding three dimensions is called Hyperspace because of their inherent difficulties of visualisation [127].

Weights associated with a neuron are a set of numbers that can be expressed as a vector or a row matrix in mathematical terms shown in figure 2.2.

*With w_{1B} as the neuron bias.



Neuron with three weights Vector representation Row matrix representation

Figure 2.2 Neurons can be written as three-dimensional vectors or a row matrix.

Vectors representing neuron weights are called *weight vectors*. They are in principle ordinary vectors that can be manipulated with all the pre-defined mathematical methodologies from the field of vector algebra. Weight vectors contain all the *knowledge* held by a neuron and are the basis of neural network knowledge representation [128, 129]. Manipulation of a neuron's weight vector will manipulate the knowledge of the neuron, therefore permitting knowledge manipulation based on the well-defined rules of vector algebra.

Linking of neurons is a process at the heart of neurons. It describes the process of combining two neuron's knowledge into one neuron, minimising the inaccuracy of the combined knowledge if compared to the original neurons. Neuron linking encompasses areas of neural network research where network topology manipulations are considered necessary to improve network generalisation for inter- or

extrapolation. Linking is a new knowledge combination approach, enriching the tools of neural network engineering for neural pruning and network fusion. It can be applied to the combination of entire neural networks or for the purpose of pruning a single network.

2.2 Linking of Neurons

Linking of neurons, which are representing similar knowledge into one single neuron, is a network topology manipulation algorithm. Like other topology manipulation algorithms, it will improve certain neural network behaviours at the price of loosing others. For example, behaviours such as generalisation can be improved but at the price of reduction of recall-accuracy [116, 130, 131].

Linking in this sense is effectively the combination of two multi-dimensional weight vectors from two neurons A and B into one weight vector of neuron R, as shown in figure 2.3. In other words, the removal of neurons will reduce the size of the hidden layer but may introduce changes in recall accuracy and generalisation.

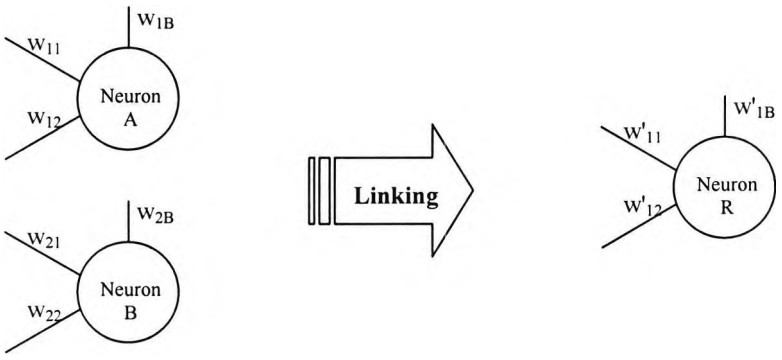


Figure 2.3 Linking of neurons A and B into a neuron R with changes in generalisation.

In the following sections different neuron linking algorithms will be introduced, starting with basic vector averaging and ending with weighted vector length and weight value averaging with correction factor. All linking algorithms introduced will try to link two weight vectors $\mathbf{v}_a$ and $\mathbf{v}_b$ to create a resulting vector $\mathbf{v}_r$. Vectors $\mathbf{v}_a$ and $\mathbf{v}_b$ are representing two hidden neurons and follow the standard weight indexing for the first two hidden neurons of a fully connected 2:2:1 neural networks as shown in figure 2.4. Equation (2.1) expresses the linking of vectors $\mathbf{v}_a$ of neuron A and $\mathbf{v}_b$ of neuron B into vector $\mathbf{v}_r$ of neuron R in mathematical vector notation.

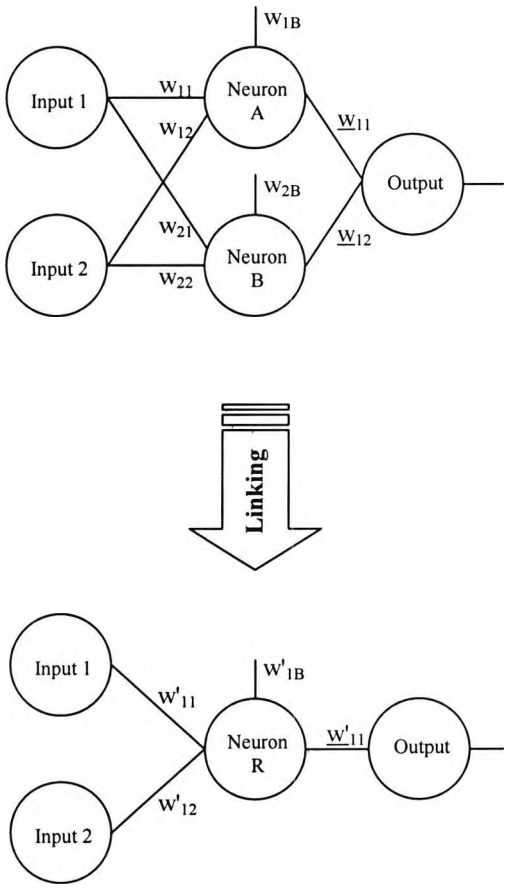


Figure 2.4 Transforming a 2:2:1 network into a 2:1:1 network by linking hidden neurons.

$$v_a = \begin{pmatrix} w_{11} \\ w_{12} \\ w_{1B} \end{pmatrix} \quad v_b = \begin{pmatrix} w_{21} \\ w_{22} \\ w_{2B} \end{pmatrix} \quad \Rightarrow \text{Linking} \quad v_r = \begin{pmatrix} w'_{11} \\ w'_{12} \\ w'_{1B} \end{pmatrix} \tag{2.1}$$

2.3 Linking Based on Averaging

Averaging is probably the most basic mechanism for combining two vectors. The resulting vector will simply be positioned between the two original vectors. Vectors are expressed as components in a Cartesian co-ordinate system or as row vectors. Combination of two vectors, which are placed in a two dimensional Cartesian co-ordinate system, into an averaged vector, involves calculating the mean value of the components for each dimension as shown in equation (2.2) and graphically in figure 2.5.

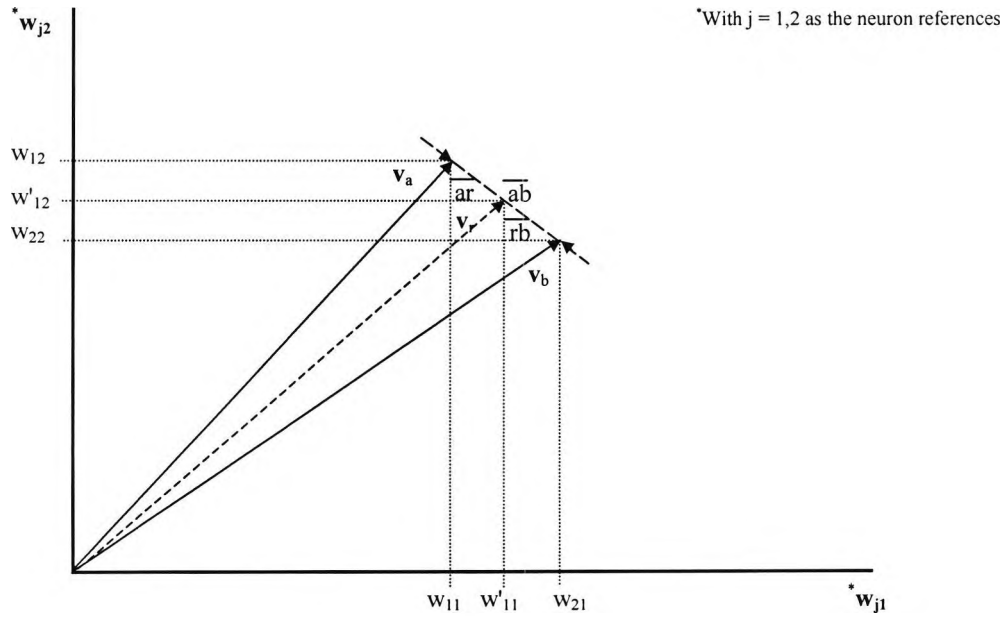


Figure 2.5 Creation of the resulting vector v_r via simple component averaging.

$$v_a = \begin{pmatrix} w_{11} \\ w_{12} \end{pmatrix} \qquad v_b = \begin{pmatrix} w_{21} \\ w_{22} \end{pmatrix} \qquad v_r = \begin{pmatrix} w'_{11} \\ w'_{12} \end{pmatrix} = \begin{pmatrix} \frac{w_{11} + w_{21}}{2} \\ \frac{w_{12} + w_{22}}{2} \end{pmatrix} \tag{2.2}$$

To be able to compare different ways of vector linking, a measure of error is required. A definition of error needs to be defined to suit the purpose to analyse the fitness of the resulting vector $\mathbf{v}_r$ with regards to the original vectors. The most apparent measure of error is the distance between the resulting vector $\mathbf{v}_r$ and each of the original vectors $\mathbf{v}_a$ and $\mathbf{v}_b$. It can be expressed as the ratio of the distance between an original vector and the resulting vector (e.g. $\overline{ar}$) divided by the total distance between the original vectors ($\overline{ab}$). Whenever two vectors are linked, as shown in figure 2.5, two errors err_{ar} and err_{rb} can be calculated as shown in equation (2.3)

$$err_{ar} = \frac{\overline{ar}}{\overline{ab}} \qquad err_{rb} = \frac{\overline{rb}}{\overline{ab}} \tag{2.3}$$

In an n-dimensional space, the distance $\overline{ab}$ between two vectors $\mathbf{v}_a$ and $\mathbf{v}_b$ can be determined via (2.4). The calculations of the distance $\overline{ar}$ between vector $\mathbf{v}_a$ and $\mathbf{v}_r$ and the distance $\overline{rb}$ between $\mathbf{v}_r$ and $\mathbf{v}_b$ are shown in (2.5) and (2.6) respectively.

$$\overline{ab} = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_n - b_n)^2} = \sqrt{\sum_{j=1}^{n} (a_j - b_j)^2} \tag{2.4}$$

$$\overline{ar} = \sqrt{(a_1 - r_1)^2 + (a_2 - r_2)^2 + \ldots + (a_n - r_n)^2} = \sqrt{\sum_{j=1}^{n} (a_j - r_j)^2} \tag{2.5}$$

$$\overline{rb} = \sqrt{(r_1 - b_1)^2 + (r_2 - b_2)^2 + \ldots + (r_n - b_n)^2} = \sqrt{\sum_{j=1}^{n} (r_j - b_j)^2} \tag{2.6}$$

Substitution of the components a_n, b_n and r_n for vectors $\mathbf{v}_a$, $\mathbf{v}_b$ and $\mathbf{v}_r$ from (2.2) into (2.4) to (2.6) and substitution of $\overline{ab}$, $\overline{ar}$ and $\overline{rb}$ from (2.4) to (2.7) into (2.3) leads to equations (2.7) and (2.8) for the errors of vector $\mathbf{v}_a$ (err_{ar}) and vector $\mathbf{v}_b$ (err_{rb}) with respect to $\mathbf{v}_r$.

$$err_{ar} = \frac{\sqrt{(w_{11} - w'_{11})^2 + (w_{12} - w'_{12})^2 + \ldots + (w_{1n} - w'_{1n})^2}}{\sqrt{(w_{11} - w_{21})^2 + (w_{12} - w_{22})^2 + \ldots + (w_{1n} - w_{2n})^2}} = \sqrt{\frac{\sum_{j=1}^{n} (w_{1j} - w'_{1j})^2}{\sum_{j=1}^{n} (w_{1j} - w_{2j})^2}} \tag{2.7}$$

$$err_{rb} = \frac{\sqrt{(w'_{11} - w_{21})^2 + (w'_{12} - w_{22})^2 + \ldots + (w'_{1n} - w_{2n})^2}}{\sqrt{(w_{11} - w_{21})^2 + (w_{12} - w_{22})^2 + \ldots + (w_{1n} - w_{2n})^2}} = \sqrt{\frac{\sum_{j=1}^{n} (w'_{1j} - w_{2j})^2}{\sum_{j=1}^{n} (w_{1j} - w_{2j})^2}} \tag{2.8}$$

In the case of simple vector component averaging, the resulting vector's components are determined as shown in (2.2) or more generally in (2.9). Using (2.9) to substitute w'_{1n} in (2.7) and (2.8), the errors err_{ar} and err_{rb} can be evaluated for the two dimensional example from figure 2.5 as shown in (2.10) to (2.12).

$$w'_{1n} = \frac{w_{1n} + w_{2n}}{2} \tag{2.9}$$

$$err_{ar} = \frac{\sqrt{\left(\frac{w_{11} + w_{21}}{2} - w_{11}\right)^2 + \left(\frac{w_{12} + w_{22}}{2} - w_{12}\right)^2}}{\sqrt{(w_{11} - w_{21})^2 + (w_{12} - w_{22})^2}} \tag{2.10}$$

$$err_{ar} = \frac{\sqrt{\left(\frac{w_{21} - w_{11}}{2}\right)^2 + \left(\frac{w_{22} - w_{12}}{2}\right)^2}}{\sqrt{(w_{11} - w_{21})^2 + (w_{12} - w_{22})^2}} \tag{2.11}$$

$$err_{ar} = \frac{\frac{1}{2}\sqrt{(w_{21} - w_{11})^2 + (w_{22} - w_{12})^2}}{\sqrt{(w_{11} - w_{21})^2 + (w_{12} - w_{22})^2}} = \frac{1}{2} \tag{2.12}$$

$$err_{ar} = \frac{\overline{ar}}{\overline{ab}} = \frac{1}{2} = 50\% \qquad err_{rb} = \frac{\overline{rb}}{\overline{ab}} = \frac{1}{2} = 50\% \tag{2.13}$$

Following the same simplifications for err_{rb} as for err_{ar} in (2.10) to (2.12) results in the equivalent rate of error of 50% between the resulting vector and the original vectors. Observation of equation (2.13) proves the equal distribution of the error between both original vectors, concluding that $\overline{ar} = \overline{rb}$ as expected from an un-weighted mean average in (2.9).

2.4 Linking Based on Weighted Average by Vector Length

Linking of neuron weight vectors into one resulting weight vector can be achieved by different methods of vector combination. In section 2.3, a simple averaging method was introduced, which distributed the distance $\overline{ab}$ between the two vectors $\mathbf{v}_a$ and $\mathbf{v}_b$ equally to 50% each. But simple weight component averaging does not take into account that stronger neuron weights have generally a higher contribution to the total neuron output [132, 133]. This is specifically true if two neurons point in similar directions but have substantially different vector lengths. The linking of two neurons where one neuron's vector length exceeds the other neuron's vector length by a noticeable magnitude, should take this unbalanced contribution towards the total neuron output into consideration.

To overcome the disadvantages of the simple averaging method, a *weighted average* can be used. The weighted average used for the linking of neurons is the *weighted arithmetic mean.* There are several other weighted average methods available, such as the weighted geometric mean, but for reasons of direct comparison to the simple mean average presented in section 2.3, only the weighted arithmetic mean is discussed.

In this method each neuron weight vector error err_{ar} and err_{rb} is *weighted* with the vector lengths $|\mathbf{v}_a|$ and $|\mathbf{v}_b|$ in such a way that the resulting vector $\mathbf{v}_r$ is closer to the longest vector. A reduced distance towards the resulting vector $\mathbf{v}_r$ represents a smaller error with respect to $\mathbf{v}_r$. An association between the vector lengths $|\mathbf{v}_a|$ and $|\mathbf{v}_b|$ and the errors err_{ar} and err_{rb} can be implemented to describe almost any functional relationship. But for reasons of simplicity a linear inversely proportional relationship between the vector lengths $|\mathbf{v}_a|$ and $|\mathbf{v}_b|$ and their errors err_{ar} and err_{rb} towards the resulting vector $\mathbf{v}_r$ has been applied as shown in (2.14) to (2.16).

$$|v_a| \sim \frac{1}{err_{ar}} \sim \overline{rb} \sim err_{rb} \tag{2.14}$$

$$|v_b| \sim \frac{1}{err_{rb}} \sim \overline{ar} \sim err_{ar} \tag{2.15}$$

$$\frac{|v_a|}{|v_b|} = \frac{err_{rb}}{err_{ar}} = \frac{\overline{\overline{ab}}}{\overline{\overline{ar}}} = \frac{\overline{rb}}{\overline{ar}} \tag{2.16}$$

Equation (2.16) can be interpreted as the "division of a straight line by a given ratio". With this interpretation, the straight line $\overline{ab}$ is to be divided into $\overline{ar}$ and $\overline{rb}$ by the ratio R_{ab} as shown in (2.17) and graphically represented in figure 2.6.

$$R_{ab} = \frac{|v_a|}{|v_b|} = \frac{\overline{rb}}{\overline{ar}} \tag{2.17}$$

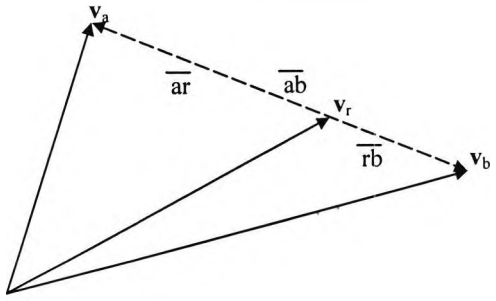


Figure 2.6 Division of a straight line $\overline{ab}$ by a given ratio R_{ab}.

The calculation of the position of the resulting vector $\mathbf{v}_r$ in terms of the ratio R_{ab} is dividing the line $\overline{ab}$ into $\overline{ar}$ and $\overline{rb}$. Using the distance ratio $\frac{\overline{ar}}{\overline{ab}} = \frac{\overline{ar}}{\overline{ar} + \overline{rb}}$ and with $\overline{rb}$ being proportional to $|\mathbf{v}_a|$ (2.14) and $\overline{ar}$ being proportional to $|\mathbf{v}_b|$ (2.15) equation (2.18) can be derived.

$$\frac{\overline{ar}}{\overline{ab}} = \frac{|v_b|}{|v_b| + |v_a|} \tag{2.18}$$

Resolved to $\overline{ar}$:

$$\overline{ar} = \left(\frac{|v_b|}{|v_b| + |v_a|} \right) \cdot \overline{ab} \tag{2.19}$$

The objective is to derive a function, which determines the position of vector $\mathbf{v}_r$ by the means of $\mathbf{v}_a$ and $\mathbf{v}_b$, taking into account the weighting constraint. Therefore, following the rules of vector algebra of free vectors[1], vector $\mathbf{v}_r$ can be defined by summing up the vectors $\mathbf{v}_a$, $\mathbf{v}_r$ and distance vector $\overline{ar}$ as shown in (2.20).

$$v_a + \overline{ar} - v_r = 0 \tag{2.20}$$

Resolved to $\mathbf{v}_r$:

$$v_r = v_a + \overline{ar} \tag{2.21}$$

Substituting (2.19) into (2.21):

$$v_r = v_a + \left(\frac{|v_b|}{|v_b| + |v_a|} \right) \cdot \overline{ab} \tag{2.22}$$

Equation (2.22) leaves $\overline{ab}$ as being the only unknown left to conclude. Following the same steps as in (2.20), equation (2.23) can be derived and used for substitution in (2.22) to obtain (2.24).

$$v_a + \overline{ab} - v_b = 0 \tag{2.23}$$

[1] Free vectors may be drawn anywhere in n-dimensional space as long its magnitude and direction are preserved.

$$v_r = v_a + \left(\frac{|v_b|}{|v_b| + |v_a|} \right) \cdot (v_b - v_a) \tag{2.24}$$

$$v_r = \frac{v_a(|v_b| + |v_a|)}{|v_b| + |v_a|} + \frac{(|v_b| v_b - |v_b| v_a)}{|v_b| + |v_a|} = \frac{|v_b| v_a + |v_a| v_a + |v_b| v_b - |v_b| v_a}{|v_b| + |v_a|}. \tag{2.25}$$

After simplification of (2.25), the resulting vector $\mathbf{v}_r$ can be determined as:

$$v_r = \frac{|v_a| v_a + |v_b| v_b}{|v_b| + |v_a|} \tag{2.26}$$

Equation (2.26) can be interpreted as the derivation of the resulting vector $\mathbf{v}_r$ by applying the *weighted arithmetic mean* to vectors $\mathbf{v}_a$ and $\mathbf{v}_b$ so that the error distances $\overline{ar}$ and $\overline{rb}$ between vector $\mathbf{v}_a$ and $\mathbf{v}_r$ and vector $\mathbf{v}_r$ and $\mathbf{v}_b$ are inversely proportional to their vector lengths $|\mathbf{v}_a|$ and $|\mathbf{v}_b|$.

To express this result in Cartesian coordinates as shown in section 2.3 equation (2.2) for the case of simple mean average, let $\mathbf{v}_a$ be $\begin{pmatrix} w_{11} \\ w_{12} \end{pmatrix}$ and $\mathbf{v}_b$ be $\begin{pmatrix} w_{21} \\ w_{22} \end{pmatrix}$ as graphically presented in figure 2.5 so that:

$$\mathbf{v}_r = \begin{pmatrix} w'_{11} \\ w'_{12} \end{pmatrix} = \begin{pmatrix} \frac{|\mathbf{v}_a| w_{11} + |\mathbf{v}_b| w_{21}}{|\mathbf{v}_a| + |\mathbf{v}_b|} \\ \frac{|\mathbf{v}_a| w_{12} + |\mathbf{v}_b| w_{22}}{|\mathbf{v}_a| + |\mathbf{v}_b|} \end{pmatrix} \tag{2.27}$$

Equation (2.27) shows how the components of the resulting vector $\mathbf{v}_r$ for a two-dimensional space are computed by substitution of the vector references in (2.26) with coordinate components. It shows that (2.26) is not restricted to any size of the dimensions and can be applied to any n-dimensional hyperspace. This is important since the number of input neurons, plus bias e.g. w_{1b} in (2.1), determine the dimensionality of the first hidden layer. The constraint of unlimited dimensionality

within (2.26) is given and therefore its usability for linking of n-dimensional hidden neurons is satisfied.

Following the derivation of (2.26) for weighted vector linking, the error err_{ar} and err_{rb} in relation to vectors $\mathbf{v}_a$ and $\mathbf{v}_b$ can be determined by applying (2.19) to (2.14) and (2.15), under the constraint of (2.28).

$$err_{ar} + err_{rb} = 1 \tag{2.28}$$

$$\frac{\overline{ar}}{\overline{ab}} = \left(\frac{|v_b|}{|v_b| + |v_a|} \right) = \frac{err_{ar}}{err_{ar} + err_{rb}} = \frac{err_{ar}}{1} = err_{ar} \tag{2.29}$$

$$\frac{\overline{rb}}{\overline{ab}} = \left(\frac{|v_a|}{|v_b| + |v_a|} \right) = \frac{err_{rb}}{err_{ar} + err_{rb}} = \frac{err_{rb}}{1} = err_{rb} \tag{2.30}$$

$$err_{ar} = \frac{|v_b|}{|v_a| + |v_b|} \qquad err_{rb} = \frac{|v_a|}{|v_a| + |v_b|} \tag{2.31}$$

To verify the correctness of equations (2.31), err_{rb} can be divided by err_{ar} in order to reconstruct equation (2.16) as shown in (2.32).

$$\frac{err_{rb}}{err_{ar}} = \frac{|v_a|}{|v_a| + |v_b|} \cdot \frac{|v_a| + |v_b|}{|v_b|} = \frac{|v_a|}{|v_b|} \tag{2.32}$$

2.5 Linking Based on Weighted Average by Vector Components

This section is introducing a less coarse method of weighted vector linking. It will not use the vector lengths as discussed in section 2.4. Instead it will utilise an error calculation for each dimension to gain a more precise weighting of the errors. In other words, instead of using only one weight ratio (the vector length ratio R_{ab}) for weighting the overall error of a vector, each vector dimension will be individually

weighted. This can be achieved by creating a dimension specific ratio R_n in order to weight individual errors for each dimension.

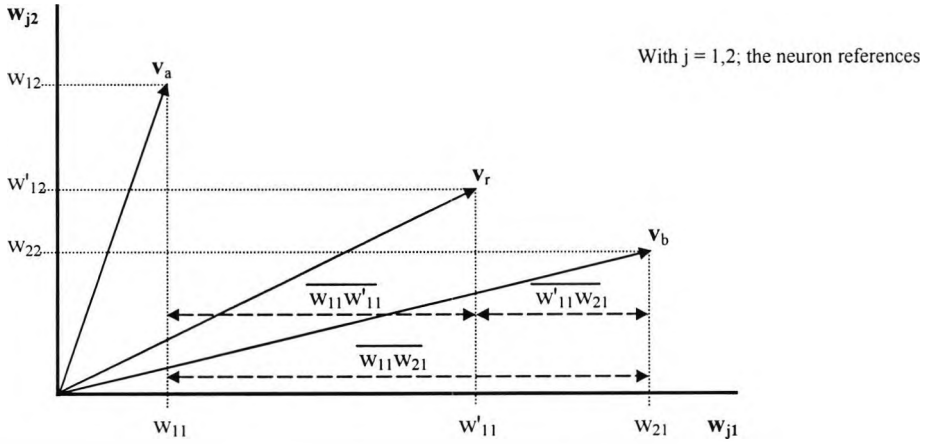


Figure 2.7 Weighting of individual dimensions by a dimension specific ratio R_n.

Calculation of the ratio R_n for each dimension is based on the individual component values j for every dimension n. Each component value is denoted by w_{jn}, with w_{11} being located in the first dimension of neuron 1 (vector $\mathbf{v}_a$) and w_{22} being located in the second dimension of neuron 2 (vector $\mathbf{v}_b$).

Following the results from section 2.4, equations (2.14) and (2.15), we can substitute the vector length $|\mathbf{v}_a|$ with w_{1n} and $|\mathbf{v}_b|$ with w_{2n} for each dimension n.

$$w_{1n} \sim \frac{1}{err_{1n}} \sim \overline{w'_{1n}w_{2n}} \sim err_{2n} \tag{2.33}$$

$$w_{2n} \sim \frac{1}{err_{2n}} \sim \overline{w_{1n}w'_{1n}} \sim err_{1n} \tag{2.34}$$

Equations (2.33) and (2.34) can be interpreted as the larger the value of w_{jn} (w_{1n} or w_{2n}), the lower its error err_{jn} (err_{1n} or err_{2n}) with respect to the distance of the resulting vector component w'_{1n} as graphically presented in figure 2.7.

Following the analogy from section 2.4, equivalent equations to (2.16) and (2.17) with respect to individual weighting ratios for each dimension R_n can be derived.

$$\frac{w_{1n}}{w_{2n}} = \frac{err_{2n}}{err_{1n}} = \frac{\overline{w'_{1n} w_{2n}}}{\overline{w_{1n} w'_{1n}}} \tag{2.35}$$

$$R_n = \frac{w_{1n}}{w_{2n}} = \frac{\overline{w'_{1n} w_{2n}}}{\overline{w_{1n} w'_{1n}}} \tag{2.36}$$

Because of the apparent similarities between (2.16) and (2.35) and between (2.17) and (2.36), the derivations of the equations for the resulting vector components will be equivalent to (2.18) to (2.26) with the following substitutions: w_{1n} for $|\mathbf{v}_a|$, w_{2n} for $|\mathbf{v}_b|$, w_{1n} for $\mathbf{v}_a$ and w_{2n} for $\mathbf{v}_b$. Therefore the derivation will not be repeated and only the result from (2.26) after substitution is shown in equation (2.37).

$$w'_{1n} = \frac{w_{1n} \cdot w_{1n} + w_{2n} \cdot w_{2n}}{w_{1n} + w_{2n}} = \frac{w_{1n}^{\,2} + w_{2n}^{\,2}}{w_{1n} + w_{2n}} \tag{2.37}$$

Equation (2.37) can be interpreted as the derivation of the resulting vector components $\mathbf{v}_r=\{w'_{11}, w'_{12}\ldots w'_{1n}\}$ by applying the weighted arithmetic mean to the original vector components of $\mathbf{v}_a$ and $\mathbf{v}_b$, weighted by the individual components for each dimension, with n as the number of dimensions.

To express this result in Cartesian coordinates as shown in section 2.3 equation (2.2) for the case of simple mean average, let $\mathbf{v}_a$ be $\begin{pmatrix} w_{11} \\ w_{12} \end{pmatrix}$ and $\mathbf{v}_b$ be $\begin{pmatrix} w_{21} \\ w_{22} \end{pmatrix}$ as graphically presented in figure 2.5 so that:

$$\mathbf{v}_r = \begin{pmatrix} w'_{11} \\ w'_{12} \end{pmatrix} = \begin{pmatrix} \dfrac{w_{11}^{\,2} + w_{21}^{\,2}}{w_{11} + w_{21}} \\ \dfrac{w_{12}^{\,2} + w_{22}^{\,2}}{w_{12} + w_{22}} \end{pmatrix} \text{ or more generally: } \mathbf{v}_r = \begin{pmatrix} w'_{11} \\ \vdots \\ w'_{1n} \end{pmatrix} = \begin{pmatrix} \dfrac{w_{11}^{\,2} + w_{21}^{\,2}}{w_{11} + w_{21}} \\ \vdots \\ \dfrac{w_{1n}^{\,2} + w_{2n}^{\,2}}{w_{1n} + w_{2n}} \end{pmatrix} \tag{2.38}$$

Extending the analogy to equations (2.28) to (2.30), equation (2.31) can be expressed with reference to the substitutions used to derive (2.37). Consequently, the errors for each individual dimension are:

$$err_{1n} = \frac{w_{1n}}{w_{1n} + w_{2n}} \qquad err_{2n} = \frac{w_{2n}}{w_{1n} + w_{2n}} \tag{2.39}$$

As for the example in figure 2.5, the two dimensional Cartesian coordinate system will find two error figures for each dimension, err_{11} and err_{21} for dimension 1 (x-axis) and err_{12} and err_{22} for dimension 2 (y-axis). Each error can be individually obtained by substituting n in (2.39) with 1 and 2 for each dimension respectively.

To determine the overall errors err_{ar} and err_{rb} of the resulting vector $\mathbf{v}_r$ with respect to the distances $\overline{ar}$ and $\overline{rb}$ between the original vectors $\mathbf{v}_a$ and $\mathbf{v}_b$ as shown in (2.5), equations (2.7) and (2.8) apply. Since the original vector components have participated individually for each dimension, the equations (2.7) and (2.8) cannot be expressed by the means of $|\mathbf{v}_a|$ or $|\mathbf{v}_b|$ as in (2.13) or (2.31). For that reason, (2.7) and (2.8) cannot be simplified and requires the errors to be calculated as presented.

2.6 Linking Based on Weighted Average by Vector Components and Length Manipulation

The combination of two vectors $\mathbf{v}_a$, $\mathbf{v}_b$ into one resulting vector $\mathbf{v}_r$ will always distribute the existing distance $\overline{ab}$ between the original vectors to some extent. Such distribution can be regulated by different error distribution methodologies, which can be derived in order to share the overall error under different conditions. These algorithms are only manipulating the error distribution but not the overall error. While the overall error is proportional to the distance between the endpoints $\overline{ab}$ of the original vectors, it will remain constant unless the original vectors are moved towards each other and the distance $\overline{ab}$ is reduced. But at this stage, the training of the

network may have already been completed and manipulation of a cluster of trained weight vectors with the objective to reduce the distance between two vectors would affect the network behaviour to some unknown extent. Therefore, a more precise method, which manipulates the resulting weight vectors length $|\mathbf{v}_r|$, will be introduced in this section.

The objective to reduce the error components err_{ar} and err_{rb} of the original vectors $\mathbf{v}_a$ and $\mathbf{v}_b$ without manipulating their original positions and the constraint for simplicity to keep the required computational complexity low, leads to the manipulation of the only free parameter, the resulting vector $\mathbf{v}_r$. With the findings of sections 2.3 to 2.5, the resulting vectors position and length can be adjusted to comply with the aims of each particular section to shift the error distribution closer to any one of the original vectors. If one of the resulting vector's parameters ($|\mathbf{v}_r|$, θ_r) is to be manipulated for the purpose of reducing the overall error, it would be the vector length $|\mathbf{v}_r|$ because the vector length can be changed through simple multiplication by a number (*scalar*) F.

Multiplying a vector by a scalar F changes the length of the vector by this factor so that if F = 4 a vector would be obtained with four times the length of the original vector. Multiplication with a negative number inverts the vector direction as well as changing its length, allowing even the linking of vectors of opposite direction.

This process is effectively a conversion of two vectors $\mathbf{v}_a$ and $\mathbf{v}_b$ into one resulting vector $\mathbf{v}_{r1}$ and a scalar F which can be used to multiply the resulting vector $\mathbf{v}_{r1}$ to change its length in order to get closer to any of the original vectors and subsequently reduce the overall error, see figure 2.8.

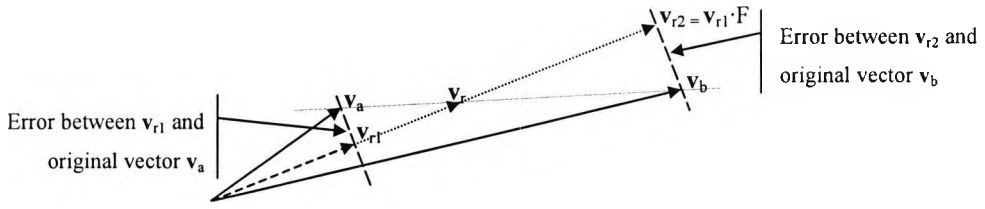


Figure 2.8 Reducing the errors between vectors by vector length multiplication.

Figure 2.9 presents a graphical solution to the calculation of the resulting vector $\mathbf{v}_r$ and the vector length-adjusting factor F. It will be used to aid the derivation of the mathematical equations required to find $\mathbf{v}_r$ and F.

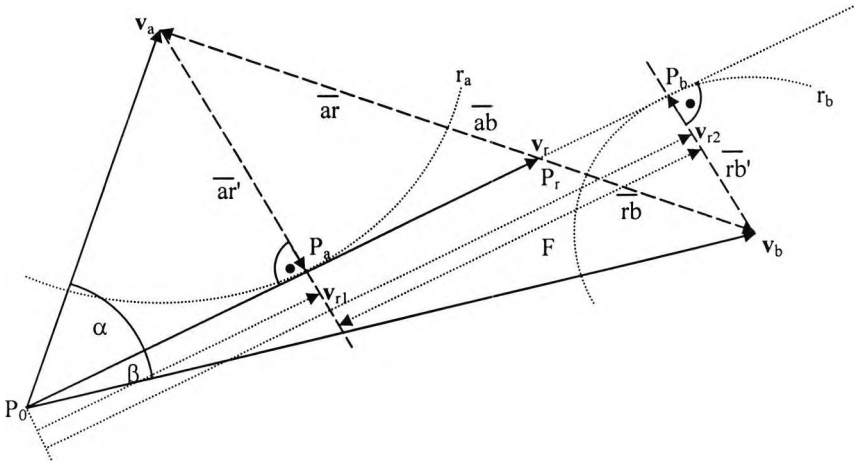


Figure 2.9 Graphical representation of distance reduction by v_r vector length adjustment.

The method of deriving the resulting vector $\mathbf{v}_r$ compared to previous sections in this chapter is similar to the method based on weighted average by vector components in section 2.5. In this section, one vector $\mathbf{v}_r$ has been created that has a fixed length and location in n-dimensional space. For the purpose of reducing the errors err_{ar}, err_{rb} between the original vectors and $\mathbf{v}_r$, the vector length $|\mathbf{v}_r|$ is changed so that $\mathbf{v}_r$ moves closer to one of the original vectors. In figure 2.9, $\mathbf{v}_r$ can be seen as a solid line starting from the point P_0 and ending at P_r. This vector has been calculated with equation (2.37) of section 2.5. Analysis of the error err_{ar} between $\mathbf{v}_a$ and $\mathbf{v}_r$ from section 2.3 has proven a direct relationship between the distance $\overline{ar}$ and the error err_{ar}. Hence, reduction of the error err_{ar} can be achieved by reducing the distance between vector $\mathbf{v}_a$ and $\mathbf{v}_r$. This can be achieved by reducing the vector length $|\mathbf{v}_r|$ so that the minimum distance $\overline{ar'}$, at point P_a between $\mathbf{v}_a$ and P_a, is met by adjusting the length of vector $\mathbf{v}_r$.

To formulate the algorithms required finding the optimal vector length of $\mathbf{v}_r$ towards vector $\mathbf{v}_a$ with mathematical means, a graphical solution can be found by producing an intersection P_a on $\mathbf{v}_r$ by a circle r_a with its centre point at $\mathbf{v}_a$. The distance between P_0 and P_a is the length of vector $\mathbf{v}_{r1}$, which is unknown. According to the law of Pythagoras, the distance $\overline{ar'}$ is part of a right-angled triangle where $\mathbf{v}_a$ is the Hypotenuse. Because of this, $\overline{ar'}$ will always be orthogonal to $\mathbf{v}_r$, permitting a trigonometric solution as shown in (2.40).

$$\overline{P_0 P_a} = |\boldsymbol{v}_{r1}| = \cos(\alpha) \cdot |\boldsymbol{v}_a| \tag{2.40}$$

To find the unknown angle α between vector $\mathbf{v}_a$ and $\mathbf{v}_r$ the dot product between two vectors can be used for substitution into (2.40), shown in equations (2.41) and (2.42).

$$\boldsymbol{a} \,.\, \boldsymbol{b} = |\boldsymbol{a}| \cdot |\boldsymbol{b}| \cdot \cos(\alpha) \tag{2.41}$$

$$\cos(\alpha) = \frac{\boldsymbol{a} \,.\, \boldsymbol{b}}{|\boldsymbol{a}| \cdot |\boldsymbol{b}|} \tag{2.42}$$

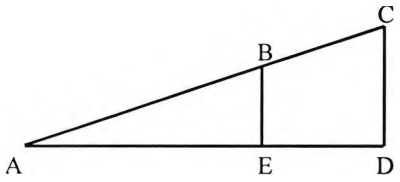
After substitution of $\cos(\alpha)$ and replacement of the letters a and b to the relevant vector references, equation (2.43) can be used to find the minimal distance between $\mathbf{v}_a$ and $\mathbf{v}_{r1}$.

$$|\boldsymbol{v}_{r1}| = \frac{\boldsymbol{v}_a \,.\, \boldsymbol{v}_r}{|\boldsymbol{v}_a| \cdot |\boldsymbol{v}_r|} \cdot |\boldsymbol{v}_a| = \frac{\boldsymbol{v}_a \,.\, \boldsymbol{v}_r}{|\boldsymbol{v}_r|} \tag{2.43}$$

Subsequent replacement of the dot product $\mathbf{v}_a \,.\, \mathbf{v}_r$ with the individual components of an n-dimensional vector, equation (2.44) can be formed.

$$|\mathbf{v}_{r1}| = \frac{\sum_{j=1}^{n} \left(w_{1j} \cdot w'_{1j} \right)}{|\mathbf{v}_r|} \quad \text{with vectors} \quad \boldsymbol{v}_a = \begin{pmatrix} w_{11} \\ w_{12} \\ \vdots \\ w_{1n} \end{pmatrix} \quad \text{and} \quad \boldsymbol{v}_r = \begin{pmatrix} w'_{11} \\ w'_{12} \\ \vdots \\ w'_{1n} \end{pmatrix} \tag{2.44}$$

Given that the direction of vector $\mathbf{v}_r$ remains unchanged and $\mathbf{v}_{rl}$ is pointing in the same direction, the length of $\mathbf{v}_r$ must be adjusted by altering the vector component values for each dimension. To find the new vector components without changing the vectors direction, each angle for each dimension needs to be kept constant. In coordinate geometry, if two points are on the same line, then their height distance ratios are constant, as illustrated in figure 2.10 and equation (2.45).



$$\frac{AC}{AD} = \frac{AB}{AE} \tag{2.45}$$

Figure 2.10 Vectors with the same direction have identical component ratios.

In analogy to (2.45), the resulting vector length $\mathbf{v}_r$ is equivalent to the distance $\overline{AC}$ and the distance $\overline{AD}$ can be represented by any arbitrary vector component of $\mathbf{v}_r$. The same is valid for the new error minimised vector $\mathbf{v}_{rl}$, the distance $\overline{AB}$ can represent its length equivalent and the distance $\overline{AE}$ can represent any arbitrary vector component.

On the basis that the vector components of vectors $\mathbf{v}_r$ and $\mathbf{v}_{rl}$ are defined as in (2.46), the replacement of the triangle components in (2.45) with their appropriate vector representations, is leading to equation (2.47).

$$\mathbf{v}_r = \begin{pmatrix} w'_{11} \\ w'_{12} \\ \vdots \\ w'_{1n} \end{pmatrix} \qquad \mathbf{v}_{rl} = \begin{pmatrix} w_{rl1} \\ w_{rl2} \\ \vdots \\ w_{rln} \end{pmatrix} \tag{2.46}$$

Substitution of equation (2.45) with the vector lengths and components from equation (2.46) leads to equation (2.47).

$$\frac{|v_r|}{w'_{1n}} = \frac{|v_{r1}|}{w_{r1n}} \tag{2.47}$$

Changing (2.47) in order to show the unknown vector components w_{r1n} of vector $\mathbf{v}_{r1}$ to the left, leads to (2.48).

$$w_{r1n} = \frac{w'_{1n}}{|v_r|} \cdot |v_{r1}| \tag{2.48}$$

With equation (2.44), $|\mathbf{v}_{r1}|$ can be substituted in (2.48), leading to equation (2.49).

$$w_{r1n} = \frac{w'_{1n}}{|\mathbf{v}_r|} \cdot \frac{\sum_{j=1}^{n} \left(w_{1j} \cdot w'_{1j}\right)}{|\mathbf{v}_r|} = \frac{w'_{1n} \cdot \sum_{j=1}^{n} \left(w_{1j} \cdot w'_{1j}\right)}{|\mathbf{v}_r|^2} \tag{2.49}$$

Equation (2.49) can be interpreted as follows: Each component of vector $\mathbf{v}_{r1}$ (w_{r1n}), which has been error minimised with regards to vector $\mathbf{v}_a$, can be determined by multiplying any of the n-dimensional components of $\mathbf{v}_r$ (w'_{1n}) with the dot product of $\mathbf{v}_a$. $\mathbf{v}_r$ or expressed with vector components $\left(\sum_n \left(w_{1n} \cdot w'_{1n}\right)\right)$ and divided by the resulting vector length squared $|\mathbf{v}_r|^2$.

Looking back to figure 2.8, vector $\mathbf{v}_r$ can be calculated with vectors $\mathbf{v}_a$ and $\mathbf{v}_b$ and vector $\mathbf{v}_{r1}$ can be calculated with vectors $\mathbf{v}_a$ and $\mathbf{v}_r$. Suppose that vectors $\mathbf{v}_a$ and $\mathbf{v}_b$ are interchangeable by swapping their indexes, vector $\mathbf{v}_{r2}$ can be calculated with the same equations as vector $\mathbf{v}_{r1}$ except using vector pair $\mathbf{v}_b$ and $\mathbf{v}_r$ instead of pair $\mathbf{v}_a$ and $\mathbf{v}_r$.

To find the factor F, which extends the length of vector $\mathbf{v}_{r1}$ to vector $\mathbf{v}_{r2}$, both vectors need to be computed and divided as shown in equation (2.50).

$$F = \frac{|v_{r2}|}{|v_{r1}|} \quad \text{so that} \quad v_{r2} = v_{r1} \cdot F \tag{2.50}$$

With equation (2.44) altered to determine vectors $\mathbf{v}_{r1}$ and $\mathbf{v}_{r2}$, equation (2.51) can be derived.

$$|\mathbf{v}_{r1}| = \frac{\sum_{j=1}^{n}\left(w_{1j} \cdot w'_{1j}\right)}{|\mathbf{v}_r|} \quad \text{and} \quad |\mathbf{v}_{r2}| = \frac{\sum_{j=1}^{n}\left(w_{2j} \cdot w'_{1j}\right)}{|\mathbf{v}_r|} \tag{2.51}$$

Utilising the vector lengths $|\mathbf{v}_{r1}|$ and $|\mathbf{v}_{r2}|$ in equation (2.51) for substitution in (2.50), equation (2.52) can be constructed.

$$F = \frac{|\mathbf{v}_{r2}|}{|\mathbf{v}_{r1}|} = \frac{\sum_{j=1}^{n}\left(w_{2j} \cdot w'_{1j}\right)}{\sum_{j=1}^{n}\left(w_{1j} \cdot w'_{1j}\right)} \tag{2.52}$$

Equation (2.52) can be interpreted as the division of the cross products of vectors $\mathbf{v}_b$. $\mathbf{v}_r$ and vectors $\mathbf{v}_a$. $\mathbf{v}_r$. Therefore, equation (2.52) can be expressed in vector algebra as the division of two cross products as shown in equation (2.53) or derived in the same way as equation (2.52) but using (2.43) instead of (2.44).

$$F = \frac{\boldsymbol{v}_b \cdot \boldsymbol{v}_r}{\boldsymbol{v}_a \cdot \boldsymbol{v}_r} \tag{2.53}$$

To summarise the process of converting the original vectors $\mathbf{v}_a$ and $\mathbf{v}_b$ into $\mathbf{v}_{r1}$ and F e.g. with both original vector in two-dimensional space (n=2), the following steps are required:

1. Calculation of the resulting vector $\mathbf{v}_r$ weighted by the original vector components as presented in section 2.5 and equation (2.37).

$$\text{With} \quad \boldsymbol{v}_a = \begin{pmatrix} w_{11} \\ w_{12} \end{pmatrix} \quad \text{and} \quad \boldsymbol{v}_b = \begin{pmatrix} w_{21} \\ w_{22} \end{pmatrix}, \quad \text{vector} \quad \boldsymbol{v}_r = \begin{pmatrix} w'_{11} \\ w'_{12} \end{pmatrix} = \begin{pmatrix} \dfrac{w_{11}^{\,2} + w_{21}^{\,2}}{w_{11} + w_{21}} \\ \dfrac{w_{12}^{\,2} + w_{22}^{\,2}}{w_{12} + w_{22}} \end{pmatrix}.$$

2. Modification of the length of $\mathbf{v}_r$ to obtain $\mathbf{v}_{r1}$ to minimise the distance between $\mathbf{v}_r$ and one of the original vectors e.g. $\mathbf{v}_a$ as presented in section 2.6 and

equation (2.49) so that $\mathbf{v}_{r1} = \begin{pmatrix} \frac{w'_{11} \cdot \left[\left(w_{11} \cdot w'_{11}\right) + \left(w_{12} \cdot w'_{12}\right)\right]}{|\mathbf{v}_r|^2} \\ \frac{w'_{12} \cdot \left[\left(w_{11} \cdot w'_{11}\right) + \left(w_{12} \cdot w'_{12}\right)\right]}{|\mathbf{v}_r|^2} \end{pmatrix}$.

3. Calculation of the length adjustment factor F to obtain $\mathbf{v}_{r2}$ for the minimisation of the distance between $\mathbf{v}_r$ and the second original vector e.g. $\mathbf{v}_b$ as presented in section 2.6 and equation (2.52), n=2, so that $F = \frac{\left(w_{21} \cdot w'_{11}\right) + \left(w_{22} \cdot w'_{12}\right)}{\left(w_{11} \cdot w'_{11}\right) + \left(w_{12} \cdot w'_{12}\right)}$.

2.7 Combination of the output weights

With neuron linking only the weights associated to the neurons in question are involved in the calculations for the linking process. Because linking is applicable to the neurons of the hidden layer, further connections towards the next layer or to the output layer must be considered. The fact that two neurons will be converted into one neuron is that after linking one connection to the next layer will become superfluous, see figure 2.11. Superfluous weight connections cannot just be dismissed. They need to be integrated into the weight connection used after linking. Therefore output weight connection algorithms or simplifications to avoid inaccuracies need to be derived.

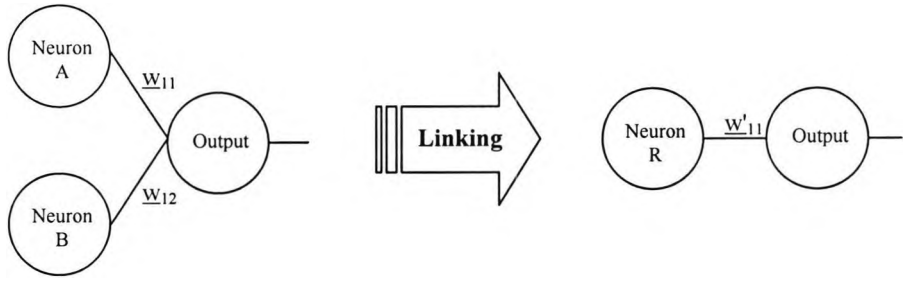


Figure 2.11 Linking hidden neurons requires weight combination from the next layer.

Since linking will always involve two neurons at a time as shown in figure 2.11, the forward path calculation of a network consisting of two neurons can be written as shown in equation (2.54).

$$out = f\left(f\left(\mathbf{x}_{in} \cdot \mathbf{m}_{h}\right)\right) \cdot \mathbf{v}_{o} = f\left(f\left(\sum_{j=1}^{n} \left(x_j \cdot w_{1j}\right)\right) \cdot \underline{w}_{11} + f\left(\sum_{j=1}^{n} \left(x_j \cdot w_{2j}\right)\right) \cdot \underline{w}_{12} \right) \quad (2.54)$$

With $\mathbf{x}_{in}$ being the input vector, $\mathbf{m}_h$ being the hidden layer matrix and $\mathbf{v}_o$ as the output vector. The neuron activation function is denoted as f() and the number of vector components in $\mathbf{x}_{in}$ is represented as n.

Combination of the weights connecting the next layer should accomplish the same output as achieved prior to linking. As a result, it can be assumed that out = out' with the linked neuron vectors. Consequently, the forward path of the linked network can be written as shown in equation (2.55).

$$out' = f\left(f\left(\sum_{j=1}^{n} \left(x_j \cdot w'_{1j}\right)\right) \cdot \underline{w}'_{11} \cdot \left(1 + F\right)\right) \quad (2.55)$$

The term (1+F) is used to expand the single linked neuron into two neurons, namely $\mathbf{v}_{r1}$ and $\mathbf{v}_{r2}$. It represents the length adjustment factor required to allow the linking of vectors with distinctive lengths.

In order to equate *out* (2.54) with *out'* (2.55) an easy but important simplification is made. With the assumption what the neuron activation function is a linear activation function with a slope of 1 and going through the point of origin (0,0), all activation functions can be ignored. With this assumption, the results of the following equations will only be valid for neurons with linear activation functions.

$$\left(\sum_{j=1}^{n} \left(x_j \cdot w_{1j}\right)\right) \cdot \underline{w}_{11} + \left(\sum_{j=1}^{n} \left(x_j \cdot w_{2j}\right)\right) \cdot \underline{w}_{12} = \left(\sum_{j=1}^{n} \left(x_j \cdot w'_{1j}\right)\right) \cdot \underline{w}'_{11} \cdot \left(1 + F\right) \quad (2.56)$$

$$\frac{\left(\sum_{j=1}^{n} \left(x_j \cdot w_{1j}\right)\right) \cdot \underline{w}_{11} + \left(\sum_{j=1}^{n} \left(x_j \cdot w_{2j}\right)\right) \cdot \underline{w}_{12}}{\left(\sum_{j=1}^{n} \left(x_j \cdot w'_{1j}\right)\right) \cdot \left(1 + F\right)} = \underline{w}'_{11} \quad (2.57)$$

Setting the linked network output equal to the original network output equation (2.56) can be derived. In order to calculate the combined output weight for the next layer, equation (2.57) is resolved for the output weight $\underline{w}'_{11}$. Because the input vector values can be of any arbitrary value, they have been set to 1 for further simplification.
In a case where the input vector consists of two components (n=2) and the network activation functions are all linear, equation (2.58) can be used to calculate the combined output weight.

$$\underline{w}'_{11} = \frac{(w_{11} + w_{12}) \cdot \underline{w}_{11} + (w_{21} + w_{22}) \cdot \underline{w}_{12}}{(w'_{11} + w'_{12}) \cdot (1 + F)} \tag{2.58}$$

Under perfect conditions $w'_{11} = w_{11}$, $w'_{12} = w_{12}$ and $w'_{11} \cdot F = w_{21}$, $w'_{12} \cdot F = w_{22}$, equation (2.59) can be derived.

$$\underline{w}'_{11} = \frac{(w_{11} + w_{12}) \cdot \underline{w}_{11} + (w_{21} + w_{22}) \cdot \underline{w}_{12}}{(w_{11} + w_{12}) + (w_{21} + w_{22})} \tag{2.59}$$

2.9 Conclusion

Neurons and their knowledge can be seen as pure vectors, containing information extracted from the training data. If two neurons contain similar knowledge their vectors will point in similar directions in hyperspace. But the length of the vectors does not seem to represent knowledge as such; it can be seen as a representation of certainty or strength of the knowledge. Therefore, neurons can be linked if they contain similar knowledge even if their length or certainty differs.
With the introduction of the vector length correction factor F, neurons that point in similar or opposite direction can be linked as long as the angle difference does not exceed a certain value. If weight vectors are facing opposite directions, the factor F will be negative, restricting the angle a vector can point to 0...180 ° instead of 0...360° doubling the probability that vectors are pointing in similar directions.

In this chapter, different types of neuron combination methods have been discussed and analysed on their performance on a measure of error. Simple averaging has an error distribution of 50% for each vector as shown in equation (2.13). In the case of weighted averaging by vector length the error distribution is non-proportional to the length of each vector as shown in equation (2.32). If weighting is performed on a vector component basis the error distribution for each vector component is it's component length divided by the sum of all component lengths as shown in equation (2.39). This weighting of vector components has been further extended to accommodate the vector length correction factor F that permits linking of vectors that have substantial differences in vector lengths.

There are certainly more possible techniques of linking neuron weight vectors into a new vector but the simplicity of the method using weighted vector components and length adjustment bears a great advantage.

Chapter 3

Pruning of Neural Network Weight Matrixes

3.1 Introduction

After the derivation of the linking equations in chapter 2, these equations are now used to demonstrate their application for pruning of weight matrixes. This chapter will highlight the effectiveness of pruning with the linking algorithm on a small, fully connected neural network. It will show how neurons can be combined to achieve pruning with the linking process instead of being removed to reduce redundant information held within the network. For this purpose, a single neural network will be trained on a simulated function approximation problem constructed as a numerical example of the utilisation of the linking process for pruning of post-trained networks.
After network training, statistical analysis has been used to measure its accuracy on seen and un-seen training data. This analysis has been used as reference for the pruning results in order to evaluate its performance.
The outcome of network pruning is heavily dependent on the complexity of the objective function contained in the training patterns and the size of the hidden layer(s) of the network. If the objective function is uncomplicated, for example linear, not many hidden neurons will be required for modelling. If too many hidden neurons have been used on a simple objective function, the weights of some redundant neurons will

have counterweights within the weight matrix that will render them superfluous [134]. If only one half of the set of compensating weights is removed, an imbalance can occur. Therefore it is important to find compensating weights in couples or groups. Furthermore, many redundant neurons can be expected if a simple objective function is used in conjunction with a large number of neurons in the hidden layer.

If, on the other hand, the objective function is complex and the number of hidden neurons in the hidden layer is small, redundant neurons may not be easily found. For the experiment in this chapter, a medium sized network with 20 hidden neurons and a reasonable complex objective function have been used in order to find redundant neurons that can be linked.

During network pruning the network accuracy may be reduced since trained information contained within the neurons is altered. But during pruning the number of free parameters is reduced, which will influence the generalisation capabilities of the network [135-137].

It is essential to set an objective for initiating the action of network pruning. This utilisation objective is dependant on the area of intended use of the network. If the network utilisation objective lies within the region of extrapolation, good generalisation capabilities outside the input space present in the training data is the priority. For this purpose it is recommended that the extracted testing data should be close to the extrapolation space.

But if the utilisation objective is interpolation of the input space close to the location of the training patterns, recall accuracy of the network must remain high during and after pruning.

Therefore it is recommended to analyse the sensitivity of the neurons prior to pruning [119, 138, 139]. Neurons with high sensitivity towards the overall network performance should only be included in the linking process if their induced error is low, keeping the overall network performance mainly unchanged. Neuron sensitivity analysis as a significance measure can be in included in the linking equation if used for weighting of the total error between the weight vectors.

3.2 Linking of two Hidden Neurons

This section gives an introduction to pruning using a simplified numeric example to show how two hidden neurons can be linked into one hidden neuron. For this purpose a neural network with two inputs, two hidden neurons and one output neuron (2:2:1) has been trained on a two dimensional mathematical function. For reasons of simplicity and to concentrate on the linking of hidden neuron weight vectors, all neurons have a linear activation function f(x) = x. To make it difficult for neurons with a linear activation function to exactly memorise the objective function [77], a non-linear objective function, shown in equation (3.1), has been used to generate training and testing data.

$$f(x) = 3.5 \cdot x_1 + 1.5 \cdot x_2^2 + 1 \tag{3.1}$$

The network weights have been initialised in the range of ±0.7 with 400 training and 100 testing patterns. The stopping criterion has been set to the point where the Sum Square Error (SSE) of the testing data set, referred to as the generalisation error, SSE_{gen} reached a plateau and did not increase any further $\frac{\Delta SSE_{gen}}{\Delta t} \approx 0$. After approximately 3000 training iterations this point was found with the learning rate of η=0.01 and the momentum of m=0.3. At this point training was stopped and the SSE of the network with the training data SSE_{trn} was 0.4702 as shown in figure 3.1 and the testing data SSE_{gen} was 7.681 as shown in figure 3.2.

Although generating the training data is simple, there were considerations to be taken into account with regards to the testing data. For a fair measure of the generalisation error of a neural network for a simulated problem, the testing data has been extracted mainly from the centre of the generated data set to avoid extrapolation issues. In cases where the majority of the testing data is taken at the edges of the input space, the network will have to extrapolate causing high generalisation errors. Consequently,

testing data is best taken where training data is surrounding the input space, thus leaving input space borders in the training data for better generalisation results [54].

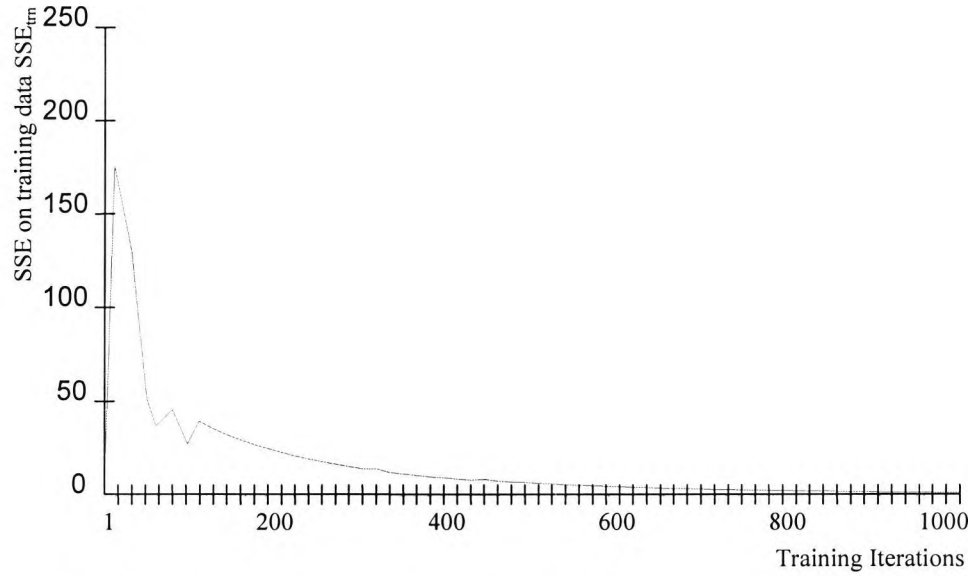


Figure 3.1 SSE_{trn} during training plotted against the number of batch training iterations.

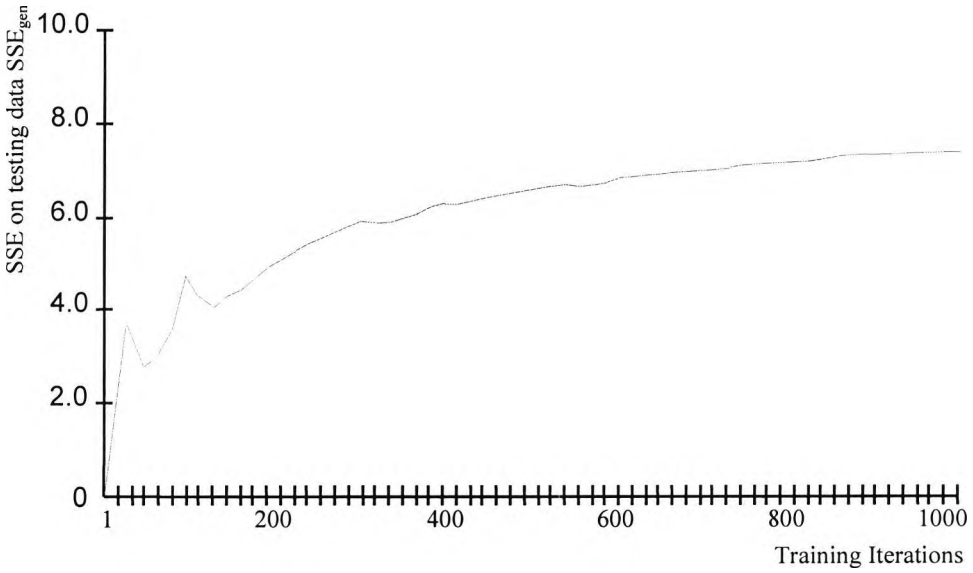


Figure 3.2 SSE_{gen} during training plotted against the number of batch training iterations.

Simplification of an uncomplicated 2:2:1 neural network into a 2:1:1 network via linking of neurons does not seem to bring an advantage to the user at first sight. But the principle of combining neurons remains the same regardless of the size of the network or the dimensionality of the hidden weight vector. With the ability to link neurons, large networks can be pruned or small networks can be linked to alter the knowledge of a network. For these reasons, the hidden neurons of the trained 2:2:1 network are linked into one hidden neuron to emphasise the simplicity of the linking paradigm.

Prior to linking, the weight vectors need to be tested for their direction because for the linking process to succeed they have to be sufficiently close. In this instance the neural network training has been restarted if the hidden weight vectors were not sufficiently close in direction, until a suitable set of weight vectors was found.

After only a few restarts, a suitable weight matrix with an angle difference of 172.89° was found. Although the angle difference appears to be high, caused by the fact that the vectors are pointing in opposite directions, the deployment of a negative reconstruction factor F will move both vectors into the same quadrant, leaving an angle difference of only 7.11°. The networks hidden and output weight matrixes and their vector notations, as denoted in chapter 2, are shown in equations (3.2) and (3.3).

$$w_{ji} = \begin{bmatrix} -2.94 & -7.70 & -0.32 \\ 4.35 & 8.21 & 0.52 \end{bmatrix} \qquad w_{kj} = \begin{bmatrix} 1.04 & 1.03 & -0.39 \end{bmatrix} \quad (3.2)$$

$$\mathbf{v}_a = \begin{pmatrix} -2.94 \\ -7.70 \\ -0.32 \end{pmatrix} \qquad \mathbf{v}_b = \begin{pmatrix} 4.35 \\ 8.21 \\ 0.52 \end{pmatrix} \qquad \mathbf{v}_o = \begin{pmatrix} 1.04 \\ 1.03 \\ -0.39 \end{pmatrix} \quad (3.3)$$

With help of the equations derived in chapter 2 for linking two hidden neurons into one, the objective is to create only two vectors, one for the resulting hidden neuron and a modified vector for the output neuron, since the hidden layer has been reduced

from 2 to 1 hidden neuron, as shown in figure 3.3. The new hidden and output weight matrixes and vectors after pruning are shown in equations (3.4) and (3.5).

$$w'_{ji} = \begin{bmatrix} w'_{11} & w'_{12} & w'_{1B} \end{bmatrix} \qquad w'_{kj} = \begin{bmatrix} \underline{w}'_{11} & \underline{w}'_{1B} \end{bmatrix} \tag{3.4}$$

$$v'_{r1} = \begin{pmatrix} w'_{11} \\ w'_{12} \\ w'_{1B} \end{pmatrix} \qquad v'_{o} = \begin{pmatrix} \underline{w}'_{11} \\ \underline{w}'_{1B} \end{pmatrix} \tag{3.5}$$

Changing the network topology has created an equivalent network that performed similarly on testing data as the original network did. The change from a fully connected to the linked network topology is graphically presented in figure 3.3.

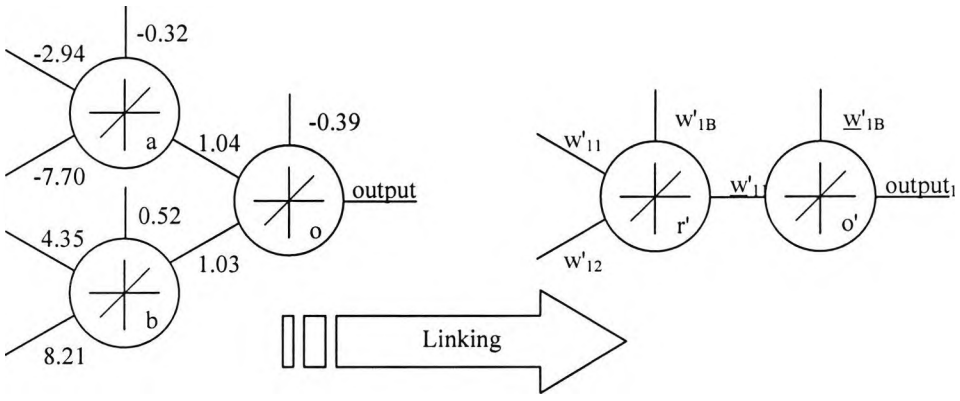


Figure 3.3 Transformation of a 2:2:1 network into a 2:1:1 network by linking of a hidden neuron.

Several possible techniques for creating a resulting hidden neuron weight vector $\mathbf{v'}_r$ have been introduced in chapter 2 but only one for the combination of the output weight vector $\mathbf{v'}_o$. Therefore the combination of the output vector $\mathbf{v'}_o$ will be introduced in the next section and alternative equations for the creation of the resulting vector $\mathbf{v'}_r$ will be introduced in later sections.

3.2.1 Combination of Output Weights

In section 2.7 the equation for the modified output weight vector has been derived. Its purpose is to compensate for the loss of one of the hidden neurons involved in the linking process. When two hidden neurons are linked, one of the connections to the next layer, in this example the output layer, will become superfluous and must therefore be combined with the weight used to connect the resulting hidden neuron. One of the major simplifications with respect to the combination of the output weight vector has been the linear neuron activation function, which has been intentionally used for this example to concentrate on linking of hidden neurons.
With equation (2.58) the combined output weight can be calculated with the vectors in (3.3) and (3.4) as given in equation (3.6).

$$w'_{11} = \frac{(w_{11}+w_{12})\cdot \underline{w}_{11} + (w_{21}+w_{22})\cdot \underline{w}_{12}}{(w_{11}+w_{12})+(w_{21}+w_{22})} =$$

$$\frac{(-2.94-7.70)\cdot 1.04 + (4.35+8.21)\cdot 1.03}{(-2.94-7.70)+(4.35+8.21)} = 0.994 \qquad (3.6)$$

Since the bias of the output neuron is not affected by the linking of the hidden neurons, no changes to $\underline{w}'_{1B}$ take place. Because of this, the bias remains unchanged and with the combined weight, the modified output weight vector $\mathbf{v}'_o$ for the linked hidden neuron can be calculated as shown in equation (3.7).

$$\mathbf{v}'_o = \begin{pmatrix} \underline{w}'_{11} \\ \underline{w}_{1B} \end{pmatrix} = \begin{pmatrix} 0.994 \\ -0.39 \end{pmatrix} \qquad (3.7)$$

3.2.2 Linking for the Purpose of Pruning

The essence of linking neurons is to link two hidden neurons trained with the training data into one resulting hidden neuron and a reconstruction factor F. The equations derived in section 2.6 for hidden neuron linking will now be used for this numeric example to create the resulting hidden neuron. For this purpose the following three steps are required:

1. Calculation of the resulting vector $\mathbf{v}_r$
2. Modification of the length of $\mathbf{v}_r$ to obtain $\mathbf{v}_{rl}$
3. Calculation of the length adjustment factor F.

Step 1 involves the evaluation of equation (2.37) as shown in equation (3.8) below.

$$\mathbf{v}_r = \begin{pmatrix} w'_{11} \\ w'_{12} \\ w'_{1B} \end{pmatrix} = \begin{pmatrix} \dfrac{w_{11}^{\,2} + w_{21}^{\,2}}{w_{11} + w_{21}} \\ \dfrac{w_{12}^{\,2} + w_{22}^{\,2}}{w_{12} + w_{22}} \\ \dfrac{w_{1B}^{\,2} + w_{2B}^{\,2}}{w_{1B} + w_{2B}} \end{pmatrix} = \begin{pmatrix} -\dfrac{2.94^2 + 4.35^2}{2.94 + 4.35} \\ -\dfrac{7.70^2 + 8.21^2}{7.70 + 8.21} \\ -\dfrac{0.32^2 + 0.52^2}{0.32 + 0.52} \end{pmatrix} = \begin{pmatrix} -3.781 \\ -7.963 \\ -0.447 \end{pmatrix} \tag{3.8}$$

For the second step equation (2.49) is used to move the resulting vector $\mathbf{v}_r$ closer towards vector $\mathbf{v}_a$ as shown in equation (3.9).

$$\mathbf{v}_{r1} = \begin{pmatrix} w_{r11} \\ w_{r12} \\ w_{r1B} \end{pmatrix} = \begin{pmatrix} \dfrac{w'_{11} \cdot \left((w_{11} \cdot w'_{11}) + (w_{12} \cdot w'_{12}) + (w_{1B} \cdot w'_{1B})\right)}{|\mathbf{v}_r|^2} \\ \dfrac{w'_{12} \cdot \left((w_{11} \cdot w'_{11}) + (w_{12} \cdot w'_{12}) + (w_{1B} \cdot w'_{1B})\right)}{|\mathbf{v}_r|^2} \\ \dfrac{w'_{1B} \cdot \left((w_{11} \cdot w'_{11}) + (w_{12} \cdot w'_{12}) + (w_{1B} \cdot w'_{1B})\right)}{|\mathbf{v}_r|^2} \end{pmatrix} = \begin{pmatrix} -3.523 \\ -7.419 \\ -0.416 \end{pmatrix} \tag{3.9}$$

In the last step equation (2.52) is used to calculate the vector length adjustment factor F, which is used to stretch vector $\mathbf{v}_{r1}$ towards vector $\mathbf{v}_b$ to minimise the distance between the vectors. The computation of F is given in equation (3.10).

$$F = \frac{(w_{21} \cdot w'_{11}) + (w_{22} \cdot w'_{12}) + (w_{2B} \cdot w'_{1B})}{(w_{11} \cdot w'_{11}) + (w_{12} \cdot w'_{12}) + (w_{1B} \cdot w'_{1B})} = -1.1304 \qquad (3.10)$$

For easier visualisation, both vectors $\mathbf{v}_a$ and $-\mathbf{v}_b$ are shown with their resulting vectors $\mathbf{v}_{r1}$ and $-\mathbf{v}_{r2}$ ($\mathbf{v}_{r2} = F \cdot \mathbf{v}_{r1}$) in figure 3.4. It becomes apparent, that vectors $\mathbf{v}_a$ and $\mathbf{v}_b$ are pointing in opposite directions since the correction factor F is negative. Therefore, vectors $\mathbf{v}_b$ and $\mathbf{v}_{r2}$ have been inverted in figure 3.4 so that all vectors can be presented in the same quadrant for easier visualisation.

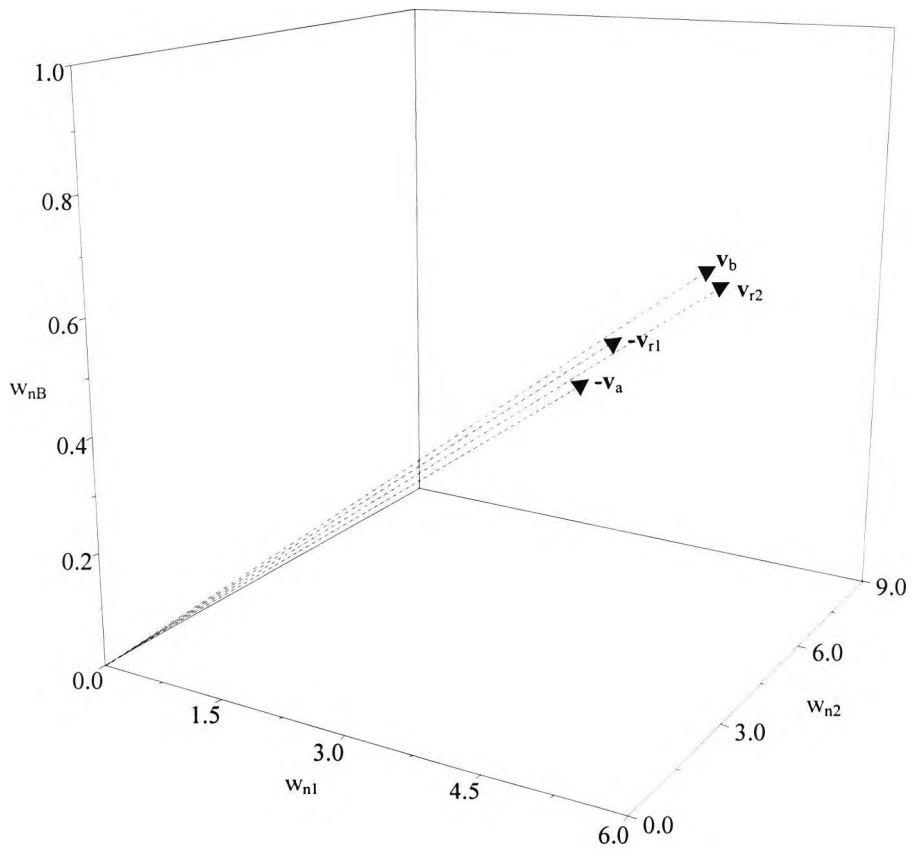


Figure 3.4 Linking of vectors v_a and v_b into one vector v_{r1} and a length adjustment factor F.

With all calculations completed, the new linked network can be assembled as shown in figure 3.5.

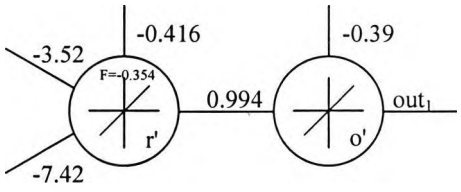


Figure 3.5 Network with only one hidden neuron after linking.

3.2.3 Measuring Linked Neurons Output Performance

To measure the performance benchmarks of the network after linking, a modified backpropagation forward path calculation needs to be applied. The forward path calculation will act as if the linked neuron contains two vectors instead of one. The first vector involved in the forward path calculation is $\mathbf{v}_{r1}$, without vector length adjustment, and the second vector is $\mathbf{v}_{r2}$, which includes the adjustment factor F.

In the current example the hidden layer neurons are denoted with the indices j and the output neurons with k. With this notion, the summed input into the neurons are denoted as net_j and net_k and the neurons output as o_j and o_k for hidden and output neurons respectively. The linear activation function used caused the neurons output to be equal to the neurons summed input $o_j = net_j$ and $o_k = net_k$. Following the definitions, the forward path can be calculated as given in (3.11).

$$o'_k = net'_k = o'_j \cdot \underline{w}'_{11} + \underline{w}'_{1B} = net'_j \cdot \underline{w}'_{11} + \underline{w}'_{1B} \tag{3.11}$$

The summed input of the linked neuron net_j will act as if the linked neuron contains two vectors $\mathbf{v}_{r1}$ and $\mathbf{v}_{r2}$.

$$net'_j = \left(\boldsymbol{x}_{in} \cdot v_{r1} + \boldsymbol{x}_{in} \cdot v_{r2}\right) = \left(\boldsymbol{x}_{in} \cdot v_{r1} + \boldsymbol{x}_{in} \cdot v_{r1} \cdot F\right) = (1 + F) \cdot x_{in} \cdot v_{r1} \tag{3.12}$$

With $\mathbf{x}_{in}$ as the input vector $x_{in} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$.

Substitution of the appropriate vector components of $\mathbf{v}_{rl}$ in equation (3.12) with (3.5) will lead to (3.13).

$$net'_j = \left(\left(x_1 \cdot w'_{11} + x_2 \cdot w'_{12} + w'_{1B}\right) + \left(x_1 \cdot w'_{11} + x_2 \cdot w'_{12} + w'_{1B}\right) \cdot F\right) \quad (3.13)$$

Now substituting equation (3.13) into (3.11) will produce the entire equation for the network output o_k with respect to the linked neuron as shown in (3.14).

$$o_k = \left(\left(x_1 \cdot w'_{11} + x_2 \cdot w'_{12} + w'_{1B}\right) + \left(x_1 \cdot w'_{11} + x_2 \cdot w'_{12} + w'_{1B}\right) \cdot F\right) \cdot \underline{w}'_{11} + \underline{w}'_{1B} \quad (3.14)$$

After reapplication of the training and testing data on the simplified linked network the linked network performance benchmarks have been calculated and presented in table 3.1. To increase statistical reliability the entire experiment has been repeated 30 times and the averages, minimums and maximums for training and testing pattern are reported. It can be seen, that the neuron linking has caused an improvement of the networks overall generalisation accuracy on average from 7.675 to 2.870 a reduction of 63%. This result can be based on the fact that the network has been over-trained, since the network did learn the objective function very accurately. On the other hand, the recall error SSE_{trn} has increased by a significant amount, by factor 13, after linking. The low SSE_{trn} error of only 0.49 confirms that the network was over-trained. Overall, linking has reduced the generalisation error but increased the recall error.

Table 3.1 Comparison between trained and linked network benchmarks for 30 runs.

Description	Trained Network				Linked Network			
	Min	Max	Average	StDev	Min	Max	Average	StDev
SSE_{trn}	0.265	1.221	0.4931	0.2451	5.695	7.486	6.950	0.4484
SSE_{gen}	6.708	8.052	7.675	0.3399	2.466	3.044	2.870	0.1598

With the linking, the network hyperplane has been shifted towards the solution space of the linked weight vector $\mathbf{v}_{rl}$ as shown in figure 3.6, where the complete input space for x_1 and x_2 ($0 \leq x_1 \leq 1$ and $0 \leq x_2 \leq 1$) against the networks output o_k is presented in a 3-dimensional coordinate system.

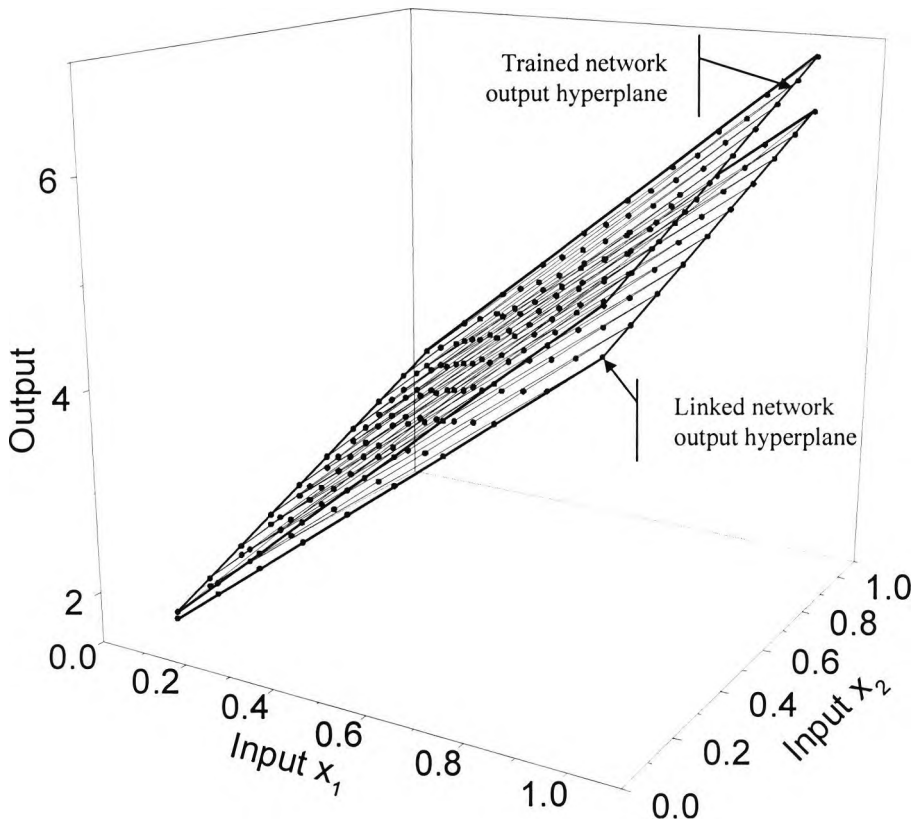


Figure 3.6 The trained and linked network hyperplanes presented for the entire input space.

This simplified pruning example on a regression-based problem has utilised all relevant equations introduced for neuron linking on a small example. It has addressed the linking process in detail with little consideration of the combination of the weights connected to the neurons output, referred to as the output layer. To overcome the problem of combining the weights connecting the linked hidden neurons to the output layer, the chosen activation function for the output neuron was linear. In the next section a larger network has been trained with a more complex non-linear regression problem where the activation function is sigmoidal. This network has the output weights set to 1, which are frozen during training, to avoid the constraint of a linear activation function in the output neuron.

3.3 Hidden Neuron Linking of a Neural Network for Pruning

For the purpose of conceptual presentation of the linking process applied to pruning, a reasonable complex non-linear objective function (3.15) and a medium-size network have been chosen. To get around the problem with the combination of weights connected to the output of the linked neurons, the weights between the hidden layer and the output layer have been set to 1 and frozen, so that they cannot be altered during training. By setting the weights between the hidden and output layer to 1, no weight combination calculations on the output layer are required. Because of this, a more complex activation function than the linear activation function, which has been used in the previous section, can be used. Since non-linear activation functions perform better on a non-linear objective function, a more complex objective function can now be used for the training of the neural network. A summary of the neural network configuration parameters used in this section is given in table 3.2.

Table 3.2 The parameters of the neural network used in this section.

Description	Value
Input Neurons	2
Hidden Neurons	20
Output neurons	1
Activation Function	non-symmetric sigmoid
Initialisation	±0.6
Learning Factor	0.1
Momentum	0.3
Number of training patterns	500
Number of testing patterns	100

The networks topological parameters are 2:20:1 with a non-symmetric sigmoid activation function and frozen output weights of the value 1. For the generation of the training and testing data a non-linear objective function as shown in equation (3.15) has been employed.

$$f(x_1, x_2) = 0.5 + 0.5 \cdot \sin(20x_1) \cdot e^{(-x_1 \cdot x_2)} \tag{3.15}$$

3.3.1 Training of Hidden Neurons

Training and testing data has been generated with random numbers for x_1 and x_2 in the input space range of $0 \leq x_1 \leq 1$ and $0 \leq x_2 \leq 1$. After training and testing data generation, the network has been initialised with weight values in the range of ± 0.7 and its output together with the target values are graphically presented in figure 3.7.

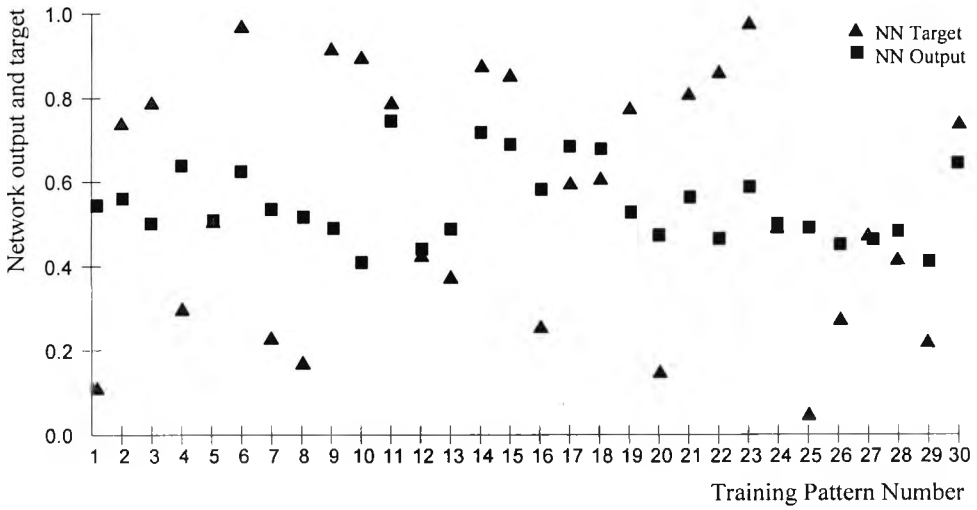


Figure 3.7 The objective function compared with the network output after initialisation.

In order to prevent training from being affected by the pattern sequence, if presented in a numerical ascending or descending order, x_1 and x_2 have been generated by a linear distribution random number generator, which has the same effect as shuffling the training data set. Shuffling training patterns prevents the network from learning associations between the data, instead of learning the objective function. It is important for optimal learning that the various patterns are presented in a different order for each training cycle.

Normalisation of the training and testing data was not required since the input range was chosen to be limited to a maximum of 1. The range of the network output was restricted by the sigmoid activation function and has been indirectly normalised by

defining the objective function in equation (3.15) in such a way that the largest output does not exceed 1.

The training algorithm selected was the standard backpropagation-learning algorithm for its good performance and ease of implementation. In the forward pass, the neuron errors and weight updates have been computed in batch mode, causing neuron weight only to be updated after presentation of the entire training data set. Batch mode reduces the risk of oscillation during training and convergence problems, but can only be effective if sufficient data samples are present.

Training has been continued until a specified training error stopping criterion was met. The stopping criterion has been met if there was no relevant change in the SSE_{trn} for more than 1000 iterations. This point can be seen as the energy minimum found by the steepest descent.

During training, the training or recall Sum Square Error SSE_{trn} on the training data and the generalisation performance on the testing data SSE_{gen} have been collected from the point of initialisation to the point where the stopping criteria was met, both are shown collectively in figure 3.8.

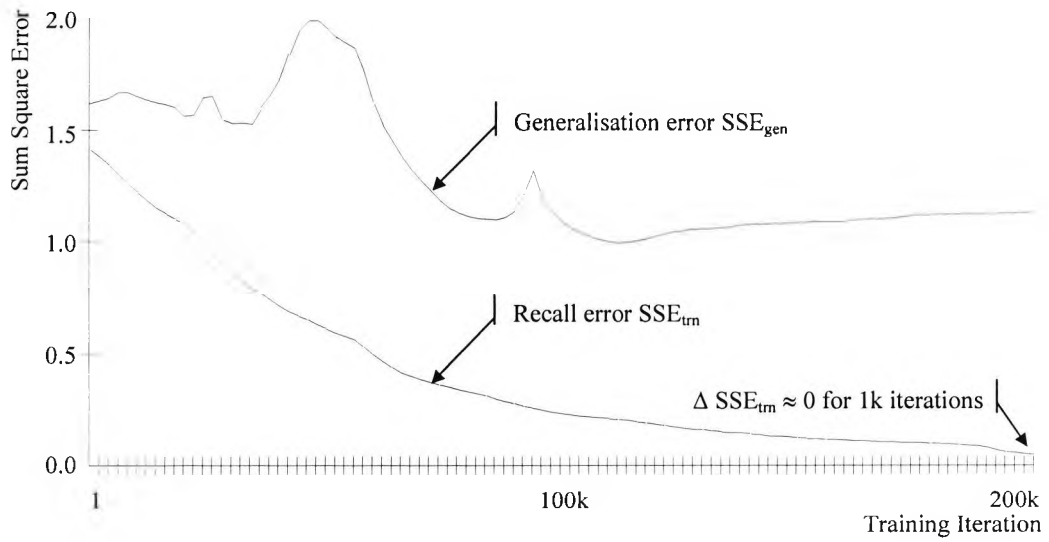


Figure 3.8 The recall and generalisation error during training of the neural network.

After approximately 200,000 iterations the stopping criterion for the training was satisfied. To visualise the network performance on the training data, the objective function against the network output it is plotted in figure 3.9.

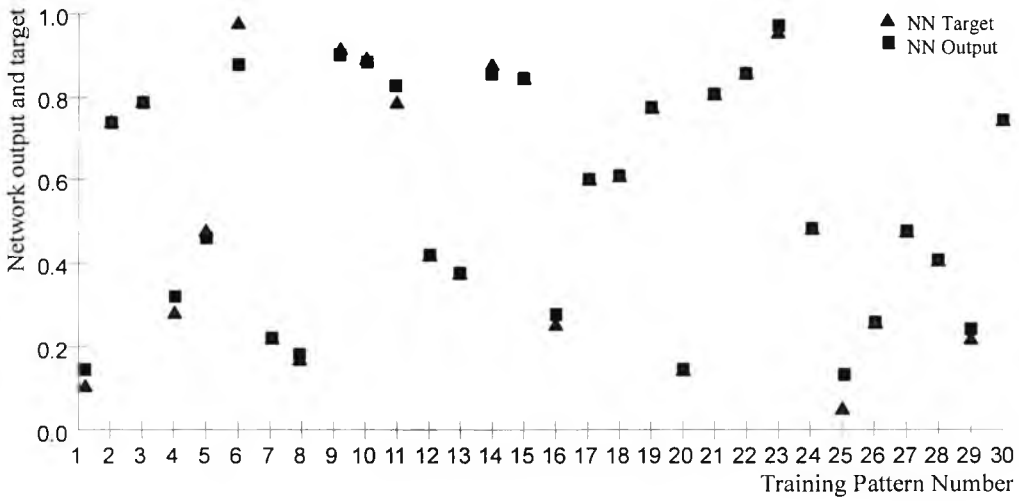


Figure 3.9 The objective function compared with the network output after training.

It can be seen in figure 3.8 that the SSE_{trn} after initialisation is high and is falling during the training. On the other hand, the generalisation error SSE_{gen} is increasing as training progresses. This is because the network is getting more specialised as training continues. During training, some weight vectors are changed to improve recall accuracy on the training data set and as training continues, the changes in weight vectors where definite data points are available gradually lessen. Whilst weight vectors, which are directly correlated to data points in the training patterns stabilise, uncorrelated weight vectors, which are free parameters used for generalisation, continue to change in one direction. This is causing the values of the weights to increase into extreme areas. Such an increase on one hand will create weights to counterweight on the other. These free weight vectors are generally pointing in the opposite direction to compensate for large weights of other free vectors. Weight vectors, which are most likely to be affected by counter weight increase, are vectors that are not associated to a data point in the training data. The problem is that exactly

these free vectors are responsible for the generalisation performance of the network and as a result the generalisation performance is reduced on an over-trained network. To overcome the problems with too many free parameters and over-training, the generalisation error SSE_{gen} can be monitored during training, which can be stopped if the generalisation error has reached a minimum. Another solution is to remove such free vectors via pruning [137, 140, 141].

3.3.2 Analysis of Hidden Neuron Weight Vectors

After completion of the training, the hidden weight vectors need to be analysed to identify those, which are pointing in the same or opposite directions. Table 3.3 shows the weight vectors of all hidden neurons of the network after training.

Table 3.3 Weight vectors of the hidden layer after training.

Vector Reference	w_{11}	w_{12}	w_{1B}	Vector Length
v_1	-1.212439	-3.047107	-9.828882	10.362
v_2	-48.65868	26.43653	27.10182	61.653
v_3	-28.27503	87.87204	-9.422333	92.789
v_4	-87.9455	88.14407	-41.08837	131.118
v_5	-0.370743	-6.823745	-10.10476	12.199
v_6	90.95148	19.12648	-34.60433	99.174
v_7	-46.05195	-90.5919	67.69712	122.109
v_8	-0.5285	-6.055976	-10.09197	11.781
v_9	-53.18032	0.324787	38.83406	65.851
v_{10}	-89.86122	90.19841	-42.12809	134.110
v_{11}	87.29947	-59.57665	-2.45873	105.720
v_{12}	-118.0139	11.72936	15.38128	119.589
v_{13}	90.14281	-38.08962	-33.1463	103.321
v_{14}	-117.8034	11.68759	15.35809	119.374
v_{15}	19.48965	64.01544	-56.13136	87.342
v_{16}	51.67821	73.66065	-81.22998	121.222
v_{17}	-59.00057	2.313215	43.91528	73.586
v_{18}	64.05762	-40.98364	-23.74516	79.667
v_{19}	110.1794	-251.8599	74.11477	284.721
v_{20}	19.74688	-48.62996	-7.57796	53.031

To find weight vectors suitable for linking, each angle between all vectors needs to be analysed. Since the weight matrix contains 20 rows, there are 380 possible angle calculations, which have been tested for the vector angle constraint of $\varphi = 10°$. Weight vectors that have an angle below 10° have been sorted in ascending order as presented in table 3.4. Because the sign of factor F can be changed, vectors located in Q3 are mapped into quadrant Q1 and vectors located in Q4 are mapped into Q2 as shown in figure 3.10.

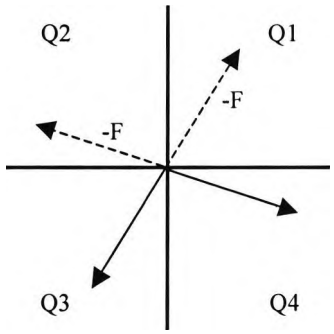


Figure 3.10 Vectors of quadrants Q3 and Q4 can be mapped into Q1 and Q2 respectively.

By mapping vectors from four quadrants into two, the probability of finding acceptably close vectors doubles.

Table 3.4 Angles between weight vectors in ascending order.

Vector pair	Angle between vectors
v_{12}, v_{14}	0.0102°
v_4, v_{10}	0.0612°
v_9, v_{17}	1.606°
v_8, v_5	3.172°
v_2, v_{13}	180°-171° = 9°[2]

[2] This vector has been mapped from Q3 to Q1.

Table 3.4 is presenting a list of vectors where the vector angle does not exceed 10°. This angle limitation has been defined as the *acceptance angle* φ throughout this thesis and has been chosen after a sequence of tests. A precise analysis of the effects of the acceptance angle on the SSE_{trn} and SSE_{gen} is given in a later section of this chapter.

3.3.3 Linking of Hidden Neurons

Subsequent to the vector angle analysis is the process of neuron linking. It follows the same method as linking from previous sections but without the requirement of combining the weights to the output layer since they have been set to one and frozen during training. Without repeating the equations used for linking of weight vector pairs listed in table 3.4, table 3.5 presents the linking results.

Table 3.5 Results of the combination of vectors with angles below 10° as listed in table 3.4.

Original vector references	Resulting vector v_{r1}			
	w_{11}	w_{12}	w_{1B}	Factor
v_{12}, v_{14}	-118.015	11.729	15.381	0.9982
v_4, v_{10}	-87.899	88.166	-41.140	1.0228
v_9, v_{17}	-52.933	1.947	39.088	1.1178
v_8, v_5	-0.455	-6.343	-9.911	1.0355
v_2, v_{13}	-33.822	24.667	40.552	-1.5698

Table 3.5 lists the components of all linked vectors involved in pruning. The components of the first two vector pairs were almost identical. Therefore the linked vector components in table 3.5 do not differ significantly if compared to table 3.3. It can be noted, that with increasing angle difference, the differences of linked and original vector components increase. In section 3.4, a more detailed investigation of vector component changes is presented.

3.4 Linking Analysis

The linking analysis is used to analyse the errors on individual weights of linked neurons. Neuron weights after training but before linking will be compared with neuron weights after linking by using the relative error with regards to the weight prior to linking. This is followed by a comparison of the neural network performances after training and after linking.

3.4.1 Analysis of Linked Neurons

To validate the results of table 3.5 an impact analysis on the effects of linking towards the differences between the original vectors and the reconstructed vectors $\mathbf{v}_{r1}$ and $\mathbf{v}_{r2}$, the relative errors of the changes in weight values are listed in table 3.6. The notation in table 3.6 will be that $\mathbf{v'}_{12}$ and $\mathbf{v'}_{14}$ represent $\mathbf{v}_{r1}$ and $\mathbf{v}_{r2}$ respectively for all vectors involved in the linking process, in accordance with figure 3.4.

Because of the extremely small angle difference of 0.01° between vectors $\mathbf{v}_{11}$, $\mathbf{v}_{14}$ and 0.06° between vectors $\mathbf{v}_4$, $\mathbf{v}_{10}$, the relative errors of their reconstructed vectors are expected to be very small. Consequently, these two vector pairs are ideal candidates for linking.

In the case of vector pair $\mathbf{v}_9$, $\mathbf{v}_{17}$, an unbalanced error of almost 500% on w_{12} can be noted. Even if the angle difference between $\mathbf{v}_9$ and $\mathbf{v}_{17}$ was only 1.6°, see table 3.4, such a large error on a single weight was not expected.

The justification for allowing such a high error lies within the relatively small weight value of w_{12} of $\mathbf{v}_9$, which is 0.32 compared to w_{12} of $\mathbf{v}_{17}$, which is 2.31. Since the value from w_{12} of $\mathbf{v}_9$ is smaller by several orders of magnitude than the largest weight involved ($|0.32| << |-59.0|$), the increase from 0.32 to 1.94 seems to be acceptable. In other words, considering that most absolute weight values of vectors $\mathbf{v}_9$ and $\mathbf{v}_{17}$ lie in the range between 40-60, smaller weights <1 contribute not as much towards the overall output compared to the large weights. Therefore small weights can accept a higher percentage error than large weights.

Vectors $\mathbf{v}_8$ and $\mathbf{v}_5$ with an angle difference of 3.17° are pointing almost in the same direction. Their component weights are very much in the same range and do not have the magnitude problem as encountered with vectors $\mathbf{v}_9$ and $\mathbf{v}_{13}$. As a result, the relative errors are acceptably distributed among the weight components involved.

Table 3.6 Vector component change impact analysis.

Reconstructed Vectors	Vector components			Relative Errors		
	w'_{11}	w'_{12}	w'_{1B}	$f(w_{11}, w'_{11})$	$f(w_{12}, w'_{12})$	$f(w_{1B}, w'_{1B})$
v'_{12}	-118.015	11.729	15.381	0.00%	-0.09%	0.01%
v'_{14}	-117.803	11.698	15.356	0.00%	0.09%	-0.01%
v'_{4}	-87.899	88.166	-41.140	-0.05%	0.02%	0.13%
v'_{10}	-89.905	90.178	-42.079	0.05%	-0.02%	-0.12%
v'_{9}	-52.933	1.947	39.088	-0.46%	499.39%	0.65%
v'_{17}	-59.170	2.176	43.693	0.29%	-5.93%	-0.51%
v'_{8}	-0.455	-6.343	-9.911	-13.93%	4.74%	-1.79%
v'_{5}	-0.471	-6.568	-10.264	27.05%	-3.74%	1.57%
v'_{2}	-33.822	24.667	40.552	-30.49%	-6.70%	49.63%
v'_{13}	53.095	-38.722	-63.660	-41.10%	1.66%	92.06%

So far each vector pair was positioned in the same quadrant because the signs for each component, which define the vector directions, are the same. Another indicator is the sign of the length adjustment factor F, which has been positive for all vector pairs except $\mathbf{v}_2$ and $\mathbf{v}_{13}$. Their angle difference is 171° without mapping, but if mapped from Q3 to Q1, the angle difference turns into 9°, only 1° below the acceptance angle threshold. Because of the increased angle, higher component as well as vector length errors are expected. Up to this point, the errors with regard to the reconstructed vector lengths have been very small, as shown in table 3.7. But without error induction, the generalisation error will remain unchanged. With the objective to improve on generalisation for the purpose of interpolation, errors will be introduced on vector removal to reduce the degree of freedom.

Table 3.7 Vector length change impact analysis.

Vector	Original length	Reconstructed length		Relative Error
v_{12}	119.589	119.589	($\|v_{rl}\|$)	0.00%
v_{14}	119.374	119.374	($\|v_{rl}\|$*F)	0.00%
v_4	131.118	131.118	($\|v_{rl}\|$)	0.00%
v_{10}	134.110	134.110	($\|v_{rl}\|$*F)	0.00%
v_9	65.851	65.830	($\|v_{rl}\|$)	-0.03%
v_{17}	103.321	73.586	($\|v_{rl}\|$*F)	0.00%
v_8	11.781	11.776	($\|v_{rl}\|$)	-0.04%
v_5	12.199	12.195	($\|v_{rl}\|$*F)	-0.03%
v_2	61.653	58.283	($\|v_{rl}\|$)	-5.47%
V_{13}	103.321	91.494	($\|v_{rl}\|$*F)	-11.45%

3.4.2 Analysis of Linked Network

Pruning has caused the removal of 5 hidden neurons and reduced the size of the hidden layer from initial 20 neurons to 15. This represents a network size reduction of 25%, with the objective of reduction of the generalisation error being successful.

In table 3.8 a comparison of the benchmarks for the trained and the linked networks is presented. To increase statistical reliability the entire experiment has been repeated 30 times and the averages, minimums and maximums for training and testing pattern are reported. It can be noted that the recall accuracy has suffered on average by 54.3%, while the generalisation error has improved an average of 12.6%.

Table 3.8 Comparison between trained and pruned network benchmarks for 30 runs.

Description	Trained Network				Pruned Network			
	Min	Max	Average	StDev	Min	Max	Average	StDev
SSE_{trn}	0.515	1.338	0.785	0.1992	0.709	2.894	1.172	0.6118
SSE_{gen}	0.750	0.963	0.852	0.0499	0.393	0.956	0.757	0.1699

Generally, if recall accuracy is more important than generalisation network pruning is not recommended. A graphical representation of the recall accuracy on the training data after linking is given in figure 3.11. Please note that all details given in this section are referring to the very first run from 30 repeated experiments.

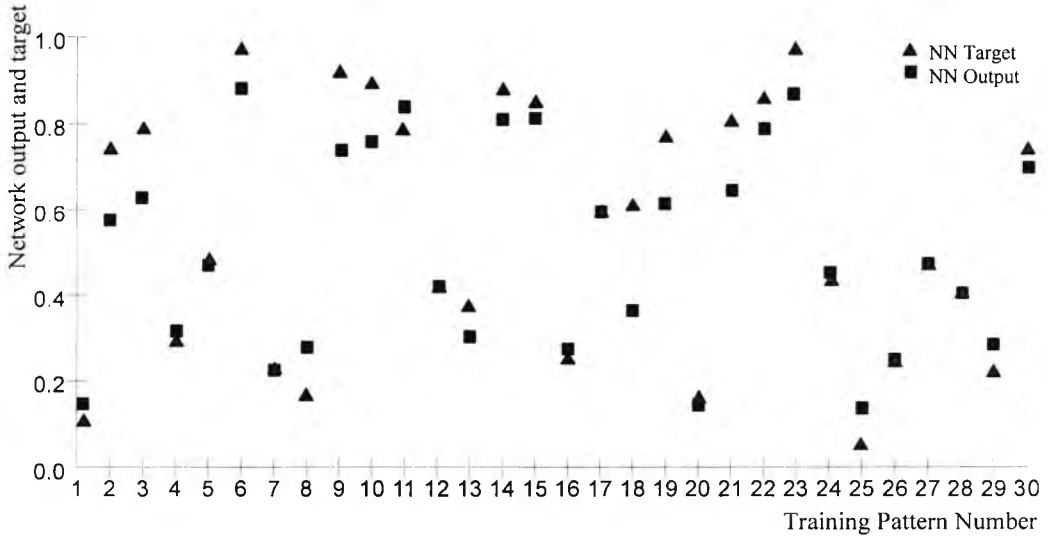


Figure 3.11 The objective function compared with the network output after linking.

Resuming the analysis of identification of the optimal acceptance angle for linking, an iterative approach has been employed where the acceptance φ has been increased in steps of 5° for the first run. During this process, the network has been linked for every iterative increase of the acceptance angle. To find the optimal acceptance angle for the objective to improve network generalisation, the error benchmarks SSE_{trn} and SSE_{gen} have been recorded. At the angle where generalisation reaches a minimum, the optimal point for linking is found.

Table 3.9 summarises which vectors satisfy the acceptance angle range limitations. To find the optimal acceptance angle φ_{opt} for generalisation, more and more vectors are included in the linking process.

Table 3.9 List of vectors satisfying the acceptance angle limitation

Acceptance angle	Linked vectors	SSE_{trn}	SSE_{gen}
0° - 5°	v12, 14; v4, v10; v9, v17; v5, v8.	0.31126	0.85867
5° - 10°	v2, v13[3].	0.52827	0.80907
10° - 15°	v7, v15; v3, v19[2].	0.86842	0.9278
15° - 20°	v11, v18[3].	0.92567	1.1628

[3] Including all of the above

Figure 3.12 shows the results from table 3.9 graphically. It has been found that acceptance angles above 15° usually worsen the generalisation and recall performances of a linked network. Figure 3.12 shows an increase of the recall and generalisation error if the acceptance angle φ is increased.

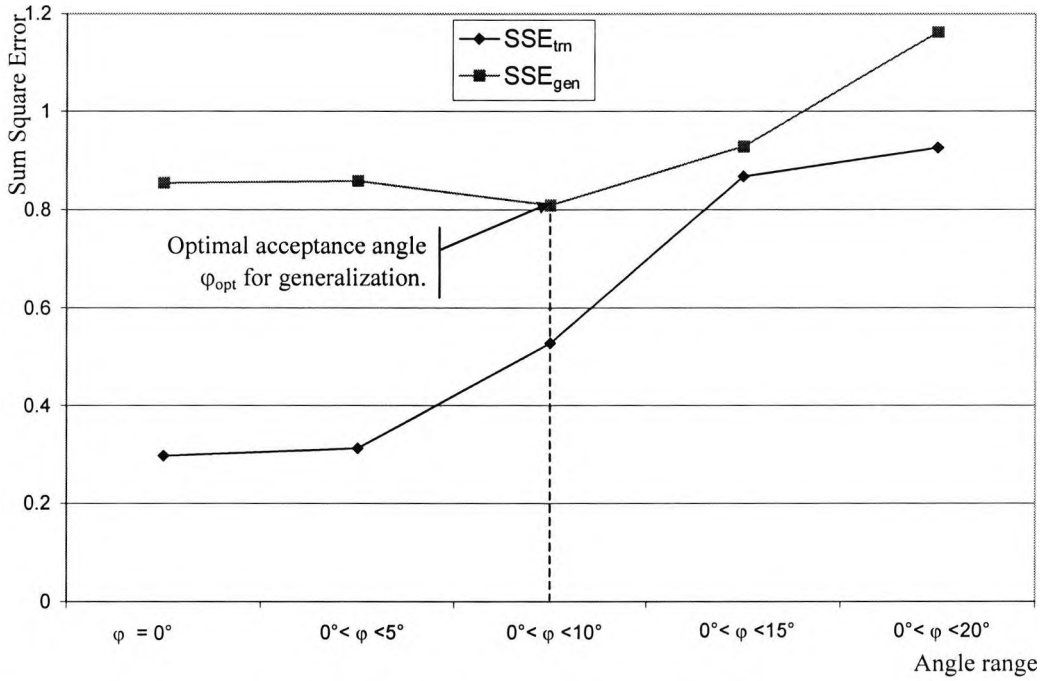


Figure 3.12 SSE_{trn} and SSE_{gen} as a function of the acceptance angle φ for finding φ_{opt}.

3.8 Conclusion

The linking process is the ideal candidate for the removal of vectors that are pointing in opposite directions because opposite vectors will only affect the sign of the length adjustment factor F. Compared to pruning methods which are only removing small weights or which attempt to cause only small changes in the overall recall error (small saliency) on neuron removal, the linking process is able to prune large weights which would have a substantial impact on the overall network error (large saliency) if removed without further consideration.

In the first example, two hidden neurons have been linked to create a single hidden neuron. For reasons of simplification, linear activation functions have been used whilst the linking process is analysed for its suitability of pruning hidden neurons. Linear activation functions within all neurons permits the combination of the weights of the hidden layer that is required for pruning an entire neuron.

In the second example non-linear activation functions have been used. To overcome the combination problem of the output neurons, all weights in the output layer have been set to 1 and frozen. Thus permitting the removal of hidden neurons via linking without disturbing the integrity of the output layer.

The sum square errors of the recall and generalisation accuracy have been recorded prior to and after pruning to perform as benchmarks for evaluation. As a result after 30 runs of each experiment, pruning has caused an increase in the error of the recall accuracy and a decrease of the generalisation error. This behaviour is typical for pruning since it reduces the network's degree of freedom that causes improvement on the generalisation but loss on the recall accuracy [142].

Many neural network variables such as the learning rate and momentum are dependent on the experience of the engineer responsible for training. The same applies for the acceptance angle φ unless an exhausting search for the optimal acceptance angle is performed as shown in figure 3.12. Experience of many linking applications for pruning by the author has shown that acceptable pruning results were still be obtained with an acceptance angle of 15° but angles above 20° resulted in intolerable loss of recall accuracy.

Chapter 4

The Stimuli Network

4.1 Introduction

A central area of study in psychology is how organisms change as a result of experience, that is, how they learn. Two major kinds of learning are usually distinguished: instrumental learning and classical conditioning [124, 143, 144].

In instrumental learning emphasis is placed on what kind of outcomes, reward or penalty, follows an action. This type of learning is referred to as reinforcement learning and is well known in the neural network research field [145]. In classical conditioning a conditioned event, the stimulus, can trigger an unconditioned response. Many researchers in the field of cognitive sciences have studied the stimulus-response theory, which originated from the field of psychology but its use is not widespread in the area of neural networks.

The human brain receives and interprets countless signals, called stimuli, which are generated from sensory parts of the body responding to the external environment. Physiologists and neurologists have mapped areas of the cerebral cortex and determined the purpose of each region as shown in figure 4.1. The crucial relay stations for incoming sensory signals can be found in parts of the brainstem. All sensory inputs to the brain connect to individual clusters of nerve cells, called nuclei,

which are stimulating groups of neurons in the cerebral cortex. Although the cortex is subdivided into distinct functional areas, fibres interconnect those clusters to share information [146].

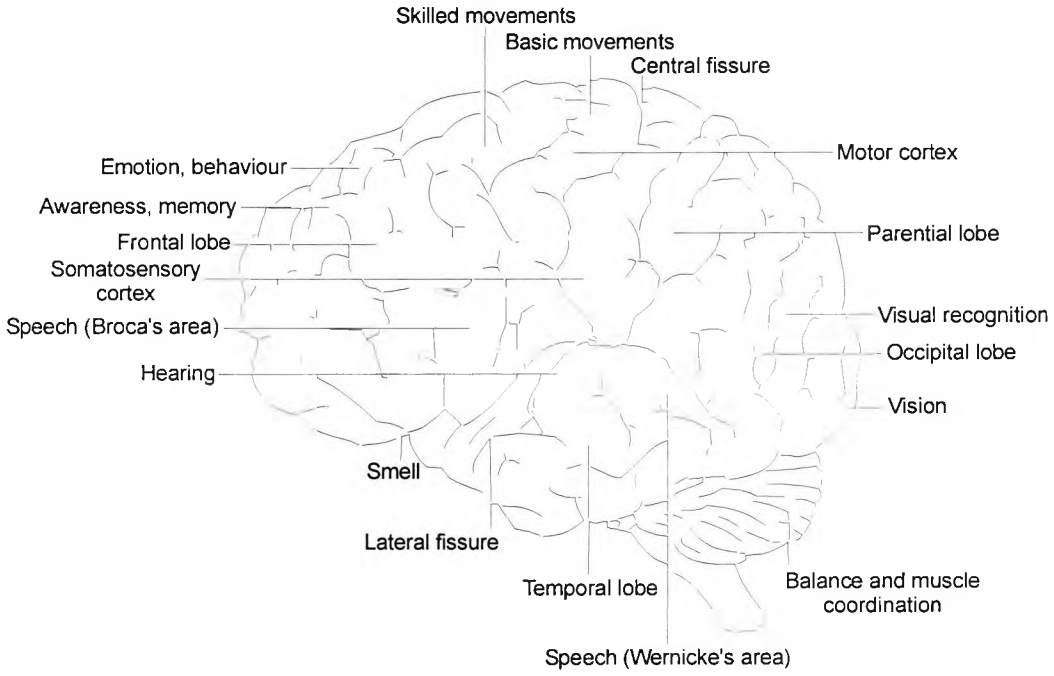


Figure 4.1 Functions of the Cerebral Cortex.

Knowledge held in the brain is accessed by the firing activity of ensembles of neurons. Present knowledge can be accessed if the right stimuli patterns are generated. Precise generation of stimuli in the brain is currently not fully understood but scientists can record and graphically present such firing patterns. The functional partitioning of the human brain has mainly been derived from analysing such stimuli patterns combined with studies of the preserved and impaired abilities in brain-damaged patients [147].

Reconstruction of sensory stimuli from observation of neural activity has been used to form an understanding of information processing in animals. Since visual stimuli can

be created under controlled conditions, they are used to analyse muscular responses of animals. For example, neurons from the visual systems from flies have been used to reconstruct stimuli for wing motion [148]. Visual stimuli have been reconstructed in a salamander retina from recorded responses of salamander cells [149]. Such studies are used to characterise neural dynamics in response to controlled stimuli.

In analogy to the brain, where specific tasks are assigned to distinct functional areas the linking process combines sub-networks by tagging their hidden neurons with a unique identifier, the stimulus code. Hidden neurons, of two or more knowledge domains, which have been involved in the vector linking process from chapter 2, will be tagged with more than one stimulus code, thus building the connections between the functional clusters, as presented in figure 4.2.

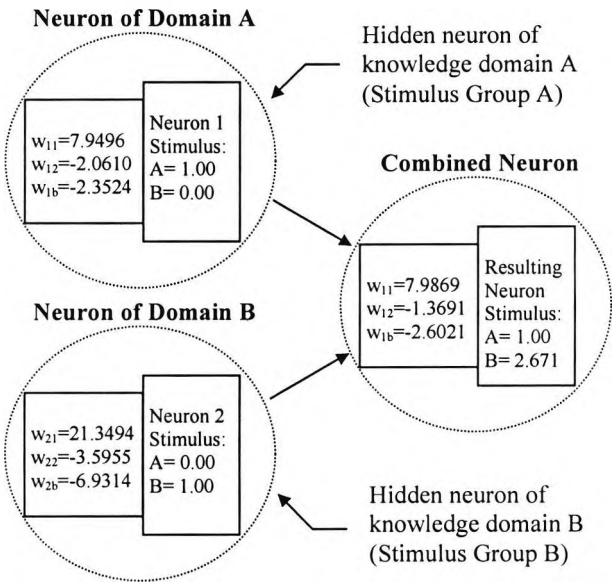


Figure 4.2 Hidden neurons before and after linking.

Different knowledge domains or functional clusters can be activated if a corresponding stimulus exists. Neuron activity between various knowledge domains

will change according to the stimuli they receive. Strong stimuli signals towards a knowledge domain resemble a strong membership of the input signal to the domain. But input signals can belong to more than one knowledge domain. In such cases, stimuli of different strengths for each domain can be generated reflecting their membership towards each domain.

There are many different ways of generating stimuli signals. Stimuli signals denote the membership of an input signal towards existing domains in a linked neural network. The generation of memberships is basically a classification task. Systems for classification include neural networks, Fuzzy Logic, statistical methods and expert systems [151, 152].

In this thesis, neural networks have been used for input signal classification and stimuli generation. The stimuli generated by such a neural network are directly induced into clusters of neurons instead of a combiner, as shown in figure 4.3. Such neural networks are in many ways similar to gating networks used in hierarchical expert systems but differ in one important aspect. They affect a neuron's output by changing its internal activation, thus changing the networks output on a basis of manipulating each individual neuron. Gating networks do not affect the overall output of a sub-network, they are used to control a linear or non-linear combiner to create an aggregated result, as shown in figure 4.4. For this reason, the networks used for stimuli generation are referred to as *stimuli networks* throughout this thesis.

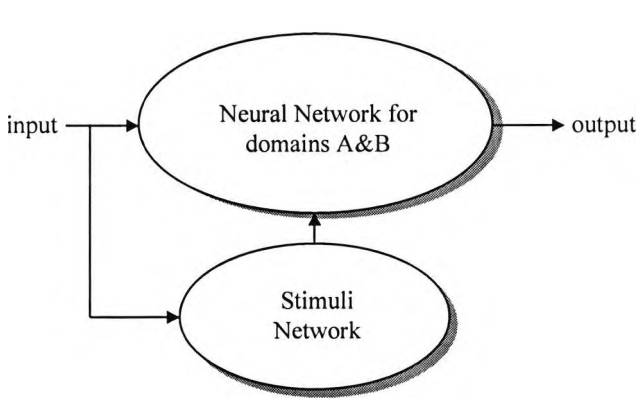


Figure 4.3 Stimuli networks induce stimuli information directly into neurons.

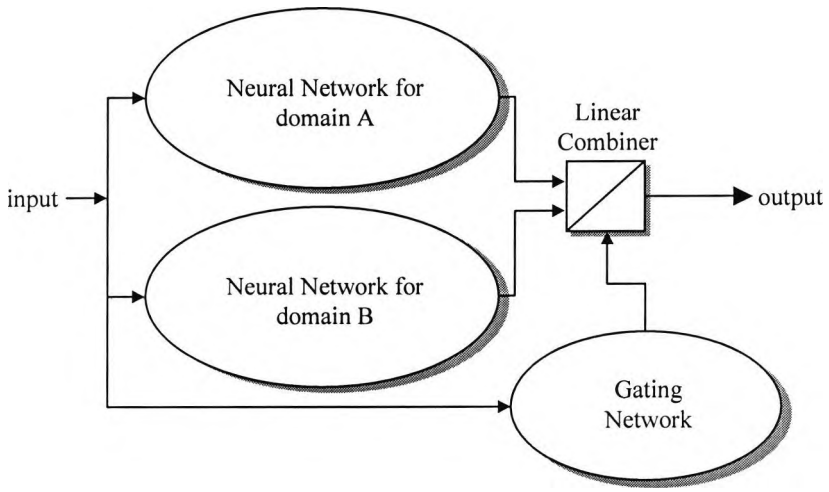


Figure 4.4 Gating networks regulate overall output via a combiner.

Stimuli networks can activate each knowledge domain or functional cluster, which is represented by a cluster of hidden neurons, by sending a stimulus identifier. They are generally designed to have outputs for each knowledge domain, which do not necessarily sum up to 1. If an input vector does not belong to any domain all outputs should be low. Otherwise if an input vector belongs to all domains all outputs should be high, activating neurons of all domains. Because of this, more than one output of the stimuli network can be high if the membership information request involves multiple domains to activate multiple clusters within the linked network.

Any type of expert systems, which has the capability to classify incoming data vectors into knowledge domains, can represent a stimuli network. The real distinction between stimuli networks and gating networks does not lay with the classifier itself or how it is trained. It lies with the points of induction where the classification results are used. Because linked neurons carry an internal length correction factor for different domains and consequently require integrated functionality for vector length corrections, this existing neuron functionality for the purpose of domain membership adjustments can be reused by utilisation of a stimuli network.

4.2 Stimuli Induction

Induction of the stimuli into neurons can be done in different ways. One way is to affect a neuron's net_j input, prior to the activation function. Another way is to alter a neuron's output after utilisation of the activation function. There are subtle differences at a neuron's output depending on the point of induction, as illustrated in figure 4.5.
It is important to take into consideration the distinction with respect to the neuron type required. There are different types of neurons available e.g. normal neurons, fuzzy neurons, GA neurons and linked neurons. Each neuron type can have different points of induction depending on its functional layout. As with linked neurons, stimuli information can be used to adjust the length of the weight vectors for each domain, as described later in this chapter.
One of the major differences between using a stimuli network and a gating network is that with a stimuli network domain memberships are depending on the activation function of the neurons involved. Whereby gating networks are situated at the overall network output and are not affected by the individual neurons configuration.

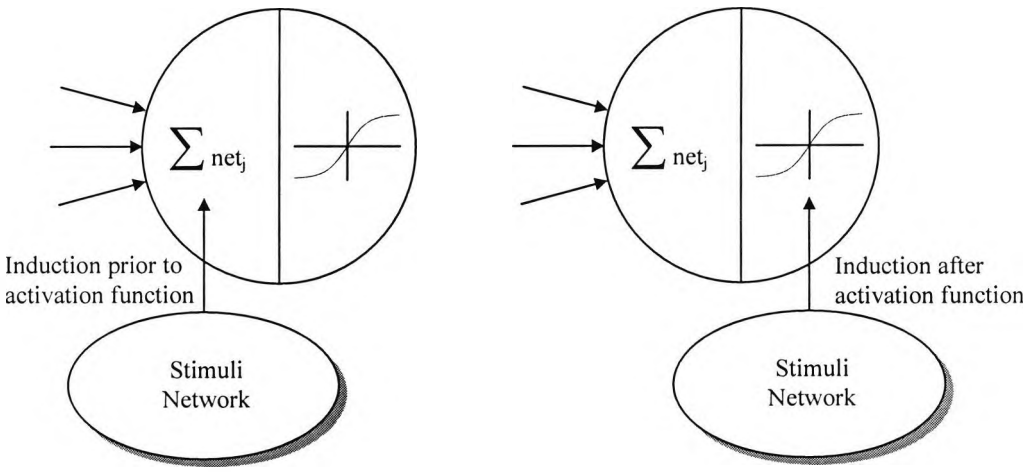


Figure 4.5 Stimuli network induction points prior and after the activation function.

The classification information induction point plays an important role in the neuron's membership function and its contribution to the overall network output. If the induction point is located prior to the activation function, it can be seen as a weight length correction on the current layer, as illustrated in section 2.6. If the induction point is after the activation function, it can be interpreted as a weight length correction on the next layer. Consequently, induction of stimuli information into a neuron can be interpreted as a means of adjusting the length of the weight vector of a neuron for different layers. A more detailed discussion can be found in sections 4.7.3 and 4.7.4.

4.3 Weight Vector Adjustment

The working principle of a stimuli network can be summarised verbally to explain its functionality. A stimuli network classifies the input vector $\mathbf{x}_{in}$ applied to the linked neural network to activate knowledge domains q with relevant information. Each identified knowledge domain q is receiving a domain stimulus factor S with a value in the range of 0 to 1. The value of the stimulus code S reflects the degree of membership of the input vector $\mathbf{x}_{in}$ to a specific knowledge domain q. If, for example, an input vector belongs 80% to domain A and 20% to domain B, the stimuli produced for domain A will be $S_1 = 0.8$ and $S_2 = 0.2$ for domain B. Depending on the design of the training data for the stimuli network, the sum of all domain memberships can be 1.0 but is not imperative. The higher the domain stimulus factor S is, the higher the contribution of that domain to the overall network output.

The net input calculation of a linked hidden neuron, net_j, differs from the normal backpropagation feed-forward calculation because the length correction factors F and the domain stimulus factors S need to be included. If two neurons of domain A and domain B are linked, they result in a linked neuron of domains A and B with factor 1 for domain A and F for domain B, as introduced in equation 2.50. With the consideration of linking q domains, each vector length adjustment factor F can be extended with indices starting from 1,...,q. Therefore, domain A will use the length

correction factor F_1 (=1), domain B will utilise F_2 and the last domain will utilise factor F_q, as illustrated in figure 4.6.

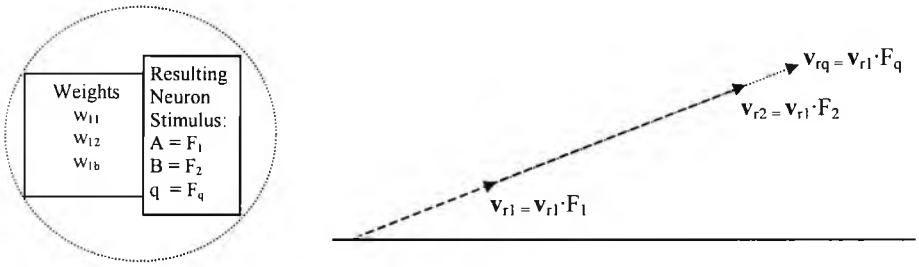


Figure 4.6 Vector length adjustments for recalling information for multiple domains.

The equations for the summed input of neurons A and B of domain A and B prior to linking are given in equation (4.1) and after linking in equation (4.2).

$$net_A = \sum_{n=1}^{i} w_{An} \cdot x_n \qquad net_B = \sum_{n=1}^{i} w_{Bn} \cdot x_n \tag{4.1}$$

$$net'_j = (1+F) \cdot \sum_{n=1}^{i} w'_{rn} \cdot x_n = (1+F) \cdot x_{in} \cdot v_{r1} \tag{4.2}$$

In the equations (4.1) and (4.2), A and B are representing two possible knowledge domains, i is the number of input neurons of the input layer, net'$_j$ is the summed input of the linked neuron and $\mathbf{w'}_m$ is the weight matrix of the linked neuron of domains A and B. The vector length adjustment factor is denoted with F and the input vector with $\mathbf{x}_{in}$.

On extending equation 3.12 from chapter 3 for the calculation of net'$_j$ of a linked neuron with the notation $F_1 \ldots F_q$ for q dimensions as shown in figure 4.6, equation (4.3) can be derived.

$$net'_j = \left(x_{in} \cdot v_{r1} \cdot F_1 + x_{in} \cdot v_{r1} \cdot F_2 + \ldots + x_{in} \cdot v_{r1} \cdot F_q\right) = x_{in} \cdot v_{r1} \cdot \sum_{d=1}^{q} F_d \tag{4.3}$$

In the equation above, $\mathbf{x}_{in}$ is the input vector, $\mathbf{v}_{rl}$ is the resulting weight vector of the linked neuron and $F_1,\ldots,F_q$ are the vector length adjustment factors for every domain involved.

Equation (4.3) has been used in chapter 3 for pruning of two neurons within only one domain. Because pruning involves only one domain, both reconstructed vector portions ($\mathbf{v}_{rl}$ and $\mathbf{v}_{rl} \cdot F$) in equation 3.12 can be summed up without constraints.

Because linking of two or more domains is using a stimulus to determine the domain membership, an additional factor called a domain stimulus factor S has been introduced. For q domains held in a linked network, q stimulus factors S are produced for the identification of the membership from each input vector $\mathbf{x}_{in}$ to every domain. The stimulus factor S is created by the stimuli network and will regulate the contribution of each linked domain towards the output. Equation (4.4) is analogous to equation (4.3) but takes the stimulus factor S for every domain q into account.

$$net'_j = \boldsymbol{x}_{in} \cdot \boldsymbol{w}_j = \boldsymbol{x}_{in} \cdot \boldsymbol{v}_{rl} \cdot \sum_{d=1}^{q} (S_d \cdot F_d) \tag{4.4}$$

In the equation above, S_d is the input vector membership and F_d the vector length adjustment factor for every domain q involved in the network linking process.

To summarise, each linked neuron r contains a resulting weight vector $\mathbf{v}_{rl}$ and a domain map with vector length adjustment factors F_d for each domain. On data recall, the stimuli network generates domain membership factors S_d for each domain by utilising the input vector $\mathbf{x}_{in}$. To calculate the summed input to the linked neuron net'_j, the input vector is applied to the weight matrix and multiplied with the sum of all domain stimuli multiplied by their vector length adjustments as given in equation (4.4). With regard to figure 4.5, equation (4.4) represents stimuli induction prior to the activation function.

If all neuron activation functions are linear the vector length correction as described will be equivalent to the usage of a gating network with a linear combiner. But with non-linear neuron activation functions, domain membership information has to pass through the neuron activation function, which will affect the output of the neuron in a

non-linear manner. Because of this, even a low membership factor multiplied with a large weight can create a large neuron output on sensitive neurons.

4.4 Input Neuron Sensitivity

Sensitivity analysis describes methods that produce a measure to determine the functional contribution of inputs to outputs. It is a simple method of finding the effect an input has on the output of the network. The relationship of an input neuron to an output neuron is found by determining the impact that a small change in the input has on the output. If a drastic change occurs at the output, the input is considered to be one of the key factors in producing the current activation value of the output and therefore reasonably high in sensitivity [60-62].

Furthermore, inputs with minor but unique information can be more significant than inputs with higher magnitude but redundant information. Inputs with unique information are more likely to have a large effect on the overall network output and can be defined as being reasonably high in sensitivity. Redundant input neurons are likely to represent a low sensitivity to the output and can be subject to pruning [119].

The brute-force sensitivity analysis is largely concerned about reducing the number of input neurons in a black-box approach. It changes input vectors within a set range to determine the effect on the output only. Other mathematical approaches on the weight matrixes try to identify sensitive inputs for elimination [139, 153].

Neuron sensitivity is an important factor if stimuli induction methods are used to control individual neuron contribution. Sensitive neurons are more dynamic than less sensitive neurons if the length of a weight vector changes. Because of this, non-sensitive neurons represent neurons with an almost constant output and are less affected by incoming stimuli. Non-sensitive neurons can still generate a dominant contribution to the overall network output, even if the stimulus for the domain they belong to is small. This can be a problem since unwanted domain contributions from non-sensitive neurons will reduce the accuracy of domains of interest.

4.5 Hidden Neuron Sensitivity

In order to control neuron outputs via the stimuli method, the summed input net_j and the activation function need to be considered as important factors. If a domain membership expressed as S_d in (4.4) is 0% the neuron contribution to the overall network output should be 0. This requires a non-linear neuron activation function that will go through the point of origin (0,0) for non-linear objective functions. There are several non-linear activation functions available that fulfil this requirement. Because of its widespread use, the symmetrical-sigmoid function has been chosen as the activation function throughout this thesis, as given in equation (4.5) and illustrated in figure 4.7.

$$f(x) = \frac{2}{1 + e^{-(net_j)}} - 1 \tag{4.5}$$

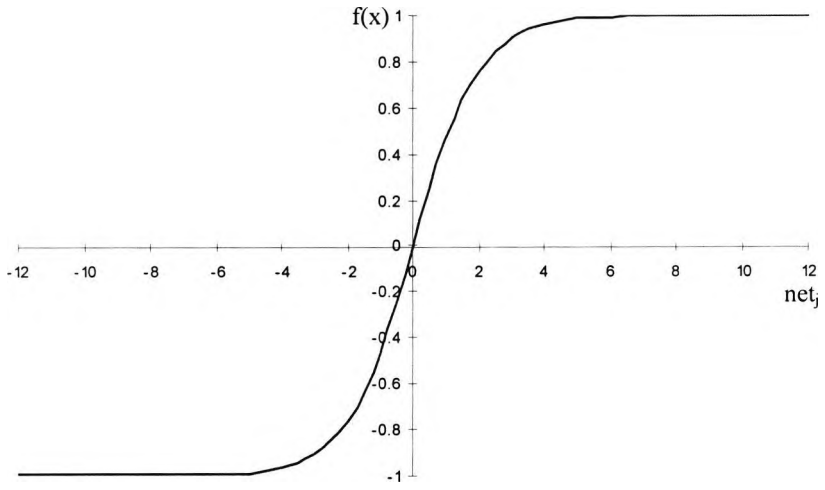


Figure 4.7 Symmetric-sigmoid activation function.

The summed neuron input net_j plays an important role if the domain membership stimulus is applied prior to the application of the neuron activation function. If, for instance, net_j is large e.g. net_j=12, a change in domain membership from S=90% to S=30% will change the neurons output from out_j=0.999 to out_j=0.946 Δ=5.32%.

Consequently, large neuron inputs will not follow the domain membership in a linear fashion as experienced with linear combiners used with gating networks. If on the other hand the neuron input is relative small e.g. net_j=1.2, a change in domain membership from S=90% to S=30% will change the neurons output from out_j=0.493 to out_j=0.178 Δ=63.88%.

It can be said that with stimuli induction the neuron output is not only dependent on the domain membership but also on the sum of all weight connections contributing to the neuron's input net_j. Equation (4.6) and figure 4.8 and represent the neuron output as a function of the neuron's summed input net_j and its domain membership S. Note that domain memberships S can be expressed as a negative to reduce the output value.

$$f(x) = \frac{2}{1 + e^{-(net_j \cdot S)}} - 1 \tag{4.6}$$

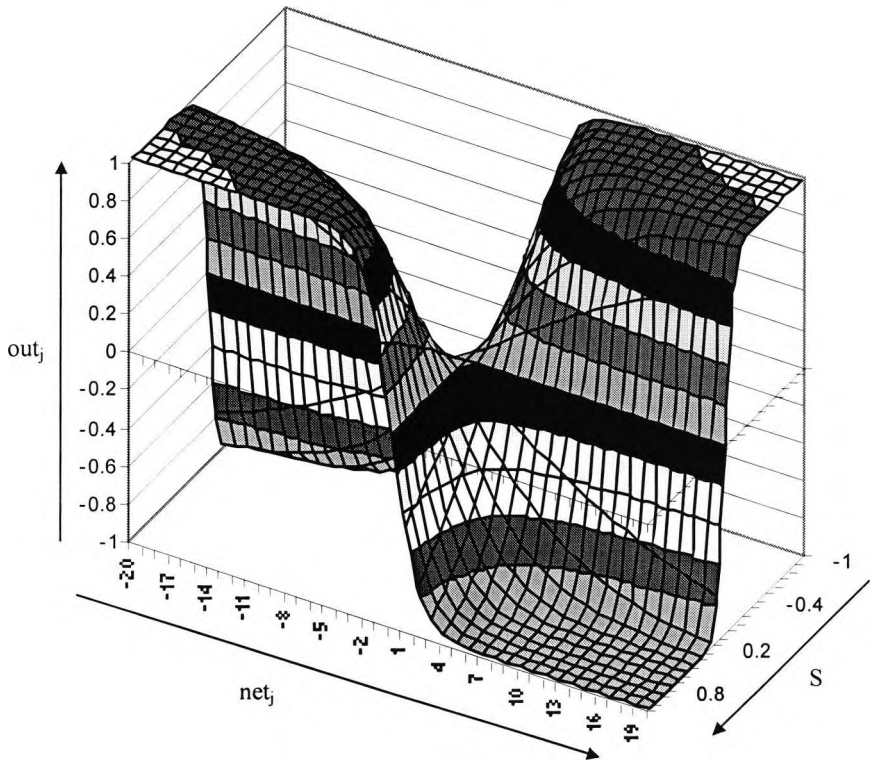


Figure 4.8 Neuron output out_j as a function of net_j and domain membership S.

Since the neuron output out_j is depending on the summed input net_j and the membership factor S, the higher net_j is, the higher the non-linearity of the domain membership factor S, as shown in figure 4.9. If the training data is normalised, for example ±1 for a symmetrical-sigmoid activation function, large values of net_j for a neuron of a hidden layer can only be computed if large weights or many connections are present. This form of input saturation can be found in over-trained networks or networks with many inputs.

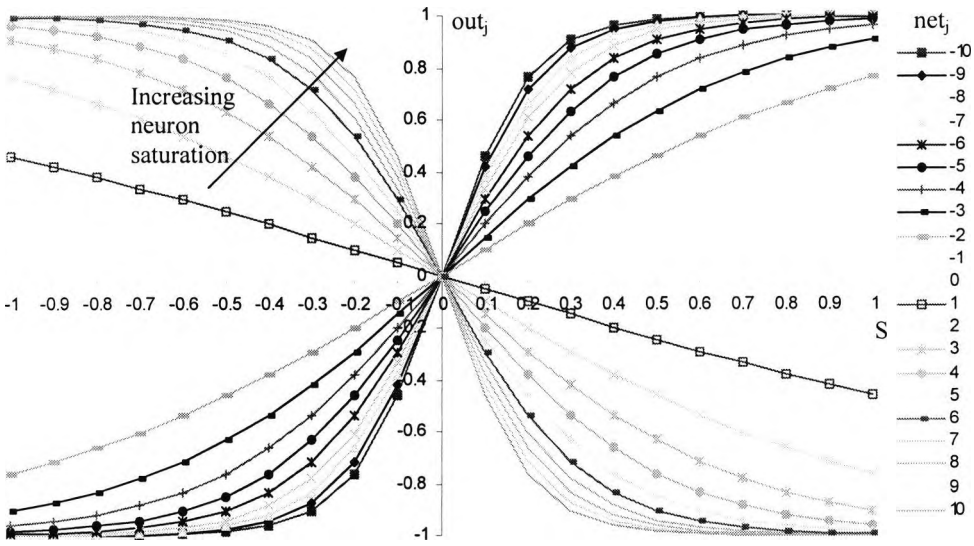


Figure 4.9 Neuron output out_j as a function of parameterised net_j and domain membership S.

To avoid neuron saturation caused by large weights, penalty terms can be included to reduce weights during training. To avoid neuron saturation caused by many incoming connections, the learning rate must initially be low and can be increased or further decreased during learning [155]. Dynamic parameter adjustments, such as weight decay, simulated annealing or pruning, are generally associated with network convergence and/or stability issues. But these are not the only problems, which can be resolved or prevented by implementing extended training algorithms. The next sections will illustrate the problem with reference to domain memberships if the neuron saturation problem is ignored.

4.6 Neuron Saturation Analysis

Applying increasing input values on the neuron and monitoring the overall network output can measure a neuron's sensitivity. Generally, it can be said that the higher the weights attached to a neuron are, the higher its sensitivity. A neuron with a very high sensitivity is most likely to suffer from saturation problems, which cause its output to remain on activation function extremes e.g. ±1. If a neuron has a very high sensitivity it can be stated that it contains large weight connections and therefore it has a higher steepness towards its domain membership, as shown in figure 4.9. That means sensitive neurons will contribute towards the overall network output even if their domain membership is very low.

Linear domain contributions on non-linear activation functions can only be archived by utilisation of a gating network in conjunction with a linear combiner. Stimuli induction points are dependent on the neuron's activation function and can only result in a linear membership contribution if the neuron's activation function is linear or nearly linear. But training a neural network with a non-linear objective function requires a non-linear activation function to achieve acceptable convergence and recall errors.

To identify a neuron that can be declared as being saturated, the author has made the following definition:

A neuron is saturated for a domain membership S between ±0.8 if the area below its parameterised activation function is equal or less than the area of an equivalent linear function.

In a graphical context, neurons with a net_j equal or below of approximately ±2.3 are outside the saturation range for a standard symmetrical-sigmoid function. This value has been determined graphically by means of visual analysis of figure 4.10.

The graphical approach to find net_j to avoid neuron saturation, involves a parameterised neuron activation function net_j and a reference linear function. In figure 4.10, several parameterised neuron output functions for different net_j are printed with the domain membership S on the x-axis. For net_j=2.3, the area below the curve is almost equal to the area below the linear function. An exact figure can be obtained by definite integration of equation (4.6) between 0 and 0.8.

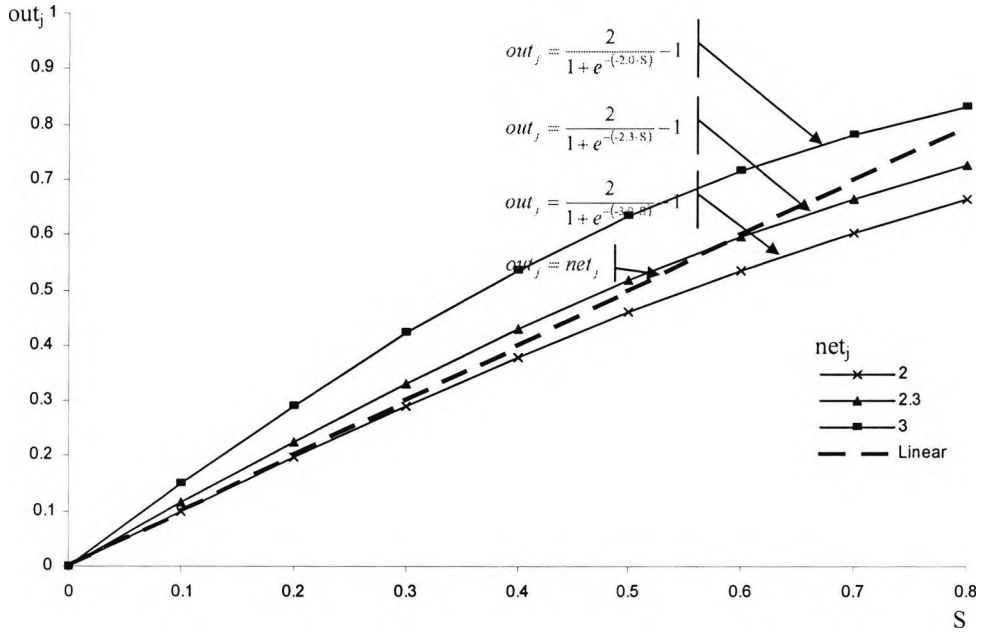


Figure 4.10 Neuron output out_j for to define point of saturation.

Determine net_j, where the area below the curves of $f_{(S)}$ and $f_{(x)}$ are equal.

$$with: f_{(S)} = \frac{2}{1+e^{-(net_j \cdot S)}} - 1 \quad \text{and} \quad f_{(x)} = x \tag{4.7}$$

$$Area = \int_0^{0.8} \frac{2}{1+e^{-(net_j \cdot S)}} - 1 \cdot dS = \int_0^{0.8} x \cdot dx = \frac{0.8^2}{2} = 0.32 \tag{4.8}$$

The solution of (4.8) has been obtained with help of figure 4.10 and is:

$$Area(S, net_j) = \left[S + \left(\ln\left(\frac{1}{net_j} + e^{-net_j} \right) \cdot \left(1 - e^{-S \cdot net_j} \right) \right) \right]_0^{0.8} = 0.32 \qquad (4.9)$$

With S = 0 and net_j = 0, Area = 0, leaving the calculation of net_j if S=0.8. Because of the complexity of the equation, equation (4.9) has been plotted and an iterative approach has been chosen to find a solution for net_j with S = 0.8 and Area = 0.32. The iterative process resulted in net_j = 2.25565.

With a definition of a neuron saturation point and a definite figure for net_j, in conjunction with the symmetric-sigmoid activation function, a more precise penalty term based not on the individual weights but based on the actual neuron input net_j can be derived. This penalty term, applied to a learning parameter i.e. learning rate, should prevent net_j exceeding 2.25565 by manipulation of weight updates.

Most weight decay regulators depend only on the network parameters such as weights and number of connections. They are generally avoiding that any single weight will not exceed a certain determined value and do not take the entire neuron input net_j into account.

Supervised backpropagation with batch weight updates generally uses a combination of training factor and momentum to avoid oscillation, given in equation (4.11). There are algorithms such as simulated annealing and weight decay available to avoid large weights but most algorithms do not take the total neuron activation into account.

Equations (4.10) to (4.14) show how the neuron input $net_j(t+1)$ is affected by a weight update Δw_j compared to its previous input $net_j(t)$. It can be seen from equation (4.14) that if $net_{j\,max}(t)$ and the input vector **x** for the largest activation are known, the neuron input change Δnet_j, which would arise after the weight update, can be computed. With the information on what the largest neuron activation will be prior to updating of the neuron's weights, a penalty term can be applied to keep the neuron away from saturation.

Consequently, neuron saturation can be avoided if the maximum neuron saturation $net_{j\,max}$ and the sum of its associated input vector are stored for each neuron after completion of an entire training epoch. After completion of the epoch, this stored information on a per neuron basis can be used to determine the penalty term χ to be applied on the weight update prior to updating of the weights. This process is graphically presented in figure 4.11.

$$net_j(t+1) = net_j(t) + \Delta net_j(t+1) \tag{4.10}$$

$$net_j(t) = \sum_{n=1}^{i} w_{jn}(t) \cdot x_n \tag{4.11}$$

$$net_j(t+1) = \sum_{n=1}^{i} \left(w_{jn}(t) + \Delta w_j(t+1)\right) \cdot x_n \tag{4.12}$$

$$net_j(t+1) = \sum_{n=1}^{i} w_{jn}(t) \cdot x_n + \sum_{n=1}^{i} \Delta w_j(t+1) \cdot x_n \tag{4.13}$$

$$\Delta net_j(t+1) = \Delta w(t+1) \cdot \sum_{n=1}^{i} x_n \tag{4.14}$$

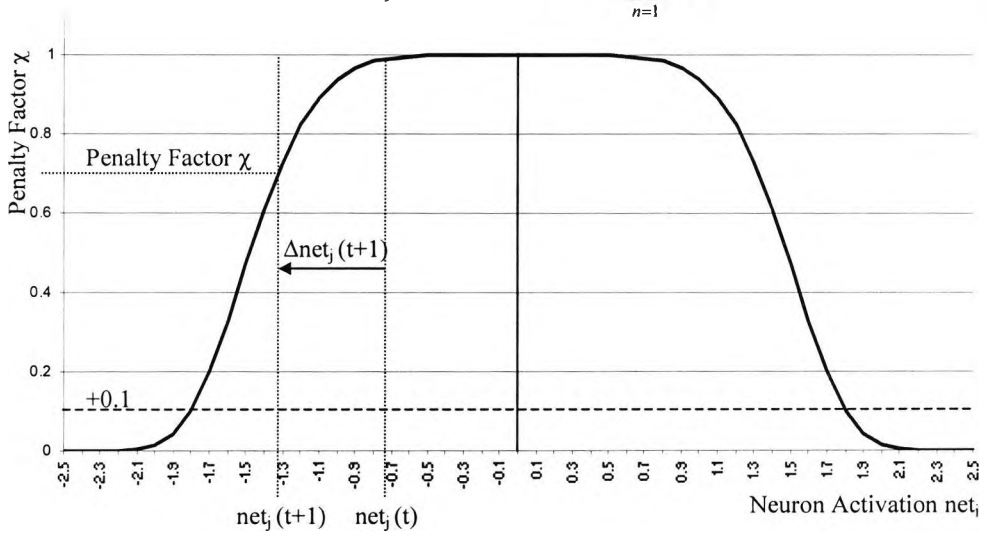


Figure 4.11 Penalty term to adjust the learning factor during training to avoid neuron saturation.

A constant of 0.1 has been added to the penalty function. Without this constant the weight update can reach 0 and the training will stagnate.

The penalty function shown in figure 4.11 has been developed with the requirement that the penalty factor χ should be 1.0, leaving the backpropagation algorithm unchanged for small neuron activations. For large neuron activations, the penalty factor χ should reduce to lower weight updates determined by the batch backpropagation algorithm. For large input activations close to 2.25 the penalty factor χ should be close to 0.0001, avoiding a further increase of the activation. With all these considerations, equation (4.15) has been developed to represent a suitable penalty function.

$$\chi = e^{-\left(\frac{net_{j\max}^{6}}{\varepsilon - |net_{j\max}|}\right)} + 0.1 \tag{4.15}$$

With ε as the function form factor, where $f(net_j) = 0.0001$ for $net_j = 2.25$, which can be calculated as shown in (4.16).

$$\varepsilon = -\left(\frac{2.25^6}{\ln(0.0001)} - 2.25\right) = 16.33703 \tag{4.16}$$

After calculation of the penalty term, neuron weights can be updated with the modified update formula as shown in equation (4.17).

$$\Delta w(n+1) = \chi\left(\eta(\delta_j o_i) + \alpha \Delta w(n)\right) \tag{4.17}$$

With the introduction of the penalty term, neurons stay dynamic and are able to respond to different levels of domain membership. It should be noted that neuron saturation could be prevented by an appropriate input vector normalisation.

This normalisation should not just be restricted to its dynamic range e.g. ±0.9 or ±0.8, but also the removal of any constants, which can be determined by calculation of the average mean. With the subtraction of the mean and division of all input vectors by the largest number from all input vectors in the training set (scaling factor), network dynamics can be greatly improved.

On denormalisation, the mean and the scaling factor can be reapplied to the network output. With the penalty function from figure 4.11, the neurons cannot saturate at their input activation and therefore stay more dynamic than neurons with a saturated input.

4.7 Numerical Experiment: Linking Saturated and Unsaturated Networks

To highlight the effects and the associated problems with neuron saturation, a numerical example, based on a mathematical problem, has been carried out. Two networks with frozen weights on the output layer have been trained with 400 data points from two different domains A and B. Each of the networks was trained with data of one domain only. Their topologies are two input neurons, two hidden neurons and one output neuron to form a fully connected 2:2:1 backpropagation network. After training, a comparison was made between combining both networks with a gating network and with a stimuli network. The purpose of the comparison is to analyse differences between the linearly combined output of a gating network with the output of a set of linked networks controlled by a stimuli network with unsaturated and saturated neurons.

Because neuron saturation problems are examined, comparisons with linearly combined networks with gate and linked networks with stimuli have been performed each with unsaturated and saturated neurons. Unsaturated neurons can be found in networks trained with the penalty function and saturated neurons in networks without penalty function, as discussed in section 4.6.

To create training data of different domains, the input space has been divided into two sections, one for domain A and one for domain B as shown in figure 4.12. The network trained with data of domain A is limited to the input space of domain A and will be referred to as network A. The network trained with data of domain B is limited to the input space of domain B and will be referred to as network B. With the separation of the input space into A and B it is possible to analyse the impact of

different kinds of assembly methods (linearly combined or linked) of networks trained with different input spaces.

The input space division into domains A and B is illustrated in figure 4.12, where their corresponding domain memberships are shown as triangular markers (▲) and square markers (■) for domains A and B respectively. All input vectors $\mathbf{v}_A = [x_1\ x_2]$ belonging to domain A are located in the top left hand corner and input vectors for domain B $\mathbf{v}_B = [x_1\ x_2]$ are located in the bottom right hand corner.

The dividing line in figure 4.12, which separates domain A from B, does not represent the actual domain partitioning. This line divides the training data sets into two files used for training. Because the training data used resembles two domains, it has been physically split into two files, allowing individual network retraining in cases of data changes.

Furthermore, for the purpose of investigation of neuron saturation, it is necessary to explore the whole input space range of ±1, because the activation function used for each neuron is the symmetric sigmoid function.

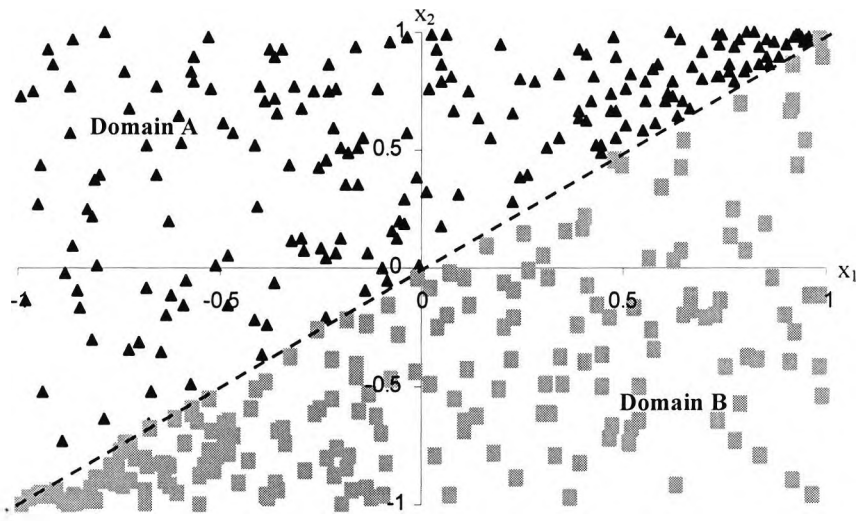


Figure 4.12 Training data for domain A (▲) and B (■).

4.7.1 Domain Memberships

In this thesis, a single network trained to perform on one partition of the input space only, represents a domain. The domain membership to a domain is defined as the weighting factor associated to an input vector to limit its contribution to the overall network output. The domain transition is a function of how domain memberships change, plotted over the entire input space.

Although the data has been split into two domains for network training, the domain transition chosen for the linear combiner used in this experiment is not abrupt. Instead each domain transition function gradually changes from domain A to domain B and vice versa on a linear scale as shown in figure 4.13.

For example, the further the input vector coordinates x_1 and x_2 located in domain A move towards domain B, the lower the domain membership of A and the higher the domain membership of B. Let point $p_1(0.45, 0.35)$ have a domain membership of 0.3 of A and 0.7 of B, if moved towards domain B to become point p_2, then point $p_2(0.5, 0.35)$ has a domain membership of 0.2 to A and 0.8 to B. For any point in this example, the sum of both domain memberships is 1.

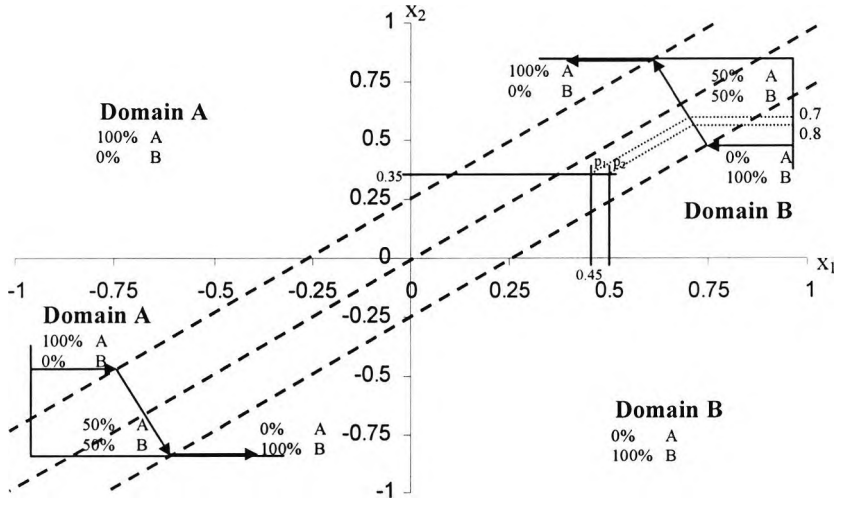


Figure 4.13 Input vector domain membership distribution of domains A and B

To present more in depth information about the membership function shown in figure 4.13, the function has been pseudo programmatically expressed with the conditions outlined in equation (4.18).

Membership of domain A and B as a function of x_1 and x_2: (4.18)

```
if ((x2-x1)>0.25){
   Membership A = 1
   Membership B = 0
} else if ((x2-x1)<-0.25) ){
   Membership A = 0
   Membership B = 1
} else {
   Membership A = 0.5+(0.5*(x2-x1)/0.25)
   Membership B = 1-A
}
```

The objective of this linear transition between domains is to emphasize the impact a linear combiner has on the combined output of two already trained networks. Thus permitting comparison between the outputs of a linearly combined and a linked set of networks, without taking generalisation or recall accuracy into account. For this reason, all 400 data points have been used for training and have not been split into training and testing data sets.

4.7.2 Network Topologies

To keep all detailed calculations small and manageable, uncomplicated networks have been used to concentrate on the gating and linking processes. Thus, for the training of each domain, simple 2:2:1 backpropagation networks with frozen weights set to 1.0 at the output layer and symmetrical sigmoid activation function (see figure 4.7) have been used, as illustrated in figure 4.14. The only purpose of the networks is to supply weight matrixes with unsaturated and saturated neurons so that linear combination using a gating network can be compared with linking using a stimulus network.

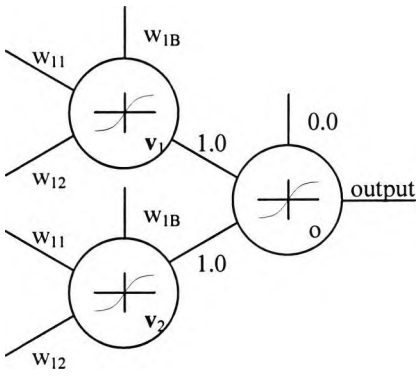


Figure 4.14 Network topology with its frozen output layer.

The network training parameters applied for all networks are presented in table 4.1. Note that the weight initialisation is set to ±2.6, which is approximately the maximum net_j for a single neuron. This rather large starting point increases the probability of large weights for the creation of saturated neurons if no restrictive measures are taken during training.

To avoid neuron saturation for the first set of weight matrixes, the penalty function from section 4.6 has been exercised during training. The resulting weight matrixes after training of both networks for domain A and B with unsaturated neurons are shown in tables 4.2 and 4.3.

Table 4.1 The parameters of the neural networks used in this section.

Description	Value
Input Neurons	2
Hidden Neurons	2
Output neurons	1
Activation Function	symmetric sigmoid
Initialisation	±2.6
Learning Factor	0.1
Momentum	0.5
Number of training patterns	400

Table 4.2 Weight vectors of the hidden layer of domain A trained with penalty function.

Vector Reference	w_{11}	w_{12}	w_{1B}	Vector Length
v_1	-0.201919	0.103184	-0.68289	0.7195
v_2	-0.62344	-0.364037	-1.166956	1.3722

Table 4.3 Weight vectors of the hidden layer of domain B trained with penalty function.

Vector Reference	w_{11}	w_{12}	w_{1B}	Vector Length
v_3	-0.195099	-0.191644	-0.372048	0.4617
v_4	0.306013	-0.498975	1.265225	1.3941

To create saturated neurons, in the second set of weight matrixes, the penalty function from section 4.6 has not been exercised during training. Therefore, the resulting weight matrixes after training of both networks for domains A and B with saturated neurons are much higher in value and are shown in tables 4.4 and 4.5.

Table 4.4 Weight vectors of the hidden layer of domain A trained without penalty term.

Vector Reference	w_{11}	w_{12}	w_{1B}	Vector Length
v_5	-6.6214	9.5011	-1.6889	11.7032
v_6	7.9496	-2.061	-2.3524	8.5427

Table 4.5 Weight vectors of the hidden layer of domain B trained without penalty term.

Vector Reference	w_{11}	w_{12}	w_{1B}	Vector Length
v_7	21.349	-3.595	-6.9314	22.7325
v_8	2.2002	4.0911	-3.1837	5.6315

4.7.3 Non-Saturated Neurons

Neuron saturation in a neural network reduces the individual neurons dynamic characteristics because saturated neurons supply the overall network output with a nearly constant contribution. Such constant contributions can be found on over-trained networks, which have characteristics of erratic behaviour if unseen input vectors for generalisation or more precisely for inter or extrapolation are presented.
The following analysis has been comparing how the overall output of combined networks with unsaturated neurons changes if saturated neurons are used. The output for this comparison has been produced with linearly combined and linked networks. For the purpose of comparing output changes, the individual network outputs are plotted with their corresponding input space coordinates in figure 4.15. This graph has been used as a reference for comparing combined network outputs created by a gating network and a stimuli network for saturated and unsaturated neurons.

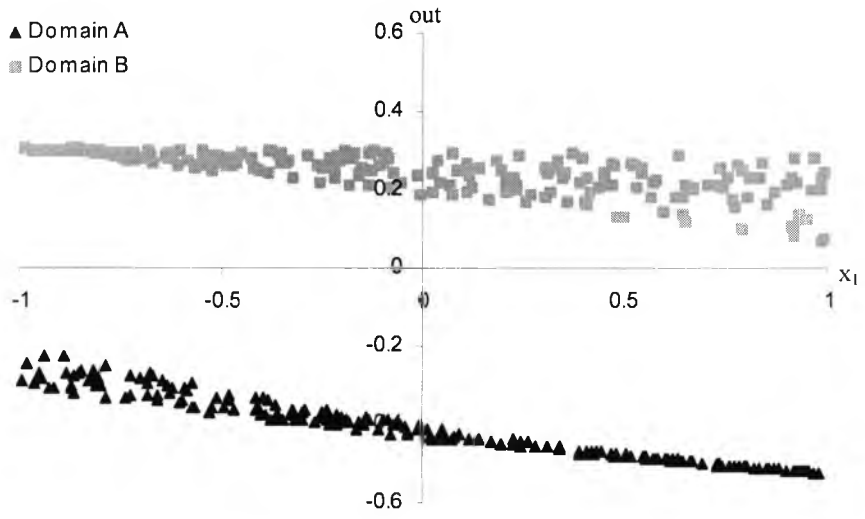


Figure 4.15 Network output for domain A (▲) and B (■) with small weights.

Besides highlighting the problems of neuron saturation, changes at the output of combined networks will determine the suitability of a specific combination method.
Figure 4.15 shows the problem of domain separation, where each domain only responds within the boundaries of its own training data. Such separation issues cannot

be avoided in cases where data is received from distinct sources, operating in different input and output spaces. With the help of network combination methods, individual network responses can be combined where contributions of both domains are valuable. This is generally the case where clear domain boundaries in the input space exist or input vectors are reasonably close to several domains.

In this numerical example, such an area, where both domain responses are important, has been introduced in section 4.7.1 and figure 4.13.

4.7.3.1 Linear Combined Output

A linear combined network assembly will apply the received input vector $[x_1\ x_2]$ to all network inputs in the assembly and additionally to the gating network, which is regulating the linear combiner as shown in figure 4.4. The network outputs are then weighted for each domain as graphically illustrated in figure 4.13 and summed to construct the combined network output.

For reasons of simplification the domain transition function shown in figure 4.13 is representing the gating network and the linear combiner as one unit. This eliminates the need for training of a domain classifier, thus permitting well-defined domain weighting for the network combination analysis.

An extract of five data points from the training set is shown in table 4.6. It includes domain memberships calculated with equation (4.18), individual network outputs and the linearly combined output.

Table 4.6 Calculation of the linearly combined network output for a selection of 5 data points.

Input x_1	Input x_2	Membership domain A	Membership domain B	NN output domain A	NN output domain B	Combined output
-0.21668	0.59406	1	0	-0.39488	0.10198	-0.39486
0.244674	0.38550	0.7816	0.21833	-0.44466	0.13725	-0.31761
-0.08546	-0.05403	0.56286	0.43713	-0.38924	0.19223	-0.13506
-0.87317	-0.99972	0.24689	0.75310	-0.20905	0.30459	0.177781
0.73945	-0.64336	0	1	-0.47138	0.25678	0.256780

The coordinates x_1 and x_2 of the input vector in the first row of table 4.6 are locating it 100% in domain A after superimposing x_1 and x_2 on figure 4.13. This results that the sole contributor is the network of domain A. The second data point belongs 78% to domain A and 22% to domain B, causing both domain networks to contribute to the overall network output, with all other data points repeating the same calculations.

The first, second and third data points, if superimposed on figure 4.12, were exclusively present in the training data of domain A. They were not included in the training data of domain B. Because domain B is contributing to the combined output for points two and three, data points which were previously bound to their domain spaces are now shifted towards other contributing domains. This shifting effect can be observed in figure 4.16 where portions of domain B are added to domain A for input vectors that are close to the domain boundary.

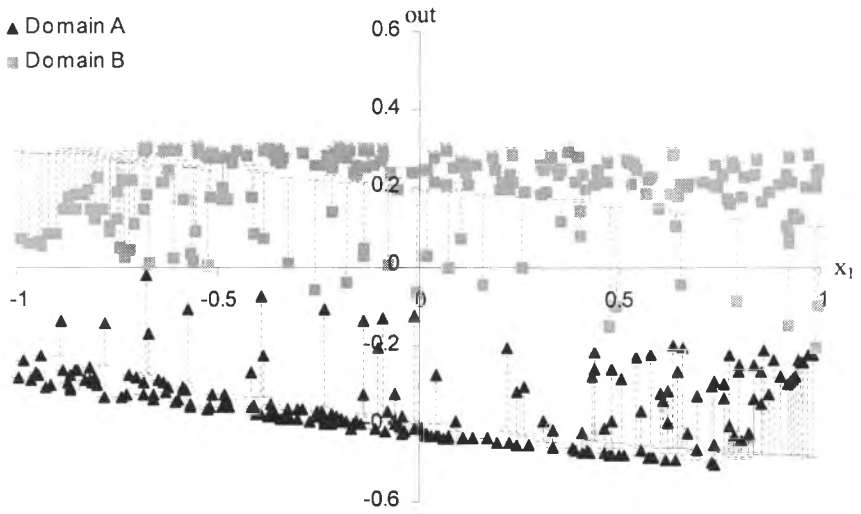


Figure 4.16 Linear combined output for domain A (▲) and B (■) with small weights.

The clear domain separation present in figure 4.15 has changed to a less separated output as shown in figure 4.16. It shows the effects of linear combination of two domains and as far as combination of knowledge is concerned, the effects of domain B on domain A and vice versa are clearly visible by the dotted error bars. The error

bars are giving an indication of how much and into which direction a data point has moved. The linked output from figure 4.16 has been compared against the individual network output from figure 4.15 and the SSE calculated as presented in table 4.7.

Table 4.7 SSE of individual domains.

Domain	Sum Square Error (SSE)
A	1.6560
B	1.4840

4.7.3.2 Linked Output

Equally to the gating network in the linearly combined output, the stimuli network for the linked output represents a classification network utilising the same classification function shown in figure 4.13 and equation (4.18). The most important distinction between the stimuli network and the gating network is, that the stimuli network controls individual neuron outputs instead of entire network outputs, as it is the case with gating networks.

For linking of neurons, vector angle comparisons must be made to find suitable linking candidates. Because this analysis is utilising very small networks, several training runs were required to find neurons where the angle difference was less than 10°. The value of 10° was acquired from figure 3.12. The weight matrixes found are shown in tables 4.2 and 4.3 and their corresponding angle differences in table 4.8.

Table 4.8 Angles between weight vectors in ascending order.

Vector pair	Angle between vectors
v_2, v_3	9.147°
v_1, v_3	34.54°
v_2, v_4	140.95°
v_1, v_4	166.97°

An acceptance angle of 10° has been chosen on the basis that a reasonable amount of recall accuracy is maintained so that a combined network output can be compared with the individual network outputs from figure 4.15. The angle difference between vectors $\mathbf{v}_2$ and $\mathbf{v}_3$ is below the acceptance angle of 10° and therefore can be linked as outlined in section 2.6 and equation (2.49).

The resulting vector $\mathbf{v}_{r1}$ after linking is presented in table 4.9 with its relative errors in table 4.10. Please note that the vector length correction factor F_1 utilised for domain A is always 1 and F_2 utilised for domain B is 0.33224.

Table 4.9 Results of the linking of two neurons with acceptance angles below 10°.

Original vector references	Resulting vector $\mathbf{v}_{r1}$			
	w_{11}	w_{12}	w_{1B}	Factor F_2
v_2, v_3	-0.62391	0.3645	-1.16656	0.33224

Table 4.10 Vector component change impact analysis.

Reconstructed Vectors	Vector components			Relative Errors		
	w'_{11}	w'_{12}	w'_{1B}	$f(w_{11}, w'_{11})$	$f(w_{12}, w'_{12})$	$f(w_{1B}, w'_{1B})$
v'_2	-0.6239	-0.3645	-1.16656	0.08%	0.13%	-0.03%
v'_3	-0.20728	-0.1211	-0.38757	6.25%	-36.81%	4.17%

Table 4.10 shows one of the characteristics of equation (2.49), which is that the largest error should be with the smallest weight and the smallest errors with the largest weight.

In section 4.2 two different points of stimuli induction to a linked neuron were introduced and equations (4.1) to (4.4) were derived for stimuli induction prior to activation function. Having determined the output of a linked neuron with stimuli induction prior to the activation function in equation (4.19), the change of the stimuli induction point after the activation function is easy. It involves simply moving the

summed product of stimuli S_d and length correction factors F_d outside the activation function as shown in equation (4.20).

$$out'_j = act\left[\mathbf{x}_{in} \cdot \mathbf{v}_{r1} \cdot \sum_{d=1}^{q} (S_d \cdot F_d)\right] \tag{4.19}$$

$$out'_j = act\left[\mathbf{x}_{in} \cdot \mathbf{v}_{r1}\right] \cdot \sum_{d=1}^{q} (S_d \cdot F_d) \tag{4.20}$$

Equations (4.19) and (4.20) define the neuron outputs in a linked network for two different stimuli induction points. To visualise and verify the appropriateness of both equations, figure 4.17 shows a corresponding diagram in more detail as previously shown in section 4.2, figure 4.5.

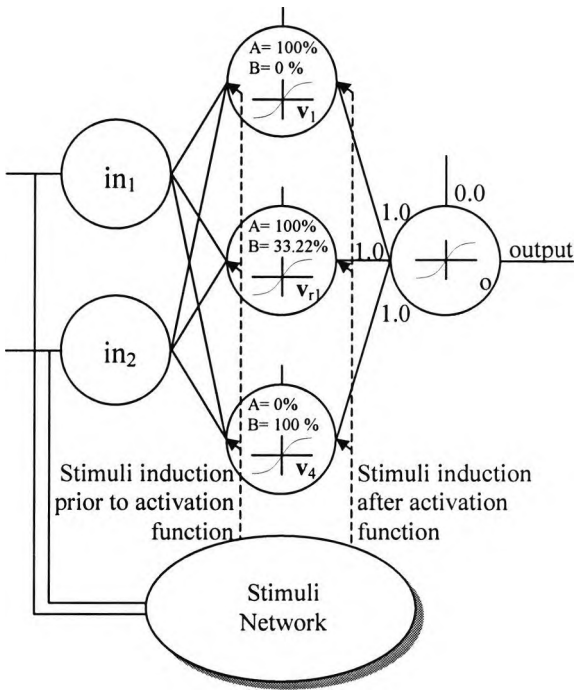


Figure 4.17 Linked neural network for illustration of stimuli induction points.

4.7.3.3 Stimuli Induction prior to Activation Function

This section is demonstrating the calculation details of the neural network with three neurons, as shown in figure 4.17. Because vectors $\mathbf{v}_2$ and $\mathbf{v}_3$ have been linked, vectors $\mathbf{v}_1$, $\mathbf{v}_{1r}$ and $\mathbf{v}_4$ correspond to the neurons in figure 4.17. Therefore, the network consists of one linked ($\mathbf{v}_{1r}$) and two non-linked neurons ($\mathbf{v}_1$ and $\mathbf{v}_4$). Stimuli induction is prior to the activation function and all detailed calculations are referring to the second point (0.24467, 0.38550) from table 4.6 and are presented for each neuron at different stages, as shown in table 4.11.

Table 4.11 Network output with stimuli induction prior to activation function.

ref	in_1	in_2	net_j	S	$net_j \cdot S$	out_j	net_k	Output
v_1	0.24467	0.38550	-0.6925	0.78166	-0.5413	-0.2642		
v_{r1}	0.24467	0.38550	-1.4597	0.85420	-1.2469	-0.5535	-0.6931	-0.3333
v_4	0.24467	0.38550	1.1477	0.21833	0.2506	0.1246		

Examination of the top row in table 4.11 for a non-linked neuron ($\mathbf{v}_1$) with 100% membership to domain A yield the following calculations:

$$net_j^{v1} = (0.24467 \cdot -0.2019) + (0.38855 \cdot 0.10318) + -0.6829 = -0.6925 \quad (4.21)$$

$$S^{v1} = 1.0 \cdot 0.7816 = 0.7816 \quad (4.22)$$

$$net_j^{v1} \cdot S^{v1} = -0.6925 \cdot 0.78166 = -0.5413 \quad (4.23)$$

$$out_j^{v1} = (2/(1+EXP(0.54132))-1) = -0.2642 \quad (4.24)$$

Examination of the second row in table 4.10 for a linked neuron ($\mathbf{v}_{r1}$) with 100% membership to domain A and 33.22% to domain B, with factor $F_1 = 1$ from section 4.3 and F_2 from figure 4.8, yield the following calculations:

$$net_j^{vr1} = (0.24467 \cdot -0.6239) + (0.38855 \cdot -0.3645) + -1.1666) = -1.4597 \quad (4.25)$$

$$S^{vr1} = 1.0 \cdot 0.7816 + 0.3322 \cdot 0.2183 = 0.8542 \quad (4.26)$$

$$net_j^{vr1} \cdot S^{vr1} = -1.4597 \cdot 0.8542 = -1.2469 \quad (4.27)$$

$$out_j^{v1} = (2/(1+EXP(1.2469))-1) = -0.5535 \quad (4.28)$$

Examination of the last row in table 4.10 for a non-linked neuron (v_4) with 33.22% membership to domain B yield the following calculations:

$$net_j^{v4} = (0.24467 \cdot 0.3060) + (0.38855 \cdot -0.49898) + -1.2652) = 1.1477 \quad (4.29)$$

$$S^{v4} = 1.0 \cdot 0.2183 = 0.2183 \quad (4.30)$$

$$net_j^{v4} \cdot S^{v4} = 1.14774 \cdot 0.21833 = 0.2506 \quad (4.31)$$

$$out_j^{v4} = (2/(1+EXP(-0.25059))-1) = 0.1246 \quad (4.32)$$

Similar output characteristics of the linked network in figure 4.18 and of the linearly combined network in figure 4.16 can be observed. The reason for this is, that small weights are causing linear operation of the linked networks, as discussed in section 4.6 and presented in figure 4.10. On comparison of the SSE's shown in tables 4.7 and 4.12, only a slight reduction in domain A is noticeable, which is a result of a reduced movement of domain A towards domain B.

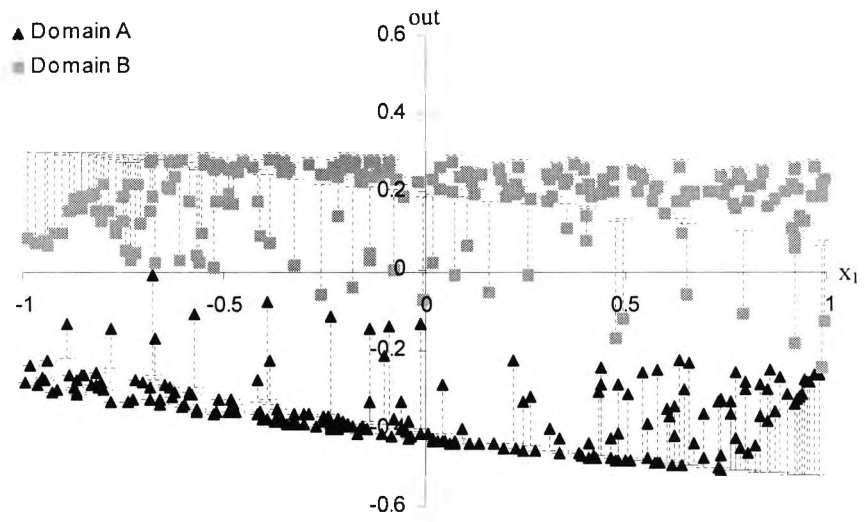


Figure 4.18 Linked output for domain A (▲) and B (■) with stimuli prior to activation function.

Above, error bars are denoting the directional change of the neuron outputs, whilst the input x_1 remain the same for all figures showing such error bars. Horizontal dashes represent the individual network outputs and the markers represent the current.

Table 4.12 SSE of linked domains with stimuli induction prior to activation function.

Domain	Sum Square Error (SSE)
A	1.4776
B	1.4890

4.7.3.4 Stimuli Induction after Activation Function

Two stimuli induction points were considered, one prior and one after the neuron activation function. The calculation for stimuli induction after the activation function was given in equation (4.20). Because of the different point of induction, equations (4.22), (4.26) and (4.30) are not applied to (4.23), (4.27) and (4.31), they are applied to (4.24), (4.28) and (4.32) instead. Because of the trivial changes, calculation details are omitted and the results are shown in table 4.13.

Table 4.13 Network output with stimuli induction after activation function.

ref	in_1	in_2	net_j	S	$out_j \cdot S$	net_k	Output
V_1	0.24467	0.3855	-0.6925	0.78166	-0.26034		
v_{r1}	0.24467	0.3855	-1.4597	0.85420	-0.53216	-0.67935	-0.32719
V_4	0.24467	0.38556	1.14774	0.21833	0.11314		

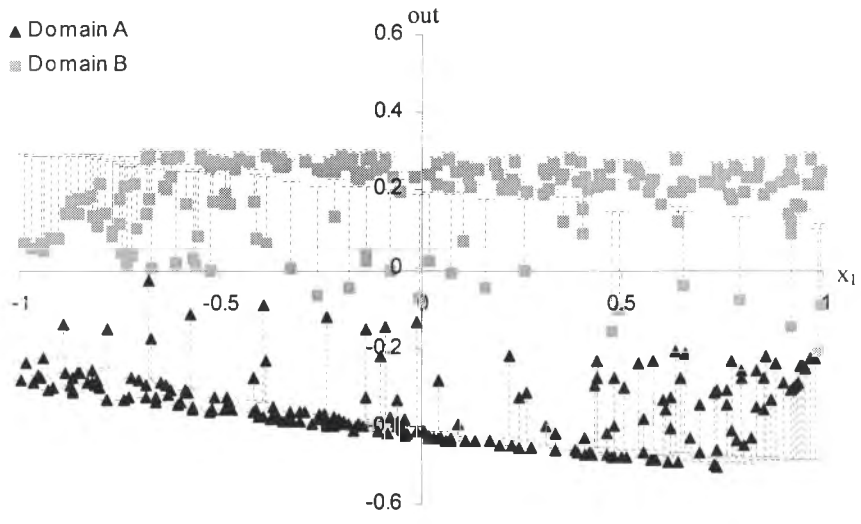


Figure 4.19 Linked output for domain A (▲) and B (■) with stimuli after activation function.

If figures 4.16 and 4.19 are compared, similar characteristics in the change of output can be observed. Stimuli induction prior or after the activation function does not differ largely when unsaturated neurons are present. The almost identical SSE in table 4.14 compared to tables 4.7 and 4.12, supports this statement.

Table 4.14 SSE of linked domains with stimuli induction after activation function.

Domain	Sum Square Error (SSE)
A	1.6035
B	1.5272

A SSE comparison between the three different network combination methods in figure 4.20 shows little differences. Linked networks behave similar to linearly combined networks if no neuron saturation is present.

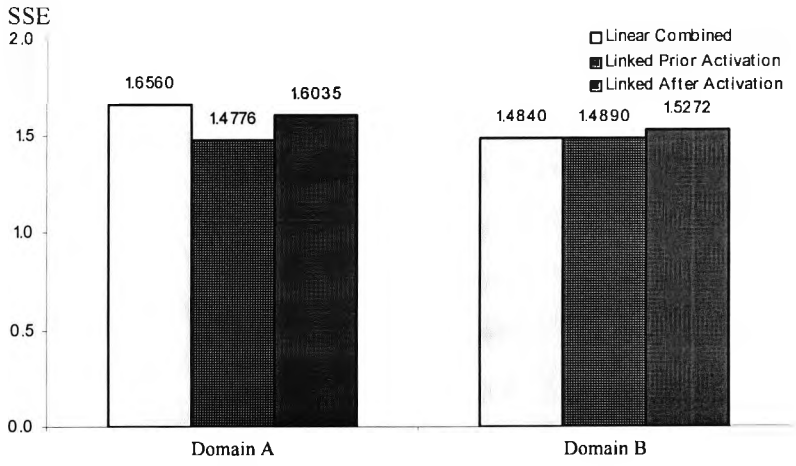


Figure 4.20 Comparison of SSE between different types of network assembles.

In the following sections, saturated neurons will be analysed and a repeat of all tests, which have been applied to non-saturated neurons, is made to saturated neurons to draw attention to the differences between stimuli induction prior and after neuron activation function.

4.7.4 Saturated Neurons

Saturated neurons are neurons where the sum of all connections net_j exceeds a value of 2.25 for most of the input space, as discussed in section 4.6 equation (4.9). Because of saturation, neuron outputs will largely remain in the activation functions max or min regions, e.g. ±1 for symmetrical-sigmoid, leaving only a small portion of the input space for neuron activity. As a result, the saturated network output in figure 4.21 shows domain separation mainly in limits of the neuron activation function.

In this section, the effects of neuron saturation on combined networks with linear combiner and linked neurons are analysed. For this purpose, the individual network output for both domains, as shown in figure 4.21, is used as a reference in the same fashion as in section 4.7.3 figure 4.15.

The problem of domain separation, where clear domain boundaries are visible, is noticeable in the third quadrant for domain B. There, an almost straight line in the area of –0.8 does not move into the area occupied by domain A. With help of network combination methods, domain output should be less confined to its boundaries.

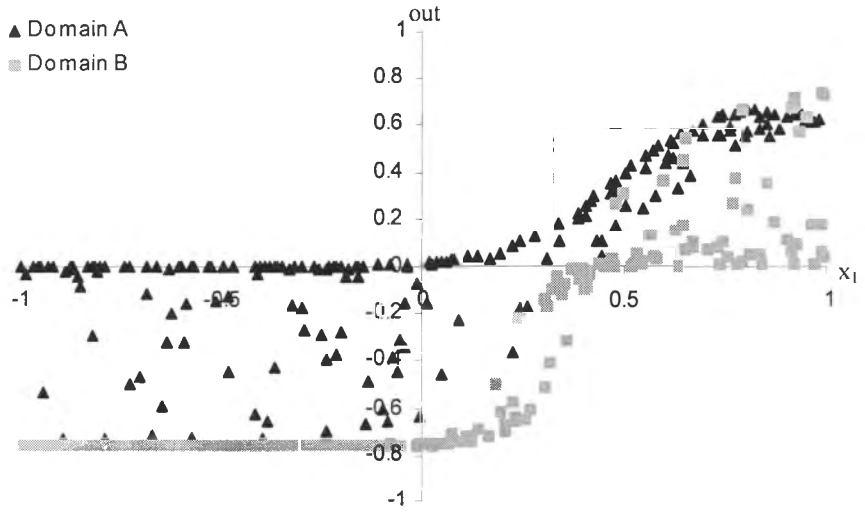


Figure 4.21 Network output for domain A (▲) and B (■).

4.7.4.1 Linear Combined Output

If networks used in an assembly are investigated for their individual performance, linear combiners are generally suitable for domain interaction as shown in section 4.3.7.1. But if networks in an assembly are suffering from over training or high sensitivity caused by neuron saturation, linear combiners may not sufficiently amalgamate individual network outputs. This section is investigating the effects of neuron saturation in a linearly combined network assembly.

Since the training input vectors remained unchanged but outputs have been changed, domain membership calculations remain unaffected. Because of this, individual network contributions are multiplied by the same magnitude prior to summation, computing the overall output with the same weighting factors as in section 4.7.3.1.

Figure 4.22 shows slight changes in the linearly combined output for domains A and B, observable by the error bars. These changes do not show as much knowledge domain interaction, if compared to figure 4.16. The reasons for this are that each network outputs operate in their own input space and are within extremes of the activation function.

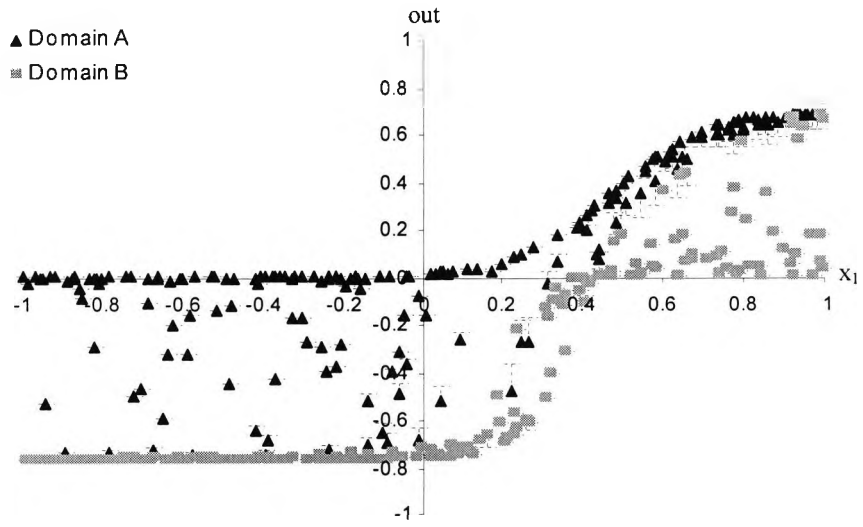


Figure 4.22 Linear combined output for domain A (▲) and B (■).

If an input vector has membership of both domains, the output of the domain in which it was present during training will be weighted whilst the output of the remaining domain is small. Thus only the network with matching domain can contribute. Because of this, satisfactory domain interaction cannot be achieved by linear combination. With this, the calculation of the SSE between figure 4.21 and 4.22 for both domains is relatively small, as shown in table 4.15.

Table 4.15 SSE between individual and linearly combined domains.

Domain	Sum Square Error (SSE)
A	0.4332
B	0.2880

4.7.4.2 Linked Output

To find suitable neurons with vector angle differences of less than 10° for linking, several training runs similar to section 4.7.3.2 have been made. The weight matrixes found are shown in tables 4.4 and 4.5 and their corresponding angle differences in table 4.16.

Table 4.16 Angles between weight vectors in ascending order.

Vector pair	Angle between vectors
v_5, v_7	128.01°
v_5, v_8	63.24°
v_6, v_7	5.078°
v_6, v_8	69.88°

An acceptance angle of 10° has been chosen, for the reasons mentioned in section 4.7.3.2. The angle difference between vectors $\mathbf{v}_6$ and $\mathbf{v}_7$ is below the acceptance angle of 10° and therefore can be linked as outlined in section 2.6 and equation (2.49).

The resulting vector v_{r1} after linking is presented in table 4.17 with its relative errors in table 4.18. Please note that the vector length correction factor F_1 utilised for domain A is always 1 and F_2 utilised for domain B is 2.6710.

Table 4.17 Results of the linking of two neurons with acceptance angles below 10°.

Original vector references	Resulting vector v_{r1}			
	w_{11}	w_{12}	w_{1B}	Factor F_2
v_6, v_7	7.9868	-1.3691	-2.6021	2.6710

Table 4.18 Vector component change impact analysis.

Reconstructed Vectors	Vector components			Relative Errors		
	w'_{11}	w'_{12}	w'_{1B}	$f(w_{11}, w'_{11})$	$f(w_{12}, w'_{12})$	$f(w_{1B}, w'_{1B})$
v'_6	7.9868	-1.3691	-2.6021	0.47%	-33.57 %	10.62%
v'_7	21.3328	-3.6567	-6.9502	-0.08%	1.70%	0.27%

Table 4.18 shows one of the main characteristics of equation (2.49), which is that the largest error should be with the smallest weight and the smallest errors with the largest weight.

4.7.4.3 Stimuli Induction prior to Activation Function

In this section both neural networks are linked and controlled by a stimuli network by utilising the domain membership function from figure 4.13. This section is equivalent to section 4.7.3.3 with the only difference being the network weights used are from tables 4.4 and 4.5.

The calculation for stimuli induction prior to the activation function was given in equation (4.19). All detailed calculations in equations (4.21) to (4.32) are applicable to this section if the corresponding parameters from table 4.19 are substituted. These are minor changes to equations (4.21) to (4.32) and further calculation details are omitted and the results shown in table 4.19.

Because vectors $\mathbf{v}_6$ and $\mathbf{v}_7$ have been linked, vectors $\mathbf{v}_5$, $\mathbf{v}_{1r}$ and $\mathbf{v}_8$ correspond to the neurons in figure 4.17. Therefore, the network consists of one linked ($\mathbf{v}_{1r}$) and two non-linked neurons ($\mathbf{v}_5$ and $\mathbf{v}_8$). Stimuli induction is prior to the activation function and all results are referring to the second point (0.24467, 0.38550) from table 4.6 and are presented for each neuron at different stages, as shown in table 4.19.

Table 4.19 Network output with stimuli induction prior to activation function.

ref	in_1	in_2	net_j	S	$net_j \cdot S$	out_j	net_k	Output
v_i	0.24467	0.38550	0.3538	0.78166	0.2765	0.1374		
v_{r1}	0.24467	0.38550	-1.1757	1.36483	-1.6047	-0.6653	-0.6440	-0.3113
v_4	0.24467	0.38550	-1.0683	0.21833	-0.2332	-0.1161		

On observation of the error bars in figure 4.23, a shift of the output from domain B towards domain A can be seen. This is for the reason that domain A is influencing the output of domain B in such a way that outputs of domain B are moved closer to domain A. This domain interaction was strongly reduced with the linear combiner if the SSE in tables 4.15 and 4.19 are compared. The increase of the SSE in table 4.20 indicates an increased interaction between knowledge domains.

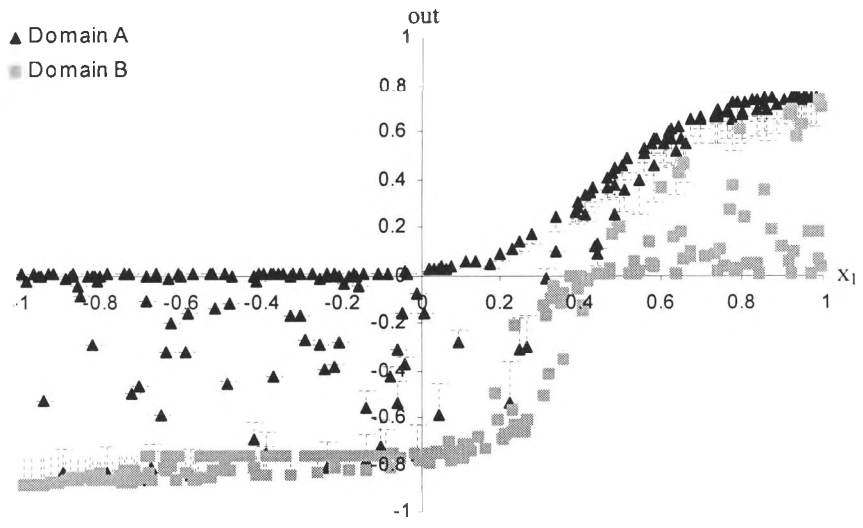


Figure 4.23 Linked output for domain A (▲) and B (■).

Table 4.20 SSE of linked domains with stimuli induction prior to activation function.

Domain	Sum Square Error (SSE)
A	0.9394
B	0.6997

4.7.4.4 Stimuli Induction after Activation Function

On observation of the error bars in figure 4.24, a strong shift of domain B towards domain A can be seen. This is caused by neuron saturation combined with induction of the stimuli after the activation function, which has only a scaling effect on the totally saturated output. This loss of information has resulted in the display of the sigmoid function for domain A. The saturated neurons of domain B combined with the stimuli induction after the neuron activation function has caused a stark domain interaction between domain A and B in quadrant 1, shifting the output of network B to domain A.

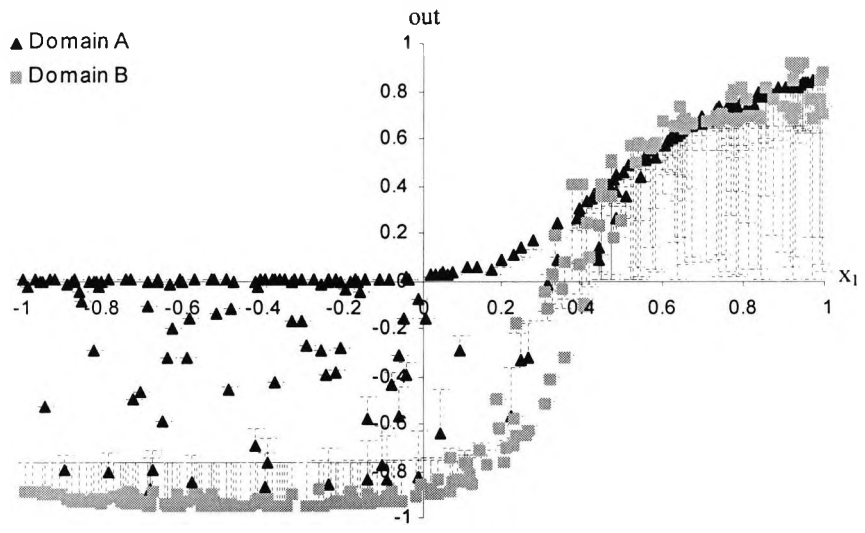


Figure 4.24 Linked output for domain A (▲) and B (■) networks.

The large SSE shown in table 4.21 indicates a very strong interaction between networks A and B to such an extent that network B has almost the same output as network A in quadrant 1.

Table 4.21 SSE of linked domains with stimuli induction after activation function.

Domain	Sum Square Error (SSE)
A	1.3336
B	4.2573

A SSE comparison between the three different network combination methods in figure 4.25 shows a large difference for saturated neurons. Therefore it can be stated that the output of a linked network is affected by saturated neurons more severely if the stimuli injection is located after the activation function.

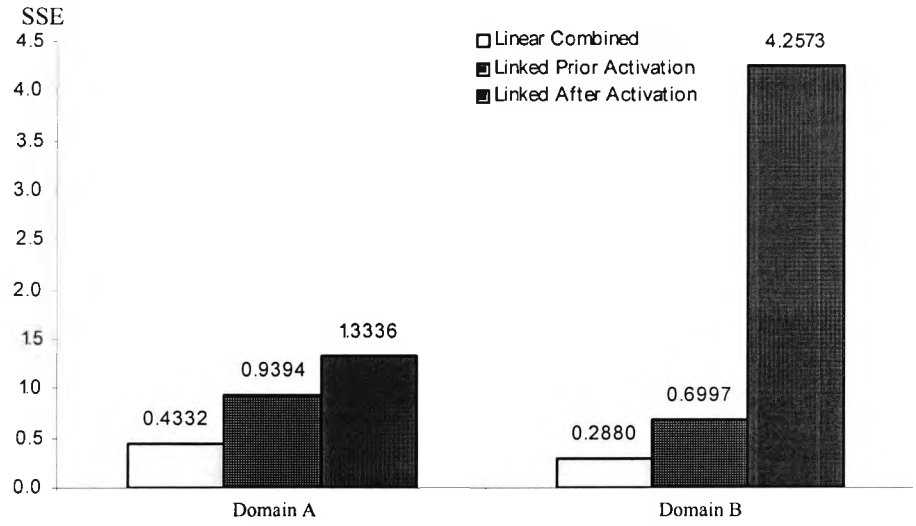


Figure 4.25 Comparison of SSE between different types of network assembles.

4.8 Conclusion

For the purpose of activating the most appropriate domain experts a stimulus for each domain has been introduced. The domain stimuli, generated by a stimuli network, have been used to control the outputs of individual experts in order to generate the overall network output. One of the major differences of existing multiple expert (ME) systems is that the classification results, generated by the stimuli network, are used to control individual neurons in each expert, instead of the output of the entire network, as shown in figure 4.3. The classification from the stimuli network, referred to as the stimuli factor S, has been used in conjunction with the weight vector length correction factor F of linked neurons to control the contribution of neurons to the entire network output.

Because classifiers networks that produce stimuli act on neurons internally, the combined input net_j of a neuron should not exceed 2.3, as shown in section 4.6. This is because the summed neuron input net_j is multiplied with the stimulus factor S and if the stimulus S is small the neuron contribution should be reduced, which is not the case if net_j is very large. Neurons with a large net_j have been defined in section 4.5 as saturated and neuron saturation should be avoided if the linked neurons should operate satisfactory.

The numerical experiment in section 4.7 concludes that the presents of saturated neurons has been the cause for interaction between two domains even if the input vector originated from one domain only. This is not desirable since the contribution of each domain expert should be controllable by the stimuli network, which is not the case if saturated neurons are present. Furthermore, the neuron output is less affected by saturation if the stimuli induction point is located after the activation function. But outputs of neurons that were not saturated had only a slight variation in the recall and generalisation errors. Since neuron saturation cannot be prevented in all cases, stimuli should be applied prior to the activation function to reduce the effects of saturation.

Chapter 5

Linking of Neural Network Weight Matrixes

5.1 Introduction

This chapter will show how neural networks can be linked to alter their recall and generalisation capabilities. In the first example, two neural networks will be trained with data describing a 3D path. This example is using the same number of training patterns as the number of hidden neurons, causing the network to suffer from memorisation or overfitting [38, 138]. In the subsequent examples, the training and testing patterns are split into quadrants to investigate the impact of linking to interpolation and extrapolation [161, 169].

The next section will extend the framework of neuron linking to the linking entire trained weight matrixes, each trained for a particular knowledge domain, to create a single entity. This entity will consist of one or more partially linked matrixes to share common information between knowledge domains. Furthermore, to categorise the domain a possible knowledge request belongs to, a classification network is used similar to existing structures of mixtures of expert systems [14, 31, 43, 48, 49]. Since the linked network structure will generate the overall network output, conventional network combination methods as used in mixtures of experts, such as switching, averaging or voting, differ substantially to a linked network [156-158].

Hidden neurons in linked networks differ in their composition and functionality to regular hidden neurons used in ordinary neural networks. Regular hidden neurons consist only of a weight matrix, a summing input and an activation function [77]. Linked hidden neurons additionally contain a vector length adjustment factor F that represents information about the degree of membership towards a particular knowledge domain. On data recall, the internal activation of a linked neuron depends on the input vector, the weight vector and their membership to the knowledge domain in question.

Commonly, a classification network is generating the knowledge domain membership categorisation [106, 159, 160]. This classification network computes the knowledge domain membership from the input space of the input vector. While a gating network acts generally on the outputs of multiple experts, the classification network for a linked network acts on the hidden neurons directly. The difference is that gating networks operate on experts as a whole, whereas in the case of a linked network, the individual neurons compute their output depending on the domain membership S and the vector length adjustment factor F.

Because of the influence of the classification network on the neurons an analogy to the terms used in psychology can be introduced. In psychology, respondent behaviour describes behaviour that occurs as an automatic response to some *stimulus* [124]. Since in a linked network different neurons are activated for different domain requests, the network response is depending on all stimuli received. For that reason, the classification network will be referred to as the stimuli network, and the outputs of the stimuli network will be referred to as the stimuli.

5.2 The Principles of Linking Sub-Networks

The linking of weight matrixes is bringing a variety of advantages, for example, reduced training time, weight matrix reusability, distributed training on different locations or computers, parallel training by divide-and-conquer of the training data,

domain knowledge sharing and improved generalisation caused by the implicit mixture of experts strategy. Once a weight matrix contains knowledge it can be added to an existing matrix without any re-training. Therefore, knowledge can be reused by induction into a system without the need of the original training data, avoiding re-training and associated issued such as training data preparation.

The linking of entire weight matrixes demonstrates the possibility of combining pre-trained matrixes with newly trained matrixes as shown in figure 5.1. Pre-trained matrixes could have been purchased from third parties or taken out from an existing repository.

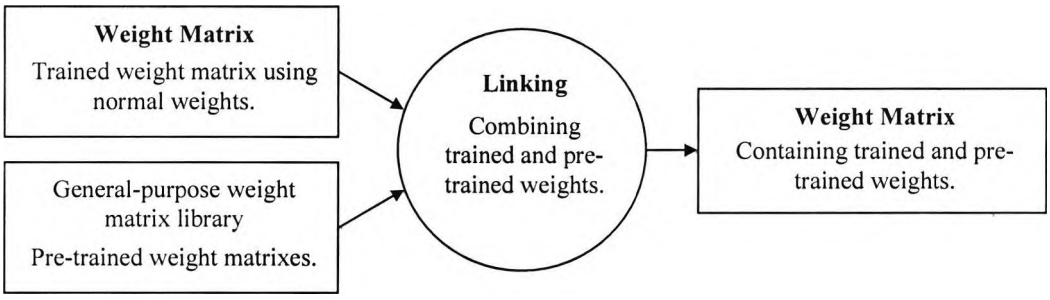


Figure 5.1 Linking trained weights with pre-trained weights from library.

Linking of trained network matrixes will allow integrating knowledge from different sources into one weight matrix without the need for the training data. With linking, pre-trained weight matrixes will become available for different problem domains without the need for data preparation, network training and testing. It will be much easier to build data models on information about different areas of interest, without having to comb through masses of data for the purpose of creating training patterns containing all the portions of interest. Instead of consolidating the training data, separate networks can be trained on independent sets of training data and subsequently combined via linking.

Databases are used to collect data subjects of interest e.g. company performance figures or weather information, but this can be converted into a collection of pre-trained neural network weight matrixes that can store similar information about the

knowledge domain, see figure 5.2. The user of a pre-trained weight matrix needs to keep in mind for which purpose a certain weight matrix has been created and for which functionality it was optimised. If for example two weight matrixes of two different domains are to be linked for the purpose of interpolation, both matrixes should have been trained and tested for optimal performance on generalisation for a defined input space. Once the individual networks are linked, the combined network is able to cover the combined input space of each individual network with confidence.

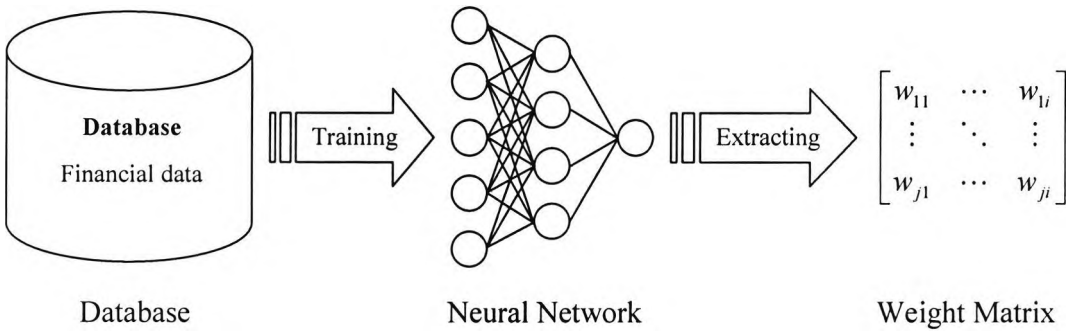


Figure 5.2 Similar knowledge contained in a database or in a weight matrix.

Further extension of the analogy of C-based programming languages will bring us into the more complex field of memory maps, which can be compared with the input space for which a weight matrix has been tested and found appropriate for its purpose. If a C-based programming language is supplied with a library of functions, it needs to have an associated memory map to locate the point of entry, as presented in figure 5.3 [1-3]. A memory map is a guide for locating where a certain function can be found in memory. The equivalent of a memory map in C-programming is the input space of a weight matrix for neural networks. In C the memory map defines where a function can be found, whereby a description of the tested input space of a weight matrix outlines where reliable knowledge can be found. In order to locate reliable knowledge in a weight matrix the associated input space is required, this is referred to as a knowledge domain map (KDM) in this thesis and is shown in figure 5.4.

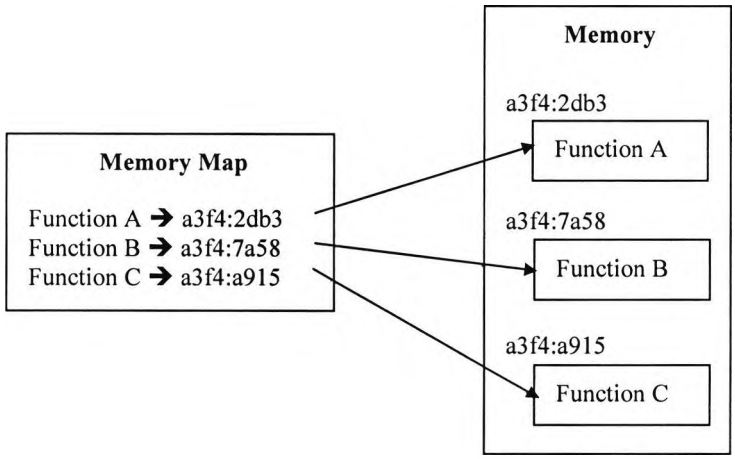


Figure 5.3 In C-based languages use memory maps to locate functions.

Identification of knowledge in a training data set can be achieved via an input space analysis. Such an analysis is basically a search for clusters within the input space to identify areas of interest where a high data density exists. Areas with high data density are higher in recall confidence than areas with sparse data density. Self Organising Maps (SOM) [77] or Principal Component Analysis (PCA) [37] are the two most common methods to find such clusters of high data density. Both are iterative dimension reduction methods, which are based on data observations.

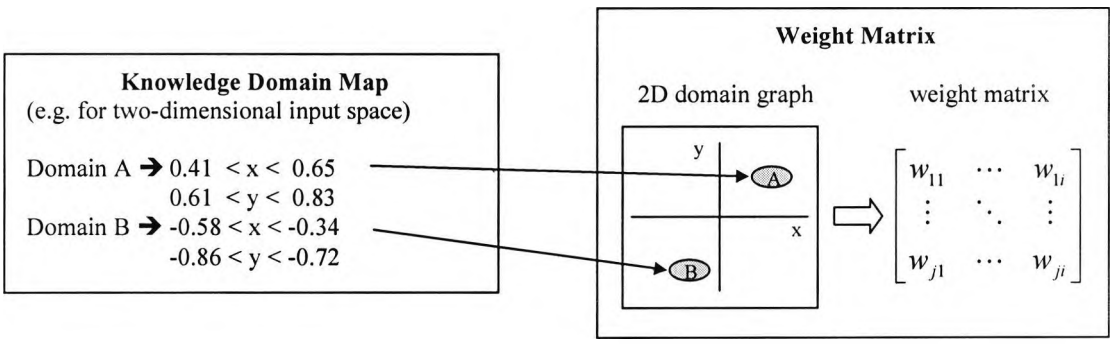


Figure 5.4 A Knowledge Domain Map locates areas of knowledge in a weight matrix.

5.3 Linking Sub-Networks

The linking process will combine two or more trained and fully connected sub-networks into one large network. This will be accomplished by comparing hidden neuron vectors of each sub-network or knowledge domain. If two or more similar vectors have been found, the neuron linking process combines them to create a single linked hidden neuron. The weight vectors of the resulting hidden neuron will be calculated as specified in section 2.6 and chapter 3. These resulting hidden neurons will build links between the sub-networks or knowledge domains and will be shared by data requests for a particular domain.

In order to retain the accuracy of each knowledge base, a unique number, referred to as a *stimulus code*, will be assigned to all hidden neurons. This number is used to tag every hidden neuron in each knowledge domain prior to linking. If two or more hidden neurons of different knowledge domains are linked, the resulting neuron will carry both stimuli codes, one from each knowledge domain. The stimuli code is used to identify the appropriate vector length adjustment factor to be utilised on data recall. All hidden neurons will carry a code for every knowledge domain involved in the linking process. Neurons, which do not satisfy the *acceptance angle* constraint on linking from chapter 3, will not contribute to the domain they were linked to. Such neurons will still carry a stimuli code reference to the domain linked to, but the contribution to the output on data recall will be zero.

If an input vector of a particular problem domain is applied to a linked network for data recall, a process of activating hidden neurons to contribute to the domain in question is required. This has been achieved by utilising a *stimuli network* to classify the input space of the incoming vector to the input layer and produce one or more *stimuli*. Only neurons, which are contributing to the stimulus categories generated by the stimuli network, will be contributing in the feed forward calculation of the linked network to produce the overall network output.

5.3.1 The Neural Network Linking Process

The neural network linking process will combine two or more trained and fully connected neural networks into one large network. The linking process can be split into five major steps:

1) Training of sub-networks with data from different knowledge domains.
2) Identification of all hidden neurons that satisfy the acceptance angle constraint from both sub-networks involved.
3) Calculation of the resulting vectors $\mathbf{v}_{rl}$ for each group of linked hidden neurons.
4) Tagging all hidden neurons with a stimulus code.
5) Training of a stimuli network, which will be used to produce the input vector classification stimuli for the linked network.

5.3.2 Training Sub-Networks

For the purpose of linking, two 2:10:1 neural networks A and B have been trained with the standard backpropagation algorithm. The training data used for each of the networks describes an arbitrary path in a 3 dimensional space. The testing data used to calculate the generalisation error of each of the networks were data points chosen in-between training data points plus 10% noise to avoid direct linear relationship to the training data. Figure 5.5 portrays two 3D functions A and B used to train both networks A and B respectively. Data points used for training are shown as filled markers (suffix: trn) and test data points as empty markers (suffix: gen). In order to use a symmetric sigmoid activation function, the weights connecting the hidden layer to the output layer have been set to 1 and frozen in the sense that they are not adjusted during training, permitting simplified linking without taking the weights connecting the hidden layer to the output layer into consideration, as mentioned in section 3.3. The parameters of the networks used are given in table 5.1.

To emphasise the problem with generalisation for over-trained networks, only 10 data points per function have been used to train the 2:10:1 networks. The reason for using

the same number of training patterns, as there are hidden neurons, is to cause a direct mapping of the training data into the hidden layer. With direct mapping, the network is effectively learning the input data as a lookup table and will not be able to generalise satisfactory, as it is the case with over-trained neural networks.

Table 5.1 The parameters of the neural networks used in this section.

Description	**Value** (for each network)
Input Neurons	2
Hidden Neurons	10
Output neurons	1
Activation Function	symmetric sigmoid
Initialisation	±0.7
Learning Factor	0.3
Momentum	0.5
Number of training patterns	10
Number of testing patterns	9 per data set

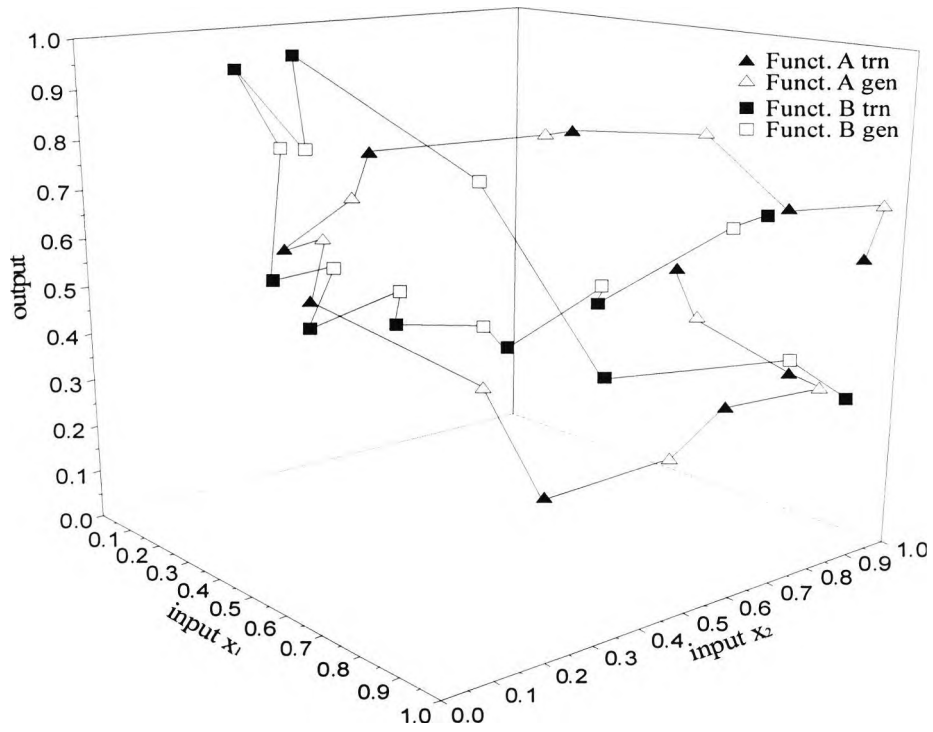


Figure 5.5 3D plot of the training and testing data for networks A and B.

During the course of training, the training SSE_{trn} and testing SSE_{gen} errors have been recorded as illustrated in figure 5.6.

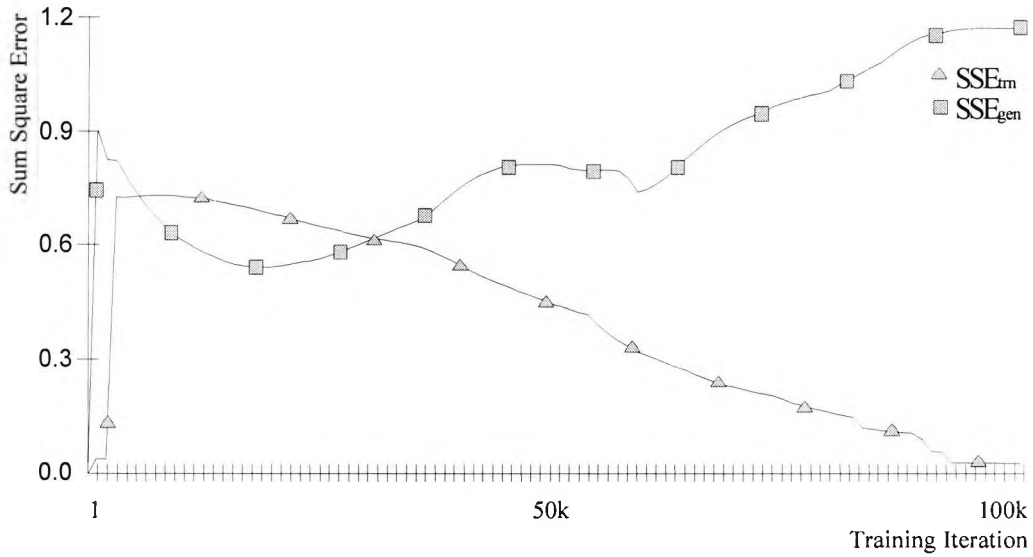


Figure 5.6 The recall and generalisation error during training of network A.

It can be observed that the generalisation error SSE_{gen} is rising during training whilst the recall error SSE_{trn} is falling. After approximately 100,000 training iterations the generalisation error as well as the recall error remain unchanged. This is because the training patterns are memorised in the hidden weight matrix. Because there are no free parameters within the network and the weights between the hidden and output layer are frozen, free large weights and large counter weights cannot be created [140, 141]. Such increasing weights are responsible for an increase in the generalisation error. But without them, the generalisation error is levelling off and remains constant.

With frozen weights in the output layer, the error optimisation used in the backpropagation training algorithm can only adjust the parameters of the hidden layer. Because of this restriction, the network is not able to use the output layer to compensate for a local minimum that it may encounter in the hidden layer. Hence it is not able to memorise the training data precisely and therefore a difference between the target function and the recall function in recall mode is certain and can be observed in figure 5.6. SSE_{trn} and figure 5.7

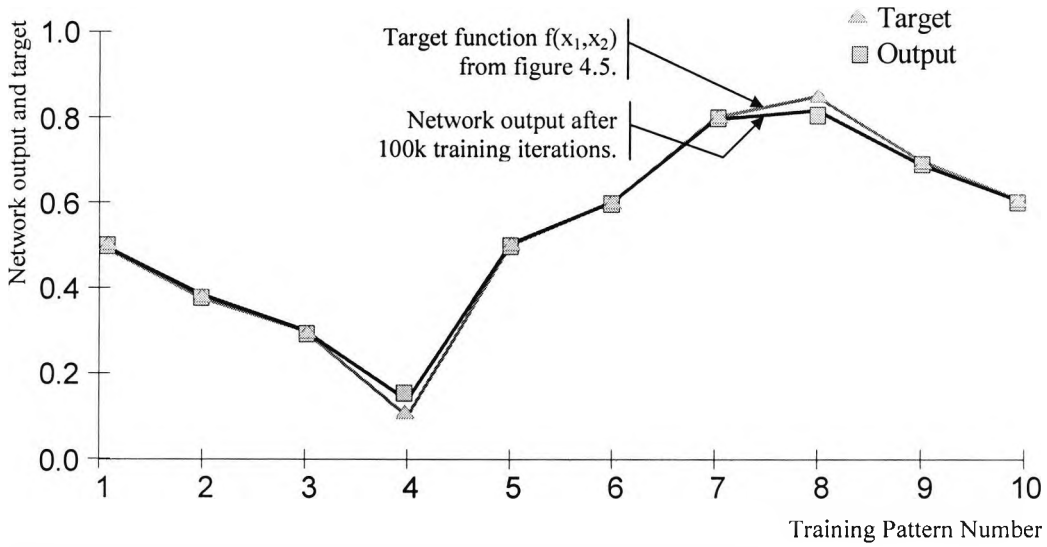


Figure 5.7 The target function plotted against the trained function of network A.

The weight matrix of network B, which has been trained on function B from figure 5.5, has been created with the same configuration as network A. The stopping criteria chosen to end training for both networks has been that the generalisation error remained almost unchanged for 1000 training iterations in batch update mode, as shown in figures 5.6 and 5.8.

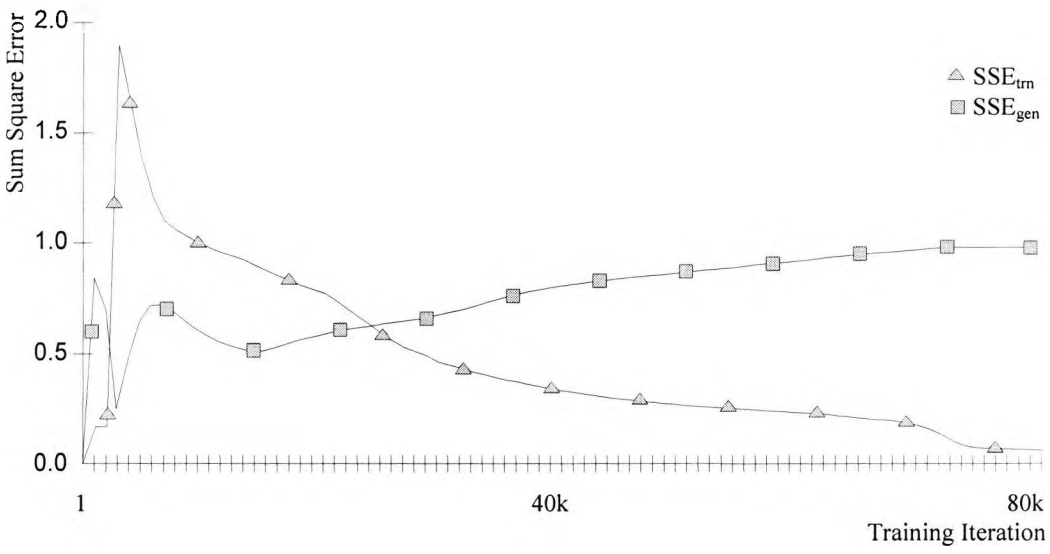


Figure 5.8 The recall and generalisation error during training of network B.

The output graphs for the training of network B with function B is shown in figure 5.9. It suffers from the same constraint of frozen output layer weights as network A, which is evident from the small difference in the target and recall output functions.

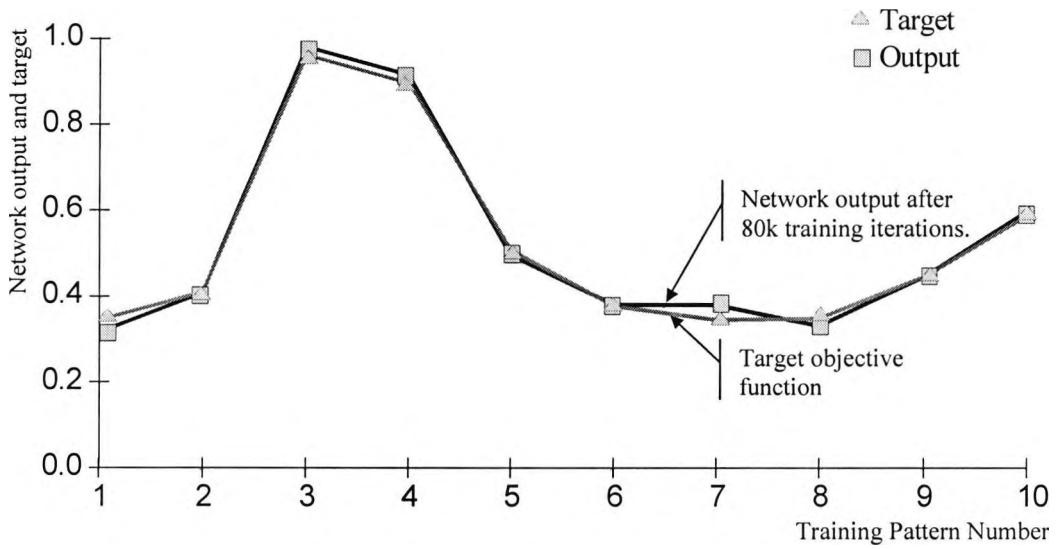


Figure 5.9 The target function plotted against the trained function of network B.

To compare the trained networks performance benchmarks before and after linking later in this chapter, the final recall error SSE_{trn} and generalisation error SSE_{gen} are summarised in table 5.2.

Table 5.2 Performance benchmarks of network A and B after training.

Description	Network A	Network B
SSE_{trn}	0.02	0.019
SSE_{gen}	1.243	1.149
RMSE for SSE_{trn} on 10 training record sets	0.00667	0.00633
RMSE for SSE_{gen} on 9 training record sets	0.41433	0.383

Because of the stopping criteria chosen, both networks training objectives have been for optimal performance in recall accuracy not for generalisation, thus causing the

generalisation error being much higher than the recall error. This is evident if recall and generalisation error are compared in table 5.2 for each of the two networks.

With the training of the two networks completed, the training data can now be represented as weight matrixes in conjunction with knowledge domain maps (KDM) as shown in figure 5.4. These components are presented in table 5.3 and in table 5.4 and describe the training data and their valid input space for each of the networks. The knowledge domain maps used are linear approximations, which accommodate 80% of the training data in a linear equation of least squares.

Table 5.3 Weight matrixes of the hidden layers of networks A and B after training.

Reference	Network A				Network B			
	w_{11}	w_{12}	w_{1B}	Length	w_{11}	w_{12}	w_{1B}	Length
v_1	-31.489202	29.520323	1.720558	43.197	-27.725769	14.899946	1.5787	31.515
v_2	7.949682	-2.061007	-2.352439	8.543	-2.445928	-1.432772	2.473297	3.762
v_3	15.573389	-15.559452	4.65308	22.501	2.200211	4.091093	-3.183682	5.632
v_4	-27.809488	26.555424	1.196535	38.471	-1.109374	0.43771	1.62007	2.012
v_5	-8.129432	-5.704075	2.906335	10.348	4.130769	-22.998222	2.681815	23.520
v_6	-3.471851	-3.113786	-0.185395	4.667	-20.964043	9.694241	1.382	23.138
v_7	-8.509409	-17.838032	19.4095	27.701	8.137968	-20.65534	0.620016	22.209
v_8	-6.621395	9.501069	-1.688886	11.703	-5.420225	-0.115172	0.124454	5.423
v_9	-18.781784	20.524035	-1.14477	27.844	-6.010482	-28.537815	7.361526	30.079
v_{10}	5.840691	4.331159	-9.77996	12.187	21.349447	-3.595532	-6.931421	22.733

Table 5.4 Knowledge Domain Maps for both training data sets.

Network	Area which matrixes were trained for
Matrix A	$f_{(a)} = x_2 = 1.05 \cdot x_1 - 0.03$ $(0.26<x_1<0.86)$
Matrix B	$f_{(b)} = x_2 = 1.47 \cdot x_1 - 0.04$ $(0.15<x_1<0.84)$

In table 5.4 the KDMs for each of the weight matrixes have been represented as linear equations and the range in which they are valid. The linear equations have been derived with help from figure 5.10, where the training data input space x_1 and x_2 has been plotted into a 2-dimensional graph for better visualisation. It can be observed that the input space region of network A lies mainly below the region separation line whereby the input space region of network B lies above. Hence $f_{(a)}$ is situated by

around 80% of data points from network A and $f_{(b)}$ is surrounded by 80% of data points from network B. Functions $f_{(a)}$ and $f_{(b)}$ have been calculated via the least squares of their surrounding data points. KDMs are used to coarsely describe the input space region for which the weight matrix has been trained. They can be used for guidance so that interpolation can be performed with confidence.

To bear in mind, the primary objective for linking of the two networks is to share knowledge acquired during training, with the purpose of improved generalisation for areas in the input space omitted by the training data of both networks involved. Therefore, the right candidates to demonstrate improved generalisation are over-trained networks of different domains, as used in this section.

To generate the appropriate training data for domain separation, the 2-dimensional input space x_1 and x_2 has been divided into two domains, A and B, as demonstrated in figure 5.10. Each set of the training data has been used to train one of the networks, set A with input space of domain A for network A and set B with input space of domain B for network B. The region separation line shown in figure 5.10 resembles the best fit for linear domain separation of the input space into A and B.

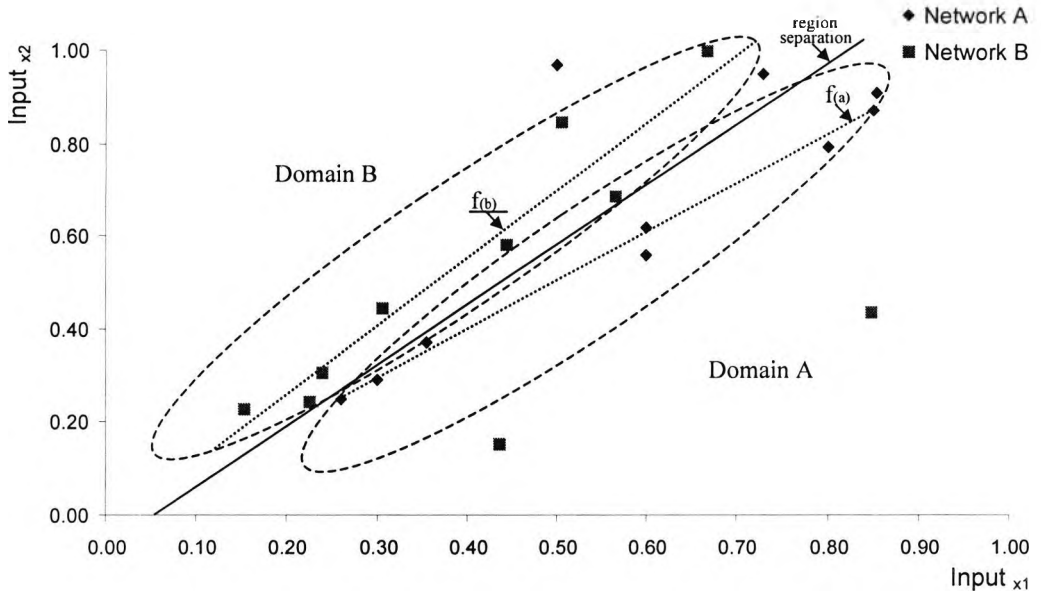


Figure 5.10 Input space separation of the training data.

Note that outliners from network A can be seen in domain B and vice versa. These outliners have been added to avoid total linear input space separation since linking requires neurons with similar knowledge. If the networks had been trained with data from different domains without any intersections, the probability of finding similar neurons would have been reduced, demanding a higher acceptance angle for linking with lower recall accuracy.

5.3.3 Sub-Network Linking Process

If two or more hidden neurons from different knowledge domains have highly correlated weight vectors, e.g. identical or opposite, then one of them can be made redundant by linking both hidden neurons into a single hidden neuron, as discussed in chapter 2. Hidden neuron vectors are unlikely to be exactly correlated, they will not match accurately. Therefore, an analytical approach utilising the acceptance angle φ, as introduced in chapter 3, is used. Hidden neurons will be understood as sufficiently correlated if the angle between their weight vectors in hyperspace does not exceed the acceptance angle φ.

Correlated hidden neuron weight vectors belong to the same domain and are holding similar knowledge. The process of linking correlating hidden neurons from different sub-networks of different knowledge domains will partially combine the knowledge of each network into one resulting network as shown in figure 5.11. Furthermore, removing superior correlations in the hidden layer will reduce the degree of freedom and therefore avoid problems such as over-fitting that can lead to an improved generalisation capability of the network as shown in section 3.3.4.

Figure 5.11 illustrates a linked network as a result of linking three domains. This network combines three knowledge domains A, B and C together into a single network and is capable of responding to requests from any of the domains involved. Intersecting areas between knowledge domains signify regions where hidden neurons have been linked and tagged with stimuli codes as members of the domains. The neurons contained in the intersections are used for recalling information by more than

one domain. If for instance a recall request has been received belonging only to domain A, only neurons carrying a membership code for domain A will be included in the generation of the overall network response. If, on the other hand, a request is received belonging to some degree to domain A and to some degree to domain B, neurons with appropriate membership codes of domain A and domain B will participate to the overall network response.

A stimuli network is used for classification to generate the degrees of memberships a specific input vector request may have. It analyses the input space and generates a measure of membership for each domain and each request. Depending on the strength of the measure, which has been referred to as the stimulus S, neurons of certain domains are activated to contribute to the overall network output.

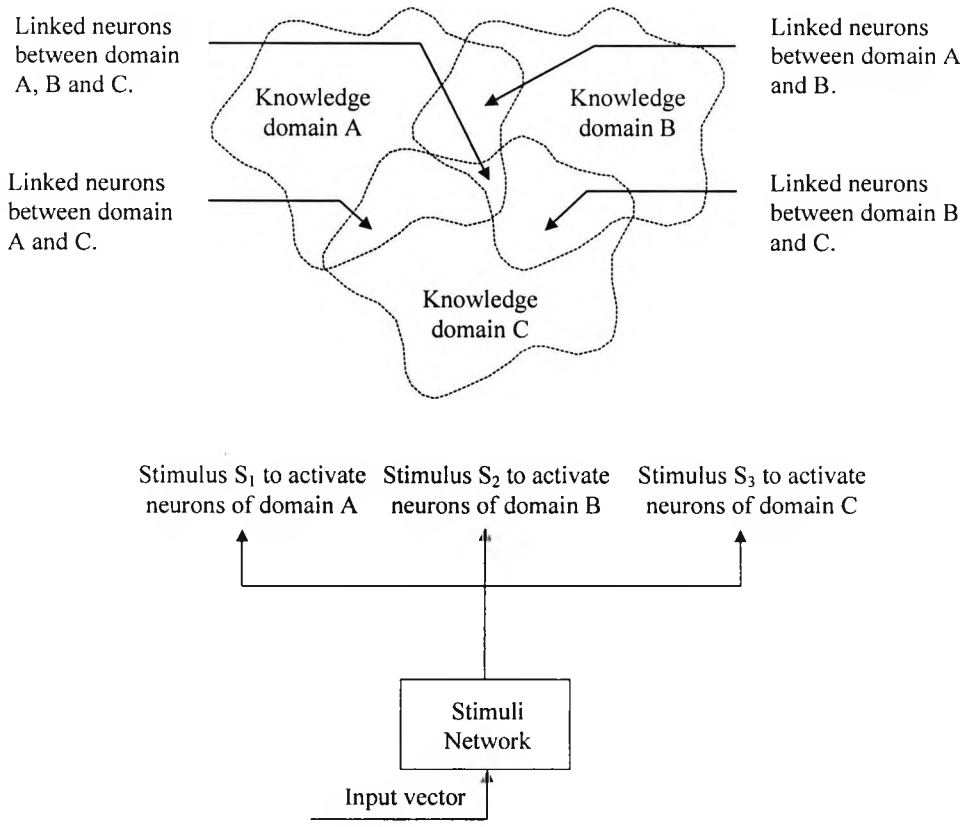


Figure 5.11 Linked domain sub-networks activated by a stimuli network.

In order to determine whether two hidden neuron vectors can be linked, an angle difference calculation must be performed for every weight vector for both sub-network weight matrixes. If the angle difference between two vectors exceeds the acceptance angle φ, the neurons cannot be linked. In this case it can be stated that weight vectors are not sufficiently linear dependent. Hidden neuron vectors where the angle difference does not exceed the acceptance angle φ can be stated as being sufficiently linear dependent and can be linked. However, in practice, there are only a few cases in which the weight vectors are exactly linear dependent. Therefore, increasing the acceptance angle φ will increase the angle tolerance, which in turn increases the probability of finding hidden neurons for linking. For the purpose of quantification of the hidden neurons involved in the linking process, a ratio called *linking rate* η has been defined. The more hidden neurons are combined the higher the linking rate η. The linking rate, as referred to in this thesis, is computed as the sum of all hidden neurons involved in the linking process divided by the sum of successfully linked neurons, as given in equation (5.1).

$$\eta = \frac{\sum_{a=1}^{N} v_A}{\sum_{b=1}^{P} v_B} \tag{5.1}$$

With N as the number of all hidden neurons from both sub-networks, $\mathbf{v}$ as the vector reference of neurons of domain A or B in the hidden layer and P as the number of successfully linked hidden neurons after completion of the linking process.

5.3.4 Analysis of Hidden Neuron Weight Vectors

Following the training of the domain sub-networks, hidden weight vectors need to be analysed for identification of those suitable for linking. For this purpose, all angles between vectors of domains A and B have been calculated. The number of angle

calculations ξ required for a full search of matching weights between domains A and B is given in (5.2).

$$\xi = N_A \cdot N_B \tag{5.2}$$

With N_A and N_B as the number of neurons to be linked from domains A and B respectively. The total number of angle calculations for the current example is N_A=10, N_B=10, ξ=100. Once all angles between the vectors of the domains A and B are known, they can be sorted in ascending order so that the acceptance angle constraint can be applied. Stepwise linking of neurons and measurement of the SSE_{trn} and SSE_{gen} has found the best acceptance angle of 13° for lowest generalisation error. The search of the 13° angle is presented later in this chapter. Table 5.3 shows the first three vector pairs that fall within the acceptance angle constraint of 13°.

Vectors from quadrants Q3 and Q4 have been mapped into quadrants Q1 and Q2 via vector inversion as previously mentioned in section 3.3.2 figure 3.10. Vector inversion will always be applicable, regardless of the number of vector dimensions, because the angle between two vectors is a scalar and the vector length adjustment factor F can be negative, as shown in tables 3.4 and 3.5 pruning of vectors $\mathbf{v}_2$ and $\mathbf{v}_{13}$.

Table 5.5 Angles between weight vectors in ascending order.

Vector pair		Angle between vectors
Network A	Network B	
$\mathbf{v}_2$	$\mathbf{v}_{10}$	5.078°
$\mathbf{v}_7$	$\mathbf{v}_3$	180°- 169.72 = 10.28° [1]
$\mathbf{v}_{10}$	$\mathbf{v}_2$	180°- 167.06 = 12.94° [1]

5.3.5 Linking of Hidden Neurons from Different Domains

Linking of the three vector pairs from table 5.5 with vector angles below 13° follows the approach derived in section 2.6 and used in section 3.3.3. After applying equations

[1] This vector has been mapped from Q3 to Q1.

(2.37), (2.49) and (2.52), as expressed in section 2.6, resulting vectors as shown in table 5.6 can be produced.

Table 5.6 Results of the combination of vectors with angles below 13° as listed in table 5.5.

Original vector references		Resulting vector v_{r1}			
Network A	Network B	w_{11}	w_{12}	w_{1B}	Factor F_2
v_2	v_{10}	7.986931	-1.369088	-2.602153	2.670965
v_7	v_3	-9.347131	-18.599433	18.201234	-0.202205
v_{10}	v_2	7.248415	4.406580	-8.547272	-0.311576

Linking will always involve two domains, one domain will use the resulting vector $v_{r1} \cdot F_1 = v_{r1}$ and all other domains will use $v_{r1} \cdot F_d$. Because the length correction factor F_2 for linking of domains A (d=1) and B (d=2) is only applicable to domain B, table 5.6 shows the vector length correction factor as F_2 since F_1, which is referring to the reference domain, will always be 1. If a third domain would be linked into a neuron already containing two domains, a new length correction factor F_3 would be added to the neuron's domain lookup table, as illustrated in chapter 4, figure 4.6.

5.4 Linking Analysis

5.4.1 Analysis of Linking Neurons

In order to evaluate the resulting weight vectors after linking, each vector component before and after linking is compared and analysed by the means of the relative error in percent, as shown in equation (5.3). Once neurons are linked, they contain the linked weight vector and a vector length adjustment factor F_d for each knowledge domain d involved in the linking process. Since this example involves only two domains (d = 2), only two vector length adjustment factors F_1 and F_2 and two domain stimulus factors S_1 and S_2 are given for each neuron. Whenever two neurons are linked, F_1 will

be 1 and F_2 is the weight correction factor as listed in table 5.6. The stimuli factors S_1 and S_2, calculated by the stimuli network during data recall, will determine the degree of membership of an input vector to each domain A and B.

$$f(w_{ji}, w'_{ji}) = \frac{w_{ji} - w'_{ji}}{w_{ji}} \tag{5.3}$$

If the stimuli network has classified an input vector on the input layer with a 100% membership to knowledge domain A and 0% to domain B, S_1 will be 1 and S_2 will be 0. The result will be, that the resulting vector $\mathbf{v}_{r1}$ will be multiplied with the sum $(S_1 \cdot F_1)+(S_2 \cdot F_2)$, but because $S_2 = 0$ only $(S_1 \cdot F_1)$ is significant. If, on the other hand, an incoming vector has been classified with a 0% membership to domain A and 100% membership to domain B, S_1 will be 0 and S_2 will be 1, but with S_1=0, only $S_2 \cdot F_2$ is significant.

With this scenario, weights for each knowledge domain can be calculated, which serve the purpose that the original weights, prior to linking, can be compared to the reconstructed weights, after linking. Possible changes in domain recall accuracy can be evaluated with the assumption that small weight changes on small weights will have a less prominent effect on the overall network output.

Table 5.7 Vector component change impact analysis.

Reconstructed Vectors	Vector components			Relative Errors		
	w'_{11}	w'_{12}	w'_{1B}	$f(w_{11}, w'_{11})$	$f(w_{12}, w'_{12})$	$f(w_{1B}, w'_{1B})$
A: v'_2	7.986931	-1.369088	-2.602153	0.47%	-33.57%	10.62%
B: v'_{10}	21.33281	-3.656787	-6.950259	-0.08%	1.70%	0.27%
A: v'_7	-9.347131	-18.599433	18.201234	9.84%	4.27%	-6.23%
B: v'_3	1.890038	3.760902	-3.680384	-14.10%	-8.07%	15.60%
A: v'_{10}	7.248415	4.406580	-8.547272	24.10%	1.74%	-12.60%
B: v'_2	-2.258432	-1.372984	2.663124	-7.67%	-4.17%	7.68%

The notation in table 5.7 will be that the reconstructed vectors v'_2 and v'_{10} represent $\mathbf{v}_{r1}$ and $\mathbf{v}_{r2}$ respectively for all vectors involved in the linking process, in accordance to

figure 3.4 in section 3.2.2 and replace vectors $\mathbf{v}_2$ and $\mathbf{v}_{10}$ from table 5.3. Furthermore, prefixes **A:** and **B:** refer to the knowledge domain for which the network matrix was trained for. Equation (5.4) has been utilised to reconstruct the vectors for domains A and B. It represents the actual weight matrix calculation for linked networks with multiple domains and has been extracted from chapter 4, equation (4.4).

$$w_j = v_{r1} \cdot \sum_{d=1}^{q} (S_d \cdot F_d) \tag{5.4}$$

Equation (5.4) has been used for the calculation of the reconstructed vectors for input vectors solely belonging to domains A and B. For example **A:** $\mathbf{v'}_2$ is the reconstructed vector for $\mathbf{v}_2$ of network A and can be compared to the original vector as listed in table 5.3. It has been computed as $\mathbf{v'}_2 = \mathbf{v}_{r1} \cdot S_1 \cdot F_1 = \mathbf{v}_{r1}$ for domain A (d=2, S_1=1, S_2=0, F_1=1, F_2=0) from table 5.6. Vector **B:** $\mathbf{v'}_{10}$ is the reconstructed vector for $\mathbf{v}_{10}$ of network B. It has been computed as $\mathbf{v'}_{10} = \mathbf{v}_{r1} \cdot S_2 \cdot F_2$ for domain B (d=2, S_1=0, S_2=1, F_1=0, F_2=1.0205) from table 5.6.

In section 2.6 the linking equations (2.37) and (2.49), used for this chapter, were derived. The primary objective of the equations was to place the lowest errors with the highest weights under the assumption that the highest weights have the highest sensitivity towards the neuron output. This can be observed in table 5.7, where the general tendency is that larger weights have lower errors and smaller weights have higher errors. But with an increased acceptance angle, this objective might not be met. For instance vectors $\mathbf{v'}_{10}$ and $\mathbf{v'}_2$ with the highest acceptance angle of 12.94 ° do not fully comply with the primary objective. Their largest relative error lies with component $\mathbf{w'}_{11}$ of vector $\mathbf{v'}_{10}$, which is the second largest weight value in terms of absolute values. The same applies to component $\mathbf{w'}_{1B}$ of vector $\mathbf{v'}_2$, which is the largest weight with the largest error.

One conclusion of the analysis of relative errors is that the larger the angle between weight vectors, the larger the relative errors of individual vector components and the higher the possibility that the smallest weight will not have the largest error.

Without inducing changes into an existing weight matrix, the generalisation error will remain unchanged. Generally, changes will only worsen the recall accuracy if the network was over-trained. Consequently, if the intent is to improve the generalisation error, it is required that the linking process is inducing errors into the trained weight matrixes. Analysing where errors are induced and to which extent will assist the verification of the acceptance angle used. High errors in weight components suggest that the chosen acceptance angle φ may be to high and non-correlating vectors are linked. The error ceiling depends on the objective, if recall accuracy is insignificant and emphasis lies only with generalisation, tolerable errors will be high e.g. up to 200%, if recall accuracy is significant, tolerable errors will be much smaller e.g. below 50%. These margins have been determined by many practical experiments and are purely based on experience. The chosen acceptance angle φ determines the field in which the linked network performs best, which can be recall or generalisation.

Table 5.8 lists the induced vector length errors of the reconstructed vectors if compared against their original lengths from table 5.3. As identified previously in the vector component change impact analysis, the vector length change impact analysis reveals the same error distribution behaviour. The general tendency is that the larger vector in a linked vector pair carries the smaller error and the smaller vector carries the larger error with the exception of $\mathbf{v'}_{10}$. This vector's components suffer from increased relative errors, see table 5.7, and therefore has a higher vector length error than $\mathbf{v'}_2$.

Table 5.8 Vector length change impact analysis.

Vector	Original length	Reconstructed length		Relative Error
A: $\mathbf{v'}_2$	8.542784	8.510973	($\|v_{r1}\|$)	-0.37%
B: $\mathbf{v'}_{10}$	22.732605	22.732509	($\|v_{r1}\|*F_2$)	0.00%
A: $\mathbf{v'}_7$	27.700796	27.651269	($\|v_{r1}\|$)	-0.18%
B: $\mathbf{v'}_3$	5.631501	5.591230	($\|v_{r1}\|*F_2$)	-0.72%
A: $\mathbf{v'}_{10}$	12.186887	12.042148	($\|v_{r1}\|$)	-1.19%
B: $\mathbf{v'}_2$	3.761994	3.752044	($\|v_{r1}\|*F_2$)	-0.26%

Because the calculation of the vector length involves all vector components, high errors on proportionally small vector components are not impacting the vector length error to any large extent, keeping them relatively small. The search for the optimal acceptance angle for interpolation or extrapolation is solely determined by the generalisation error not errors of individual vector components. Errors in individual vector components demonstrate that induction of errors into a trained weight matrix will improve generalisation on over-trained networks.

5.4.2 Analysis of Linked Network

The linking of three neurons from two sub-networks each with 10 hidden neurons has reduced the overall number of hidden neurons from 20 to 17. This represents a network size reduction of 15%. Size reduction is one of many objectives of linking sub-networks. Since size reduction is depending on the chosen acceptance angle, the network will be trimmed to its optimum size for the field in which the linked network should perform.

In order to analyse the linking results without the necessity of training a stimuli network for classification, the individual weight matrixes for each knowledge domain A and B can be reconstructed in isolation by using equation (5.3), as introduced in section 5.4.1. To create a weight matrix where the original vectors after training are replaced with vectors involved in linking, vectors from table 5.3 are substituted with reconstructed vectors from table 5.7.

Because no stimuli network will be used, the generalisation data points are tested in the domain for which they were created. For example data points along the path of function A in figure 5.5 will be tested on the reconstructed weight matrix for network A. Whereby data points on the path of function B will be tested on the reconstructed weight matrix for network B.

With this analysis the recall and generalisation performances of each linked domain network can be directly compared to the trained domain network, providing valuable information on how both errors change.

Table 5.9 Weight matrixes of the hidden layers of networks A and B after reconstruction.

Reference	Reconstructed weight matrix for network A				Reconstructed weight matrix for network B			
	w_{11}	w_{12}	w_{1B}	Length	w_{11}	w_{12}	w_{1B}	Length
v_1	-31.4892	29.5203	1.7205	43.197	-27.7257	14.8999	1.5787	31.515
v_2	7.9869	-1.3691	-2.6021	8.510	-2.2584	-1.3729	2.6631	3.752
v_3	15.5734	-15.5594	4.6530	22.501	1.8900	3.7609	-3.6803	5.591
v_4	-27.8095	26.5554	1.1965	38.471	-1.1093	0.4377	1.6200	2.012
v_5	-8.1294	-5.7041	2.9063	10.348	4.1307	-22.998	2.6818	23.520
v_6	-3.4718	-3.1138	-0.1853	4.667	-20.9640	9.6942	1.382	23.138
v_7	-9.3471	-18.5994	18.2012	27.651	8.1379	-20.6553	0.6200	22.209
v_8	-6.6214	9.5011	-1.6889	11.703	-5.4202	-0.1151	0.1244	5.423
v_9	-18.7818	20.5240	-1.1448	27.844	-6.0104	-28.537	7.3615	30.079
v_{10}	7.2484	4.4066	-8.5473	12.042	21.3328	-3.6568	-6.9502	22.732

In table 5.9 the framed row entries on grey backgrounds are vectors that have been replaced with reconstructed vectors. With the resulting weight matrixes, evaluation of the recall accuracy and the generalisation errors can be repeated for networks A and B as shown in table 5.2 for the originally trained networks.

Table 5.10 Comparison between trained and linked network benchmarks.

Description	Network A		Network B	
	Trained	Linked	Trained	Linked
SSE_{trn}	0.02	0.584	0.019	0.237
SSE_{gen}	1.243	1.047	1.149	1.122
RMSE for SSE_{trn} on 10 training record sets	0.00667	0.19467	0.00633	0.079
RMSE for SSE_{gen} on 9 training record sets	0.41433	0.349	0.383	0.374

Re-evaluation on the weight matrixes with 3 linked neurons is shown in table 5.9 and has been undertaken with the same data used for the initial evaluation of the recall accuracy SSE_{trn} and the generalisation error SSE_{gen} during the training of the networks.

On inspection of table 5.10 it can be noted that the recall error has increased and the generalisation error has decreased because of the error induction of the linking process. Although the generalisation error has been reduced, it has not been reduced

to any significant amount so that generalisation confidence is boosted. This is because the generalisation data has been applied in their original domains and no advantage of knowledge domain intersection has been taken. For the re-evaluation of network A, generalisation data only from domain A has been used only on weights from domain A, without taking into account a possible degree of membership to domain B.
It might be the case that the generalisation data of domain A lies in the region outside the input space of the training data of domain A, therefore causing extrapolation problems and an increased generalisation error for domain A, as discussed in section 3.1. Hence, domain B can be included to assist in cases where the input space of an incoming input vector covers parts of the input space of domain B.
For this reason, classification by a stimuli network for the purpose of determining the degree of membership of unseen data for different domains will reduce the possibility of extrapolation errors, since it includes neurons from different domains to create an overall network output. Knowledge domain intersection with stimuli network was introduced in chapter 4 and will not be repeated at this point.

5.5 Numerical Experiment: Linking of Extrapolating Networks

Practical applications with neural networks very often involve finding functional relationships between variables. For such situations, neural networks can be used to transform training data into a mathematical data model equation, which is representing an approximated mathematical relationship called objective function between the training data input and output. The more variables or dimensions that are used for the problem representation, the more complex the objective function. To determine an equation that represents the objective function, a neural network must be trained to model the relationship between the input and output. Once training is completed, the data model, contained within the neural network weight matrix can be used for estimating data values missing from the training data set. Gaining access to results of unknown data values is one of the major utilisation of neural networks. If

good quality estimations of unknown data can be added to an incomplete database the end product will add value to the data, making this process a valuable business tool. Depending on the area of interest for the prediction, a neural network can be used for interpolation or occasionally for extrapolation of unknown data.

Interpolation is a mathematical procedure, which estimates values of a function at positions between listed or given values [161, 162]. Interpolation works by fitting a "curve", i.e. a function, to two or more given points and then applying this function to the input values of the prediction. Generally, the more parameters are used to model such an approximated curve, the more accurate its interpolation. But if too many free parameters are used, which are represented as hidden neurons in neural networks, the easier a network can over-train and interpolation will become unsatisfactory. Interpolation in relation to neural networks means that the unknown input vector lays inside, or interior to, the given training data, e.g. region C in figure 5.12.

Extrapolation is using a fitted curve for estimating a value of a variable outside a known data range, which has been used for creating the approximated curve. It is assuming that the estimated value follows logically from the known data values [163]. Extrapolation with neural networks is suffering from the same parameterisation problems as interpolation, where over-training can occur. Extrapolation in relation to neural networks means that the unknown input vector lies outside of, or exterior to, the given training data, e.g. region D in figure 5.12.

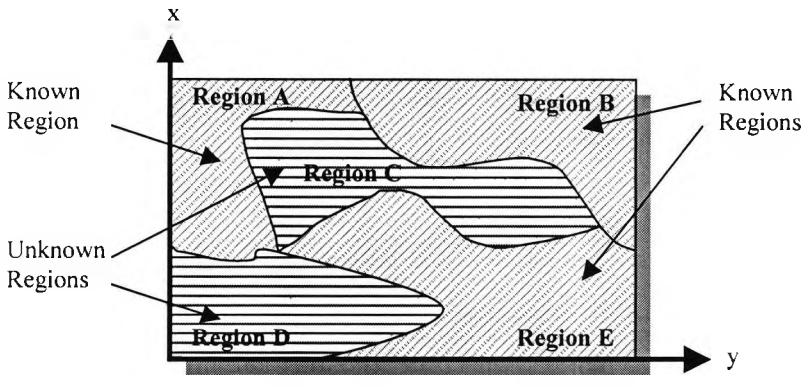


Figure 5.12: Training data input space shown as a 2D diagram.

Graphical analysis of training data input-space can be used to identify if certain regions of interest are encapsulated within known training data, e.g. region C, or outside known training data e.g. region D in figure 5.12. But this analysis is limited to 3-dimension. If more dimensions need to be analysed, a hyperspace navigation system of other mathematical analysis is required [164-166].

Because a trained neural network interpolates unknown areas, which are encapsulated by known data regions and extrapolates for areas outside known data regions, analysing the training data set can answer the question if the area in which the unknown values for prediction are located requires interpolation or extrapolation. This is an important question since in most cases interpolation can be performed with higher confidence than extrapolation [167-169].

Additionally to the objective function complexity, there are two major factors to be determined as a measure of an interpolation/extrapolation confidence. They are distance from the unknown data to the known training data and the known data granularity often referred to as data density [170, 171].

The purpose of this analysis is to train two neural networks each with data containing only selected clusters of the available input space and use linking to combine the trained networks into one network. The aim is to show how linking of neural networks trained with selected input space clusters will change the generalisation error on unseen extrapolated clusters not used for training. Depending on where the unseen testing data is located in the input space in relation to the training data, interpolation or extrapolation will be used to analyse the generalisation performance of the linked networks.

5.5.1 Clustering of Input Space

The two-dimensional input space $(x_1\ x_2)^T$ used for 2:10:1 neural networks are divided into 16 equally sized quadrants or clusters, as illustrated in figure 5.13. Each quadrant is defined by its position in the coordinate system and its boundaries with respect to x_1 and x_2, which are listed in table 5.11. Two numbers compose the numbering of the

clusters in clockwise direction. The first number represents the quadrant in which the cluster is located for Cartesian systems. The smallest second number 1 is located in the outside corner and the largest 4 in the inside corner, with 2 and 3 mirrored on each of the axis. With this arrangement, Qx1 represent all outside clusters, Qx4 all inside clusters and Qx2 and Qx3 clusters in-between.

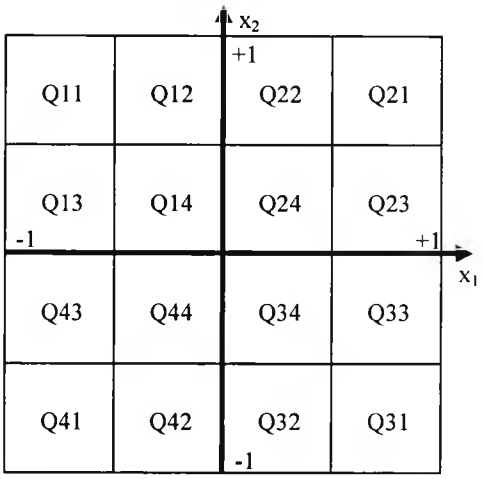


Figure 5.13 Input space divided into 16 equally sized quadrants.

Each cluster can be used for either training or testing but are mutually exclusive and no cluster used for training will be used for testing. By arranging the input space graphically into clusters, generalisation by interpolation or extrapolation can be easily distinguished, permitting categorisation of the linking results.

Table 5.11 The clusters and their range in input space.

Quadrant	x_1	x_2	Quadrant	x_1	x_2
Q11	$-1.0 < x_1 < -0.5$	$+1.0 < x_2 < +0.5$	Q13	$-1.0 < x_1 < -0.5$	$+0.5 < x_2 < 0.0$
Q12	$-0.5 < x_1 < 0.0$	$+1.0 < x_2 < +0.5$	Q14	$-0.5 < x_1 < 0.0$	$+0.5 < x_2 < 0.0$
Q21	$+1.0 < x_1 < +0.5$	$+1.0 < x_2 < +0.5$	Q23	$+1.0 < x_1 < +0.5$	$+0.5 < x_2 < 0.0$
Q22	$+0.5 < x_1 < 0.0$	$+1.0 < x_2 < +0.5$	Q24	$+0.5 < x_1 < 0.0$	$+0.5 < x_2 < 0.0$
Q31	$+1.0 < x_1 < +0.5$	$-1.0 < x_2 < -0.5$	Q33	$+1.0 < x_1 < +0.5$	$-0.5 < x_2 < 0.0$
Q32	$+0.5 < x_1 < 0.0$	$-1.0 < x_2 < -0.5$	Q34	$+0.5 < x_1 < 0.0$	$-0.5 < x_2 < 0.0$
Q41	$-1.0 < x_1 < -0.5$	$-1.0 < x_2 < -0.5$	Q43	$-1.0 < x_1 < -0.5$	$-0.5 < x_2 < 0.0$
Q42	$-0.5 < x_1 < 0.0$	$-1.0 < x_2 < -0.5$	Q44	$-0.5 < x_1 < 0.0$	$-0.5 < x_2 < 0.0$

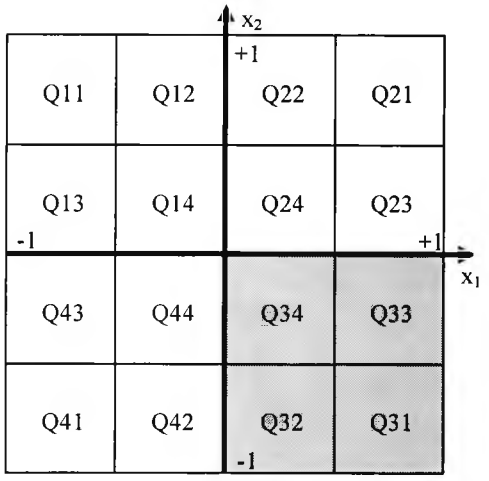


Figure 5.14 Dataset A for extrapolation.

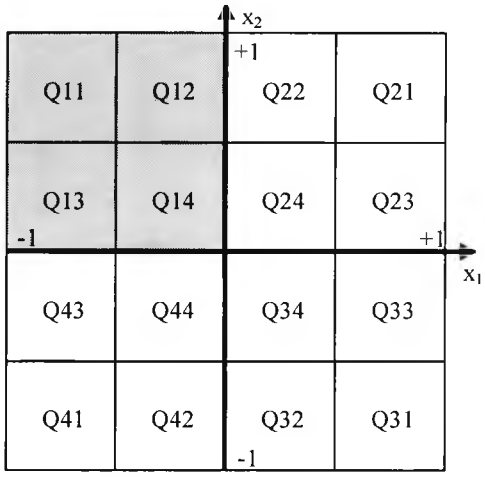


Figure 5.15 Dataset B for extrapolation.

This section is analysing how generalisation performance is changing if two extrapolating networks are linked. For this reason the training data used for each network is located in the corners of the input space, causing forced extrapolation for each unseen quadrant. Extrapolation is forced because all unseen clusters are exterior to the clusters used for training.

Network A is trained with dataset A from figure 5.14, which is including all clusters of quadrant Q3x. Network B is trained with dataset B, which is including all clusters of quadrant Q1x.

5.5.2 Training of Domain Networks

The training and testing data will consist of two inputs x_1 and x_2 and one output o_1. It's objective function $o_1 = f(x_1, x_2)$ is given in equation (5.5) and is valid for the entire input space ($0<x_1<1$; $0<x_2<1$). The chosen objective function shown in figure 5.16 is non-linear and continuous for the entire input space, thus permitting good generalisation for interpolation. The graph has been rotated anti-clockwise by 90° for better visualisation because the steepness of Q4x would otherwise cover all other

quadrants. For easier orientation all quadrants from figure 5.13 have been rotated and projected on top of the graph. It can be noted that Q41 has the highest target value of approximately 0.80 and Q31 the lowest target value of approximately –0.76.

Training and testing data records have been generated with random values of x_1 and x_2, within the cluster boundary constraints of table 5.11, and the target value has been calculated with equation (5.5).

The training data of selected clusters has been used to train neural networks, which act as domain experts for each of the clusters used. Networks A and B have been trained with the same training parameters and stopping criteria from section 5.3.2 and are shown in table 5.1. The stopping criteria chosen to end training for both networks has been that the SSE remained unchanged for 1000 training iterations in batch update mode. This stopping criterion used represents the location of a minimum of the error function.

$$y = 0.55 \cdot e^{-\frac{x_1^2}{5}} \cdot \{[(x_1 - 0.26) \cdot (x_2 - 0.73)] - 0.4\} \tag{5.5}$$

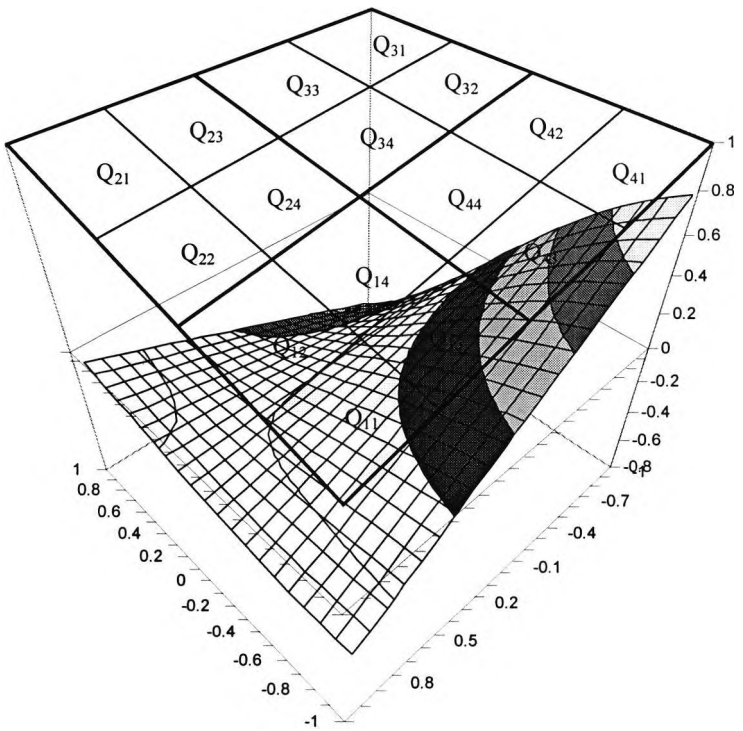


Figure 5.16 The objective function of the output for networks A and B.

After training each of the 16 clusters have been presented to each of the networks for measuring its SSE with respect to the target values. Tables 5.12 and 5.13 show the SSE for each cluster for networks A and B respectively and figures 5.17 and 5.18 present the data graphically in 2D and figures 5.19 and 5.20 in 3D.

It can be noted that the clusters used for training have low errors. Network A has been trained with all data from quadrant Q3. Therefore clusters Q31, Q32, Q33 and Q34 have the lowest errors. After inspection of table 5.12, it can be seen that quadrant Q1 and especially Q11 have the highest error. This is because cluster Q11 has the largest distance from quadrant Q3.

Table 5.12 Sum Square Errors for each cluster after training of network A.

Cluster	SSE	Cluster	SSE
Q11	5.28756	Q31	0.04748
Q12	2.13071	Q32	0.01790
Q13	3.55409	Q33	0.03569
Q14	1.18332	Q34	0.02579
Q21	0.52109	Q41	0.96923
Q22	0.30077	Q42	0.25616
Q23	0.07540	Q43	1.74874
Q24	0.10328	Q44	0.54570

The distance of Q21 and Q41 to quadrant 3 is the same but the SSE for Q21 (0.52) and for Q41 (0.96) differs substantially, see table 5.12. The explanation for this is that the target value of Q21 (-0.25) is lower than the target value of Q41 (0.8). High target output values require an even higher net_j within a neuron because of the squashing characteristic of the sigmoid activation function and are therefore more difficult to extrapolate.

On inspection of table 5.13, which shows the SSE for each cluster for network B, the same extrapolation behaviour as found for network A can be noted. Here, quadrant Q1 has been used for training, causing the lowest errors in Q11, Q12, Q13 and Q14. Cluster Q31 has the furthest distance to quadrant Q1 and therefore the highest error and Q41 has a much higher error than Q21 because of the higher target value.

Table 5.13 Sum Square Errors for each cluster after training of network B.

Cluster	SSE	Cluster	SSE
Q11	0.01680	Q31	5.24866
Q12	0.00758	Q32	2.41018
Q13	0.01542	Q33	2.49046
Q14	0.00774	Q34	0.88148
Q21	0.52966	Q41	1.32985
Q22	0.12068	Q42	0.52379
Q23	0.73015	Q43	0.35050
Q24	0.16115	Q44	0.12920

As a result, it can be said that the larger the distance Δd of a cluster for extrapolation is from the training data, the higher its error and the lower its confidence. Additionally, the higher the expected target value |t| of the extrapolation, the higher its error and the lower its confidence. With this, a simple proportional equation for a confidence measure c can be derived as shown in equation (5.6).

$$c \sim \frac{1}{\Delta d \cdot |t|} \tag{5.6}$$

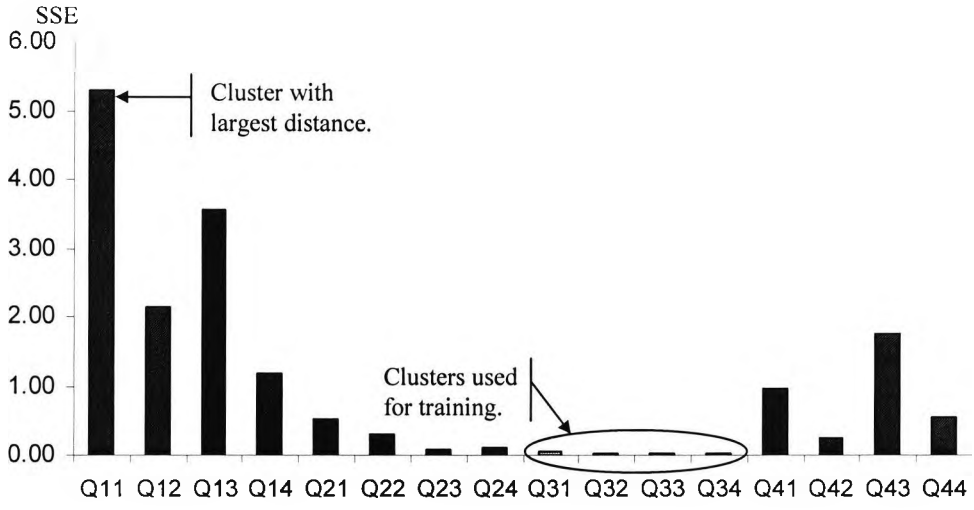


Figure 5.17 SSE for each cluster after training from figure 5.14.

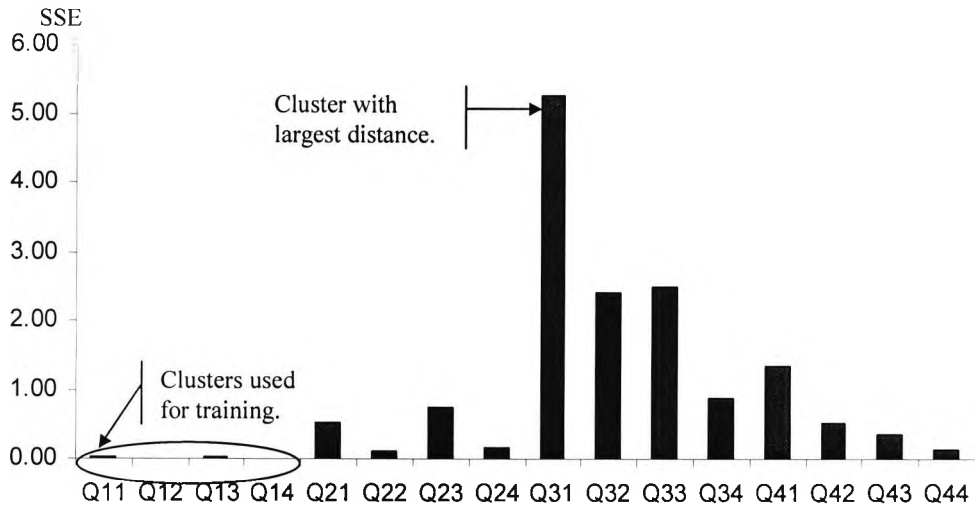


Figure 5.18 SSE for each cluster after training from figure 5.15.

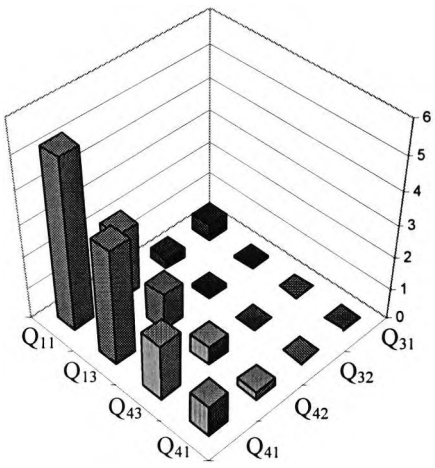


Figure 5.19 3D representation of figure 5.17.

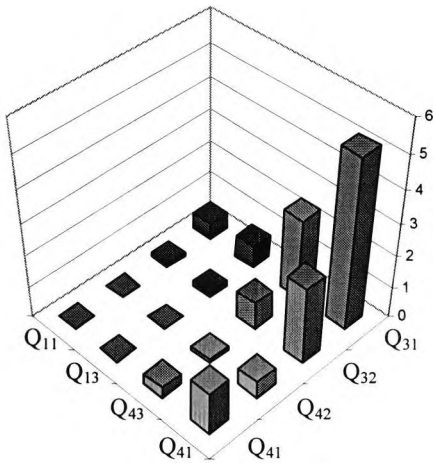


Figure 5.20 3D representation of figure 5.18.

The 3D SSE representation of all clusters in figures 5.19 and 5.20 for networks A and B shows that the SSE increases with distance Δd and that Q41 is larger than Q21 located at the opposite corner. In the following section, both networks will be linked

and all SSE for each cluster will be measured again and compared with the results of this section.

5.5.3 Linking of Domain Networks

In order to combine the knowledge of each domain expert networks A and B are now linked as described in detail in sections 5.3.3 to 5.3.5. An acceptance angle of 10° has been used for linking of both domains. This value has been chosen empirically from experience because the intention is not to optimise the generalisation error as was the case in section 3.4.2 in figure 3.12, instead the impact of linking on interpolation and extrapolation are investigated.

Table 5.14 shows the trained weight matrixes of networks A and B. To find suitable vectors for linking, each vector from one domain is compared against the vectors of the other domain as described in section 3.3.2 and equation (5.1). The three vectors listed in table 5.15 have an acceptance angle below 10 ° and are therefore suitable for linking. Vector pairs $\mathbf{v}_7$, $\mathbf{v}_3$ and $\mathbf{v}_6$ and $\mathbf{v}_1$ are pointing in opposite directions and are mapped from quadrant Q3 to Q1 as indicated in section 3.3.2.

Table 5.14 Weight matrixes of the hidden layers of networks A and B after training.

Reference	Network A				Network B			
	w_{11}	w_{12}	w_{1B}	Length	w_{11}	w_{12}	W_{1B}	Length
v_1	-0.83348	-0.90902	-0.91949	1.53833	1.96033	-1.33922	-0.45051	2.41648
v_2	0.50862	0.53704	0.18746	0.76305	-0.68043	-0.04933	-0.09201	0.68839
v_3	0.46481	-0.12644	0.05564	0.48490	0.12802	1.23932	0.75058	1.45453
v_4	0.46683	-0.05713	0.73720	0.87444	0.21231	0.76366	-0.42105	0.89752
v_5	-1.62017	-0.82645	-0.75155	1.96794	-0.69594	0.83353	1.62410	1.95367
v_6	-1.93530	1.56101	0.09489	2.48820	0.42754	0.30195	0.02316	0.52393
v_7	0.03216	-0.42111	-0.25007	0.49082	-0.98202	-0.55862	-0.71396	1.33648
v_8	0.58610	-0.41012	0.81301	1.08291	-1.29074	-0.83226	-0.76685	1.71660
v_9	-0.64106	-0.85676	-0.84200	1.36160	-1.74247	-1.31733	-1.04761	2.42261
v_{10}	1.28727	0.32535	-0.35149	1.37348	-0.60833	0.88289	1.57870	1.90836

Table 5.15 Angles between weight vectors in ascending order.

Vector pair		Angle between vectors
Network A	Network B	
v_5	v_8	6.66°
v_7	v_3	180°- 171.18= 8.82° [1]
v_6	v_1	180°- 170.32 = 9.68° [1]

Since weight matrixes contain domain knowledge specific to the area they were trained for, a knowledge domain map should be supplied, as shown in table 5.4. This is particularly important if weight matrixes are exchanged without the presence of training data. In table 5.16 the Knowledge Domain Maps (KDMs) for each of the weight matrixes have been represented as the boundaries of their training data quadrants Q3 for network A and Q1 for network B.

Table 5.16 Knowledge Domain Maps for both training data sets.

Network	Area which matrixes were trained for
Matrix A	$(0<x_1<1)$ and $(-1<x_2<0)$
Matrix B	$(-1<x_1<0)$ and $(0<x_2<1)$

Linking of vectors listed in table 5.15 follows the approach derived in section 2.6 and the results are shown in table 5.17. Factor F2 is negative for the second and third vector because of the mapping from Q3 to Q1. The next section is a short analysis of the impact the linking has on the weights if the trained weight matrix is compared against the linked weight matrix.

Table 5.17 Linking results of vectors with angles below 10° as listed in table 5.15.

Original vector references		Resulting vector v_{r1}			
Network A	Network B	w_{11}	w_{12}	w_{1B}	Factor F_2
v_5	v_8	-1.56289	-0.87932	-0.80501	0.87126
v_7	v_3	-0.02173	-0.41842	-0.24989	-2.97867
v_6	v_1	-1.83004	-0.11512	0.00360	-1.02160

[1] This vector has been mapped from Q3 to Q1.

5.5.4 Linking Analysis

Linking analysis was introduced in detail in section 3.4 and all items discussed are applicable to this section. Table 5.18 is equivalent to table 3.6 and shows that the relative errors between the weights are generally lowest for large weights and largest for small weights and that the error increases as the angle between the linked vectors grows.

Table 5.18 Vector component change impact analysis.

Reconstructed Vectors	Vector components			Relative Errors		
	w'_{11}	w'_{12}	w'_{1B}	$f(w_{11}, w'_{11})$	$f(w_{12}, w'_{12})$	$f(w_{1B}, w'_{1B})$
A: v'_5	-1.56289	-0.87932	-0.80501	-3.54%	6.40%	7.11%
B: v'_8	-1.36168	-0.76611	-0.70137	5.50%	-7.95%	-8.54%
A: v'_7	-0.02174	-0.41842	-0.24989	-167.58%	-0.64%	-0.07%
B: v'_3	0.06474	1.24633	0.74433	-49.43%	0.57%	-0.83%
A: v'_6	-1.83004	-0.11512	0.00360	-5.44%	-107.37%	-96.21%
B: v'_1	1.86956	0.11761	-0.00368	-4.63%	-108.78%	-99.18%

Table 5.19 is equivalent to table 3.7 and shows that the relative errors between the vector lengths are generally lowest on linked vectors with small angle differences and increases as the angle between the vectors grows.

Table 5.19 Vector length change impact analysis.

Vector	Original length	Reconstructed length		Relative Error
A: v'_5	1.96794	1.96567	($\|v_{r1}\|$)	-0.12%
B: v'_8	1.71660	1.71261	($\|v_{r1}\|*F_2$)	-0.23%
A: v'_7	0.49082	0.48784	($\|v_{r1}\|$)	-0.61%
B: v'_3	1.45453	1.45312	($\|v_{r1}\|*F_2$)	-0.10%
A: v'_6	2.48820	1.83366	($\|v_{r1}\|$)	-26.31%
B: v'_1	2.41648	1.87326	($\|v_{r1}\|*F_2$)	-22.48%

The high linking error between vectors $\mathbf{v}_6$ and $\mathbf{v}_1$ are an indication that the recall accuracy will be considerably reduced. This has been caused by network over-training

and is unavoidable if better generalisation is desired. Improving the generalisation error that causes loss of recall accuracy has been discussed in the previous chapters. Table 5.20 is equivalent to table 3.9 and shows the reconstructed weight matrix for domains A and B after linking. In table 5.20 the framed row entries on grey backgrounds are vectors that have been replaced with reconstructed vectors. With the resulting weight matrixes, evaluation of each of the input space clusters from figure 5.13 can be repeated for networks A and B as shown in tables 5.12 and 5.13 for the originally trained networks. The next section repeats the process of measuring all SSE for all clusters as shown in section 5.5.2 for the linked weight matrixes of table 5.20.

Table 5.20 Weight matrixes of the hidden layers of networks A and B after reconstruction.

Reference	Reconstructed weight matrix for network A				Reconstructed weight matrix for network B			
	w_{11}	w_{12}	w_{1B}	Length	w_{11}	w_{12}	w_{1B}	Length
v_1	-0.83348	-0.90902	-0.91949	1.53833	1.86956	0.11761	-0.00368	1.87326
v_2	0.50862	0.53704	0.18746	0.76305	-0.68043	-0.04933	-0.09201	0.68839
v_3	0.46481	-0.12644	0.05564	0.48490	0.06474	1.24633	0.74433	1.45312
v_4	0.46683	-0.05713	0.73720	0.87444	0.21231	0.76366	-0.42105	0.89752
v_5	-1.56289	-0.87932	-0.80501	1.96567	-0.69594	0.83353	1.62410	1.95367
v_6	-1.83004	-0.11512	0.00360	1.83366	0.42754	0.30195	0.02316	0.52393
v_7	-0.02174	-0.41842	-0.24989	0.48784	-0.98202	-0.55862	-0.71396	1.33648
v_8	0.58610	-0.41012	0.81301	1.08291	-1.36168	-0.76611	-0.70137	1.71261
v_9	-0.64106	-0.85676	-0.84200	1.36160	-1.74247	-1.31733	-1.04761	2.42261
v_{10}	1.28727	0.32535	-0.35149	1.37348	-0.60833	0.88289	1.57870	1.90836

5.5.5 Linking Results

To evaluate the impact of linking, if the networks involved are used for extrapolation, the trained weight matrixes have been replaced with the linked weight matrixes from table 5.20. The impact of linking has been measured with regards to the SSE for each cluster of the entire input space. By using the linked weight matrixes, the errors for each cluster can increase or decrease if compared with the trained SSE results listed in tables 5.12 and 5.13 for networks A and B respectively. Table 5.21 contains the SSE

for each cluster for network A and table 5.22 for network B. It can be noted that the linking process has reduced the extrapolation errors with the furthest distance; quadrant Q1 in table 5.21 for network A and quadrant Q3 in table 5.22 for network B. Quadrant Q4 shows a slight improvement for network A but worsened for network B.

Table 5.21 Sum Square Errors for each cluster after linking of dataset A.

Cluster	SSE	Cluster	SSE
Q11	2.55619	Q31	0.08766
Q12	0.48926	Q32	0.04759
Q13	1.72065	Q33	0.04364
Q14	0.25082	Q34	0.03166
Q21	0.38004	Q41	1.08590
Q22	0.48450	Q42	0.22921
Q23	0.06479	Q43	0.79075
Q24	0.19843	Q44	0.17832

Table 5.22 Sum Square Errors for each cluster after linking of dataset B.

Cluster	SSE	Cluster	SSE
Q11	0.35056	Q31	4.44658
Q12	0.23728	Q32	2.03295
Q13	0.36733	Q33	1.80917
Q14	0.28527	Q34	0.55317
Q21	0.82563	Q41	1.92059
Q22	0.29750	Q42	0.66472
Q23	0.33802	Q43	0.74010
Q24	0.20874	Q44	0.42719

For easier visualisation, figures 5.21 and 5.22 compare the SSE from training with the SSE after linking by utilisation of error bars. The horizontal marker of the error bars represents the SSE after training and the bar graph the SSE after linking.

Linking of networks of two distinct domains has reduced the extrapolation error for quadrants included in the training data of the other network. Quadrant Q1 used to train network A has been reduced in SSE error by linking in network B and quadrant Q3 used to train network B has been reduced in SSE error in network A after linking. The combination of knowledge between both networks has been successful only for

areas they were trained in but not for areas not contained in both networks. It can be observed in figure 5.21 that Q43 and Q44 slightly reduced their SSE after linking but quadrant Q2 worsened. But in figure 5.22 quadrants Q2 and Q4 increased their SSE after linking.

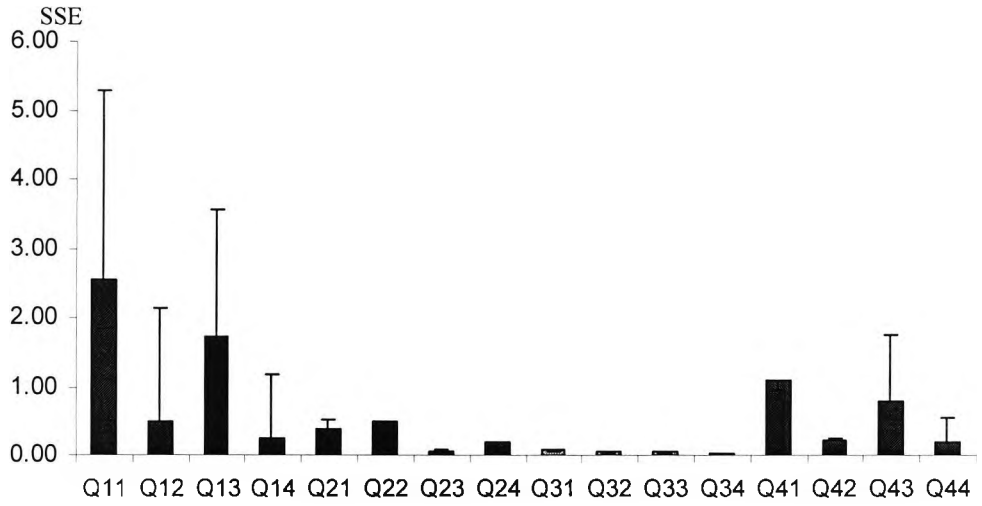


Figure 5.21 Error distribution for each cluster after linking 3 neurons of clusters from figure 5.15.

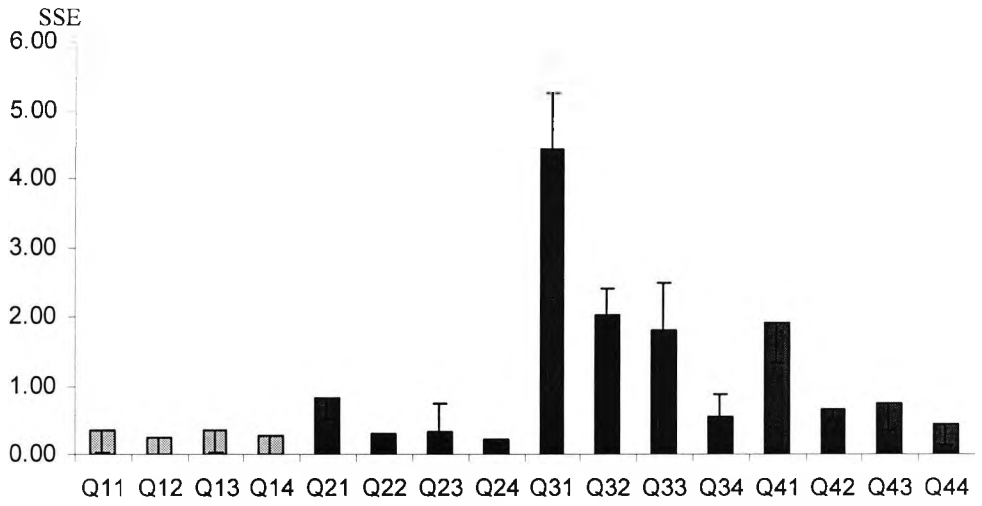


Figure 5.22 Error distribution for each cluster after linking 3 neurons of clusters from figure 5.16.

For an area view of figures 5.21 and 5.22, figures 5.23 and 5.24 are representing the SSE measures in 3D.

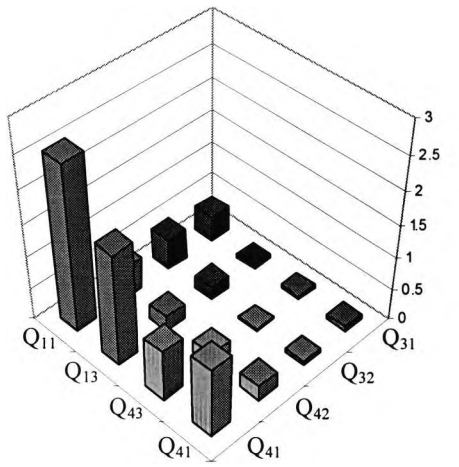


Figure 5.23 Cluster errors after linking for A.

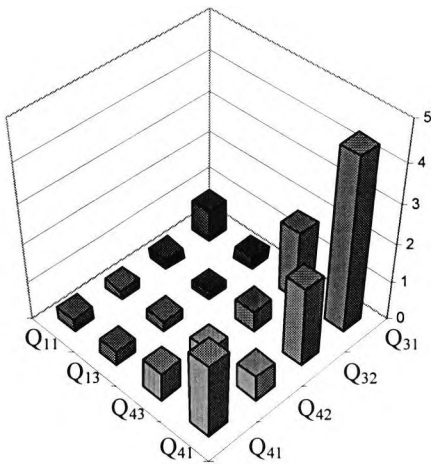


Figure 5.24 Cluster errors after linking for B.

It can be concluded that the linking of two networks trained for generalisation in extrapolated areas will reduce the SSE for areas, which were available for training. It is evident that after linking the confidence c for a cluster increases even if the relative distance Δd to the closest cluster used for training is large. If P denotes the probability that a cluster for extrapolation is included in a domain for linking, equation (5.6) can be extended to form equation (5.7).

$$c \sim \frac{P}{\Delta d \cdot |t|} \tag{5.7}$$

With Δd as the distance between the cluster for extrapolation and the nearest cluster used for training, | t | as the expected target value of the extrapolation and P as the probability that the cluster for extrapolation is present in any other domain included in the linking process.

5.6 Numerical Experiment: Linking of Inter- and Extrapolating Networks

In the previous example in section 5.5, two networks trained for extrapolation have been linked and their changes in extrapolation have been analysed. In this section, two networks, one for interpolation and one for extrapolation are linked to investigate the impact linking can have on the interpolation and extrapolation errors of each network. Interpolation requires unseen testing data to be located in region C and extrapolation requires unseen testing data to be located in region D as illustrated in figure 5.12.
This numeric example will follow exactly the same process as the previous example in section 5.5. The only difference is the composition of clusters used for training and testing for each of the networks involved, which in turn defines if a network is used for interpolation or extrapolation.

5.6.1 Clustering of Input Space

The total number of sectors that divide the input space into clusters and their numbering are the same as introduced from section 5.5.1 and were illustrated in figure 5.13. With this, only the clusters used for training and testing of both domain networks C and D must be changed according to their purpose of interpolation or extrapolation.
This section is analysing how generalisation performance is changing if an interpolating and an extrapolating network are linked. For this reason the training data used for the interpolating network C is located in each of the corners of the input space as indicated in figure 5.25, leaving all other clusters of unseen testing data for generalisation surrounded by data. Network D is extrapolating and therefore has its training data in a diagonal cluster formation as indicated in figure 5.26 thus, leaving all other clusters of unseen testing data for generalisation outside the training data area.

In terms of confidence, unseen clusters of testing data with the furthest distance to the clusters used for training are Q21 and Q41. These clusters will have the lowest confidence and are therefore of prominent interest. Because Q41 has a higher target value than Q21 the extrapolation error of Q21 should be lower than the error of Q41.

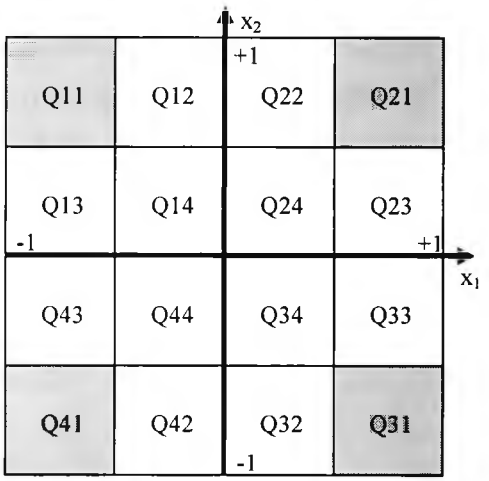


Figure 5.25 Dataset C for interpolation.

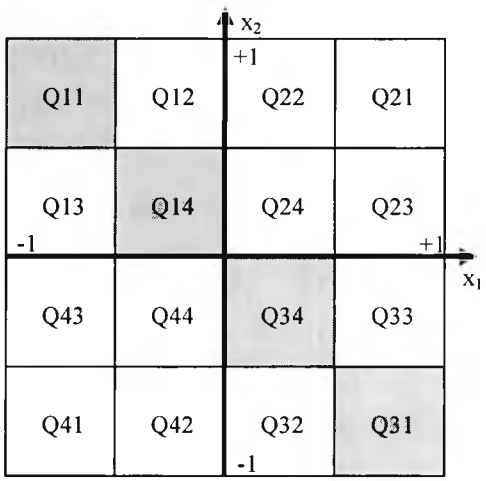


Figure 5.26 Dataset D for extrapolation.

5.6.2 Training of Domain Networks

The training and testing objective function used in this section is exactly the same as previously used in section 5.5.2, which is illustrated in figure 5.16 and expressed in equation (5.5).

Both domain networks C and D used are equivalent to the ones used in the previous section 5.5.2, which are 2:10:1 backpropagation networks with a symmetric sigmoid activation function and frozen output weights that are set to 1.

Because the data contained in each of the clusters remained unchanged, the same training parameters with a learning factor of 0.7 and a momentum of 0.5 as presented in section 5.3.2 table 5.1 have been used.

After training each of the 16 clusters have been presented to each of the networks for measuring its SSE with respect to the target values. Tables 5.23 and 5.24 show the

SSE for each cluster for networks C and D respectively and figures 5.27 and 5.28 present the data graphically in 2D and figures 5.29 and 5.30 in 3D.

Table 5.23 Sum Square Errors for each cluster after training of network C.

Cluster	SSE	Cluster	SSE
Q11	0.01543	Q31	0.01644
Q12	0.02860	Q32	0.03608
Q13	0.07019	Q33	0.02550
Q14	0.04451	Q34	0.01569
Q21	0.01548	Q41	0.02478
Q22	0.04949	Q42	0.08384
Q23	0.02615	Q43	0.06368
Q24	0.03627	Q44	0.04345

Table 5.24 Sum Square Errors for each cluster after training of network D.

Cluster	SSE	Cluster	SSE
Q11	0.02270	Q31	0.03522
Q12	0.04658	Q32	0.14554
Q13	0.05418	Q33	0.10358
Q14	0.01799	Q34	0.01826
Q21	1.21126	Q41	1.65331
Q22	0.38512	Q42	0.53847
Q23	0.40224	Q43	0.38204
Q24	0.09621	Q44	0.06228

It can be noted that the clusters used for training have low errors. Network C has been trained with data located in the corners Q11, Q21, Q31 and Q41. Therefore these clusters have the lowest errors. Clusters used for training are represented in a lighter grey colour in figure 5.27 and 5.28 than unseen clusters.

In table 5.23 all interpolation errors are relatively low if compared to the extrapolation errors in table 5.24, with none of them exceeding 0.1. The main reason for this is that the objective function from equation (5.5) is very smooth and can therefore be described quite accurately by a network with 10 hidden neurons, thus permitting excellent interpolation.

In table 5.24, on the other hand, all extrapolation errors are quite large with quadrant Q41 and Q21 boasting the highest errors. Even if clusters Q41 and Q21 have the same distance Δd to the training data, Q41 has a higher extrapolation error because its target value | t | is larger (0.96) than the target value of Q21 (0.52).

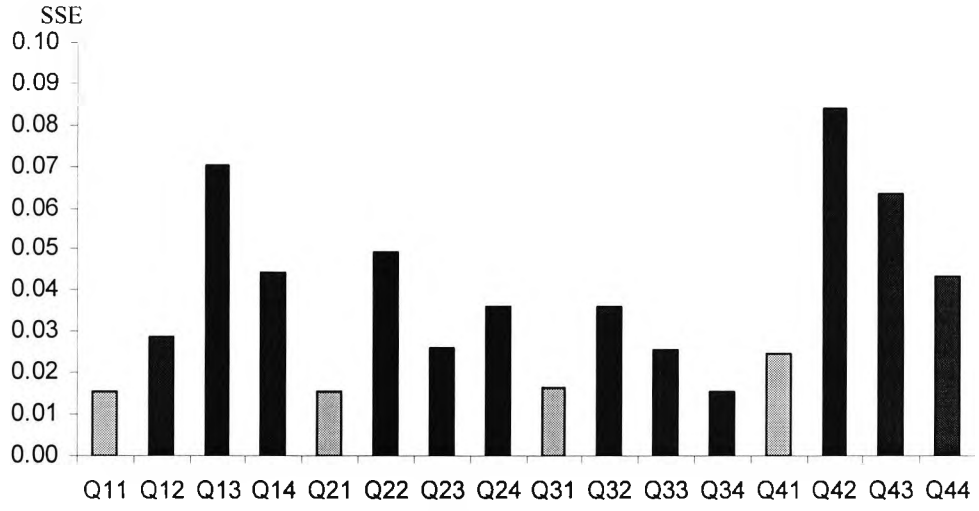


Figure 5.27 Error distribution for each cluster after training of clusters from figure 5.25.

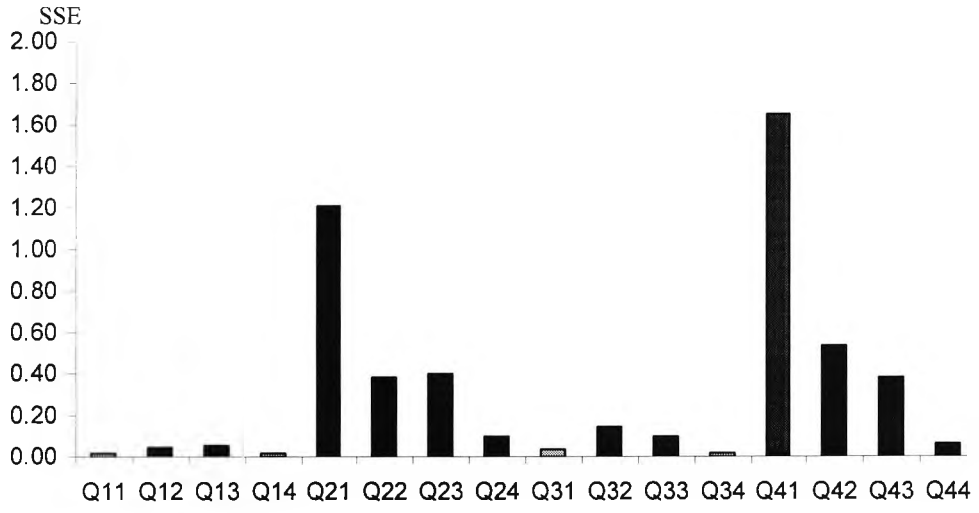


Figure 5.28 Error distribution for each cluster after training of clusters from figure 5.16.

Figures 5.29 and 5.30 show the spatial representation of the cluster errors across the input space from figures 5.27 and 5.28 in 3D. It can be noted that the interpolation errors shown in figure 5.29 are higher than the errors of the clusters used for training (Q11, Q21, Q31 and Q41). The highest interpolation error of 0.083 can be found in cluster Q42, followed by Q13 (0.07) and Q43 (0.063). Figure 5.30 shows the extrapolation error distribution, which is highest at the clusters Q41 (1.65) and Q21 (1.21) with the largest distance Δd to the training data.

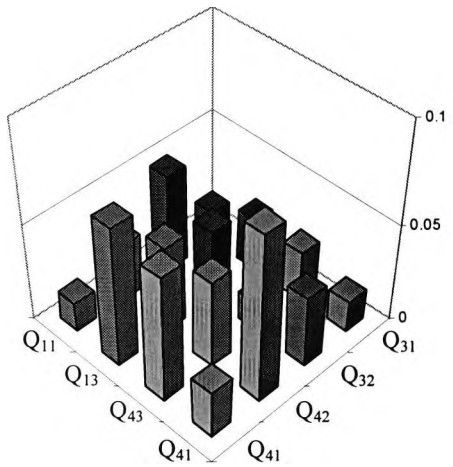


Figure 5.29 Cluster errors after training for A.

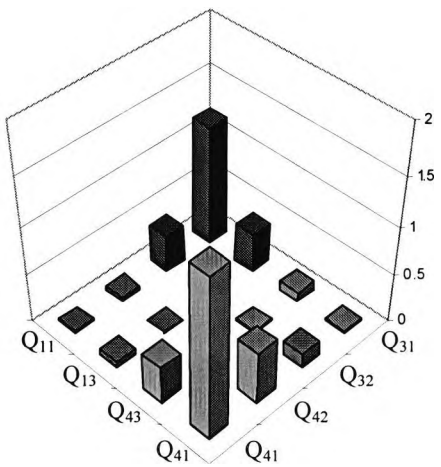


Figure 5.30 Cluster errors after training for B.

5.6.3 Linking of Domain Networks

To combine the knowledge of both domain experts, networks C and D are now linked as described in section 5.5.3. As in previous chapters, an acceptance angle of 10° has been used for linking of both domains. The linking process follows exactly the same process as demonstrated in section 5.5.3 and will therefore be described in brief.

Table 5.25 shows the trained weight matrix of networks C and D and table 5.26 shows the vectors, which have an angle difference below the acceptance angle. All four vectors listed in table 5.26 qualify for linking and table 5.27 shows the resulting vectors and their associated length correction factor F_2.

It can be observed that both training domains have intersecting clusters Q11 and Q31 in datasets C and D. Because of this, more neurons contain similar information and are pointing in the same direction in hyperspace. Therefore more neurons have been identified for linking than in the previous example in section 5.5.

Table 5.25 Weight matrixes of the hidden layers of networks A and B after training.

Reference	Network A				Network B			
	w_{11}	w_{12}	w_{1B}	Length	w_{11}	w_{12}	w_{1B}	Length
v_1	0.01814	0.12714	0.32884	0.35303	-0.20192	0.10318	-0.68289	0.71955
v_2	-1.45389	0.74088	1.61548	2.29619	-0.30162	0.03888	-0.64425	0.71242
v_3	-0.64706	-0.47454	-0.33795	0.87068	0.43947	-0.48111	0.25449	0.69954
v_4	-1.61199	1.73849	4.55330	5.13356	-0.29893	0.00092	-0.81110	0.86443
v_5	-2.51647	1.80877	5.94466	6.70397	-1.18565	1.56259	2.60402	3.26012
v_6	-1.46253	-2.22265	-4.75988	5.45304	-3.38593	2.23422	7.43186	8.46692
v_7	1.80993	-1.22117	2.55825	3.36329	2.02362	-0.98535	2.57601	3.42079
v_8	-1.45224	-0.77717	-1.00396	1.92897	-0.62344	-0.36404	-1.16696	1.37222
v_9	2.04481	1.01383	-2.60103	3.46042	0.14505	1.12559	0.76522	1.36878
v_{10}	-3.30301	-3.13889	-7.48923	8.76647	-1.87252	-1.64221	-1.31091	2.81454

Table 5.26 Angles between weight vectors in ascending order.

Vector pair		Angle between vectors
Network A	Network B	
v_5	v_6	1.53°
v_7	v_7	180°- 174.70= 5.30° [1]
v_3	v_{10}	180°- 173.32= 6.67° [1]
v_{10}	v_8	6.93°

Table 5.27 Linking results of vectors with angles below 10° as listed in table 5.26.

Original vector references		Resulting vector v_{r1}			
Network A	Network B	w_{11}	w_{12}	w_{1B}	Factor F_2
v_5	v_6	-2.62866	1.78183	5.90283	1.26314
v_7	v_7	1.90235	-1.10403	2.53995	1.01694
v_3	v_{10}	-0.57040	-0.50546	-0.40698	3.25751
v_{10}	v_8	-3.24420	-3.21372	-7.48258	0.15518

[1] This vector has been mapped from Q3 to Q1.

5.6.4 Linking Analysis

Table 5.28 shows the relative errors between the trained weights and the weights after linking. It can be observed that the relative errors are generally lowest for large weights and largest for small weights and that the error increases as the angle between the linked vectors grows. All relative errors are comparatively low with only five weights being above 10%. Similarly in table 5.29, which lists the relative errors of the vector length, where none of the errors exceeds 1%.

Table 5.28 Vector component change impact analysis.

Reconstructed Vectors	Vector components			Relative Errors		
	w'_{11}	w'_{12}	w'_{1B}	$f(w_{11}, w'_{11})$	$f(w_{12}, w'_{12})$	$f(w_{1B}, w'_{1B})$
A: v'_5	-2.62866	1.78183	5.90283	4.46%	-1.49%	-0.70%
B: v'_6	-3.32035	2.25070	7.45608	-1.94%	0.74%	0.33%
A: v'_7	1.90235	-1.10403	2.53995	5.11%	-9.59%	-0.72%
B: v'_7	1.93458	-1.12274	2.58298	-4.40%	13.94%	0.27%
A: v'_3	-0.57040	-0.50546	-0.40698	-11.85%	6.52%	20.43%
B: v'_{10}	-1.85810	-1.64653	-1.32575	-0.77%	0.26%	1.13%
A: v'_{10}	-3.24420	-3.21372	-7.48258	-1.78%	2.38%	-0.09%
B: v'_8	-0.50343	-0.49871	-1.16115	-19.25%	36.99%	-0.50%

Table 5.29 Vector length change impact analysis.

Vector	Original length	Reconstructed length		Relative Error
A: v'_5	6.70397	6.70285	($\|v_{r1}\|$)	-0.02%
B: v'_6	8.46692	8.46662	($\|v_{r1}\|*F_2$)	0.00%
A: v'_7	3.36329	3.35993	($\|v_{r1}\|$)	-0.10%
B: v'_7	3.42079	3.41686	($\|v_{r1}\|*F_2$)	-0.11%
A: v'_3	0.87068	0.86399	($\|v_{r1}\|$)	-0.77%
B: v'_{10}	2.81454	2.81446	($\|v_{r1}\|*F_2$)	0.00%
A: v'_{10}	8.76647	8.76595	($\|v_{r1}\|$)	-0.01%
B: v'_8	1.37222	1.36030	($\|v_{r1}\|*F_2$)	-0.87%

[1] This vector has been mapped from Q3 to Q1.

Table 5.30 shows the reconstructed weight matrix for domains C and D after linking, where the framed row entries on grey backgrounds are vectors that have been replaced with reconstructed vectors. With the resulting weight matrixes, evaluation of each of the input space clusters from figure 5.13 can be repeated for networks C and D as shown in table 5.23 and 5.24 for the originally trained networks.

Table 5.30 Weight matrixes of the hidden layers of networks A and B after reconstruction.

Reference	Reconstructed weight matrix for network A				Reconstructed weight matrix for network B			
	w_{11}	w_{12}	w_{1B}	Length	w_{11}	w_{12}	w_{1B}	Length
v_1	0.01814	0.12714	0.32884	0.35303	-0.20192	0.10318	-0.68289	0.71955
v_2	-1.45389	0.74088	1.61548	2.29619	-0.30162	0.03888	-0.64425	0.71242
v_3	-0.57040	-0.50546	-0.40698	0.86399	0.43947	-0.48111	0.25449	0.69954
v_4	-1.61199	1.73849	4.55330	5.13356	-0.29893	0.00092	-0.81110	0.86443
v_5	-2.62866	1.78183	5.90283	6.70285	-1.18565	1.56259	2.60402	3.26012
v_6	-1.46253	-2.22265	-4.75988	5.45304	-3.32035	2.25070	7.45608	8.46662
v_7	1.90235	-1.10403	2.53995	3.35993	1.93458	-1.12274	2.58298	3.41686
v_8	-1.45224	-0.77717	-1.00396	1.92897	-0.50343	-0.49871	-1.16115	1.36030
v_9	2.04481	1.01383	-2.60103	3.46042	0.14505	1.12559	0.76522	1.36878
v_{10}	-3.24420	-3.21372	-7.48258	8.76595	-1.85810	-1.64653	-1.32575	2.81446

5.6.5 Linking Results

The impact of linking of an interpolating and an extrapolating network has been measured with respect of the SSE changes in each cluster. Table 5.31 and table 5.32 list the SSE error of each cluster for networks C and D.

Table 5.31 Sum Square Errors for each cluster after linking of network C.

Cluster	SSE	Cluster	SSE
Q11	0.36684	Q31	0.03593
Q12	0.18797	Q32	0.07329
Q13	0.39388	Q33	0.05525
Q14	0.21208	Q34	0.12181
Q21	0.02651	Q41	0.29644
Q22	0.11790	Q42	0.15367
Q23	0.05978	Q43	0.38837
Q24	0.14428	Q44	0.22603

Table 5.32 Sum Square Errors for each cluster after linking of network D.

Cluster	SSE	Cluster	SSE
Q11	0.44401	Q31	0.25628
Q12	0.34688	Q32	0.15527
Q13	0.19470	Q33	0.14888
Q14	0.15266	Q34	0.11817
Q21	0.98650	Q41	1.47453
Q22	0.35590	Q42	0.36705
Q23	0.35842	Q43	0.34347
Q24	0.13925	Q44	0.03899

It can be noticed that the interpolation errors have increased substantially. This is because the objective function was easy to interpolate for the network used, which has resulted in very low interpolation errors to begin with. The error induction into network C caused by linking has reduced the networks recall accuracy and its interpolation capability on a similar scale. Figure 3.31 presents the SSE of the interpolating network in 2D and figure 5.33 in 3D. It can be observed that the quadrant with the highest error prior to linking Q42 is now on 8'th position if all quadrants are ranked by their errors.

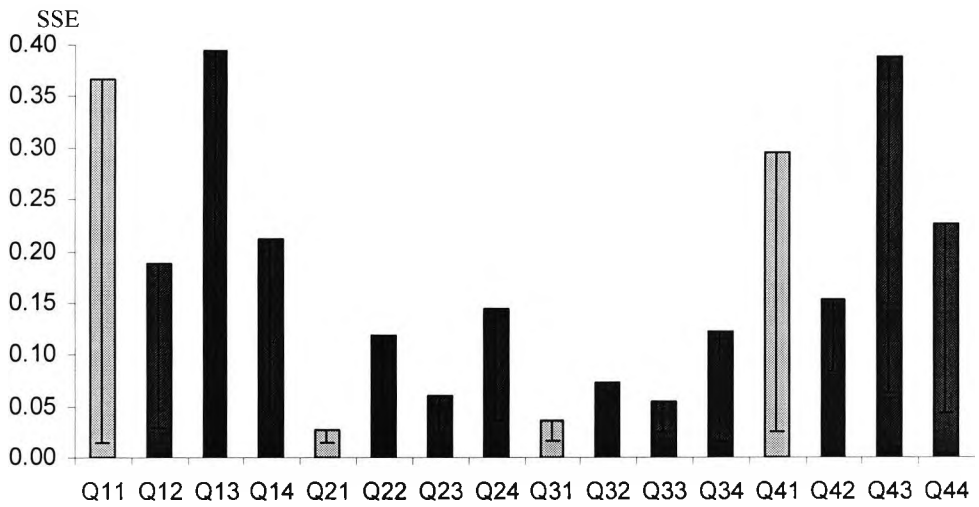


Figure 5.31 Error distribution for each cluster after linking 3 neurons of clusters from figure 5.25.

All errors for each cluster for network D have been exacted in the same manner as network C and are listed in table 5.32 and presented in figures 5.32 and 5.34. If the errors are compared with the errors after training, a general increase can be noted except for the extrapolated quadrants Q2 and Q4, where the error decreased.

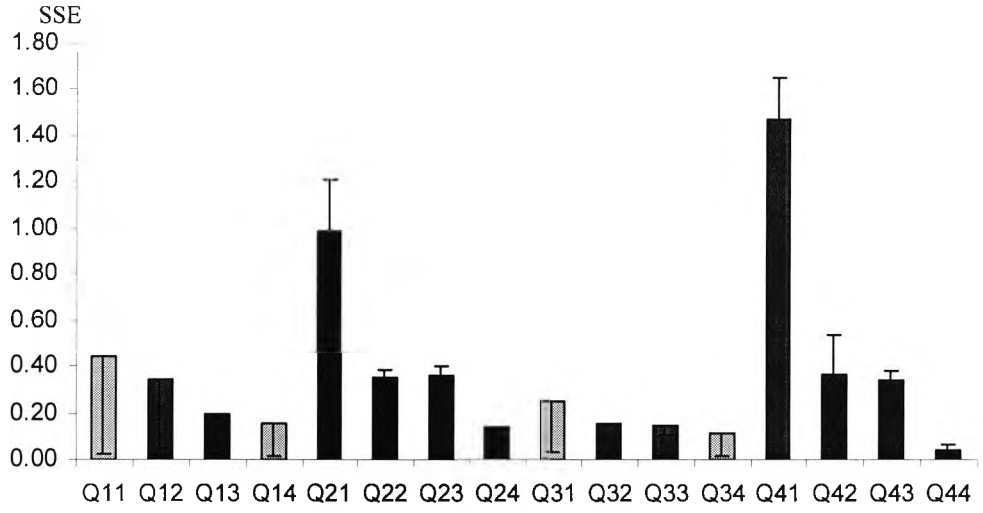


Figure 5.32 Error distribution for each cluster after linking 3 neurons of clusters from figure 5.16.

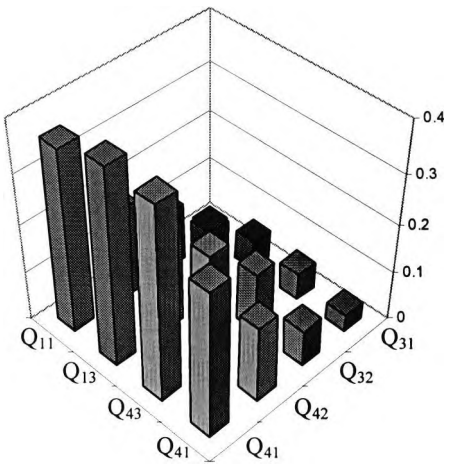


Figure 5.33 Cluster errors after linking for A.

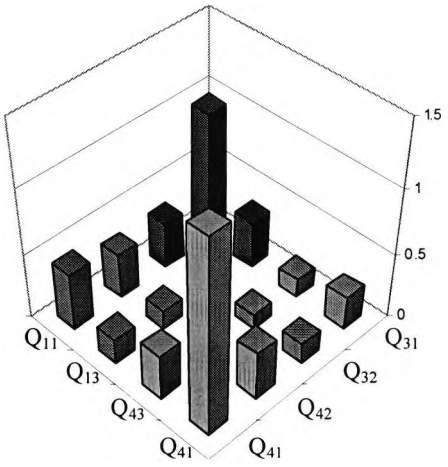


Figure 5.34 Cluster errors after linking for B.

5.7 Conclusions

Data commonly held in databases can easily be used for the creation of data models for the purpose of forecasting or analysis of input parameters for *what-if* cases [172]. With linking, the reusability of trained weight matrixes should be encouraged given that linking can be used for rapid integration and combination of knowledge. In order to use a trained weight matrix from a third party the input space margins for all input parameters should be known so that a generalisation request can be categorised into inter or extrapolation for a measure of confidence. In section 5.3.3 two networks with objective functions not based on mathematical functions have been trained and linked. After linking of three neurons, the linking analysis from section 5.4.2 has shown a reduction in the generalisation error but an increase in the recall accuracy error.

To analyse linking results from networks that have been trained for inter or extrapolation, the input space of an objective function based on a mathematical function has been split into partitions. Depending on the selection of partitions, networks have been trained for inter and extrapolation and subsequently linked to study their performance change on generalisation based on inter and extrapolation.

Extrapolation errors for clusters are increasing with distance to the nearest cluster used for training and with increasing target value. The larger the extrapolation target value, the larger the error because the extrapolation of smaller target values e.g. 0.6 resulted in smaller errors compared to extrapolation of higher values e.g. 0.8. After linking of three neurons, the analysis has shown good improvements for both networks with clusters located in the extrapolation areas of the input space.

In section 5.6 one interpolating network C and one extrapolating network D have been linked. The objective function was based on a mathematical function and was therefore relatively easy to interpolate by the network C, whilst the extrapolation error of network D was high. After linking of four neurons, the analysis has shown increased recall errors but improved extrapolation was achieved by network D. As a result, linking of interpolating and extrapolating networks improved extrapolation.

Chapter 6

Claims Reservation

6.1 Introduction

Claims reservation is a very important topic for non-life or property and casualty insurance companies such as liability and accident insurance. Financial reserves are needed for accounting, calculation of premium, reinsurance and asset liability management [125, 173, 174]. Reserves have to be present, due to the legal agreement in insurance contracts. Insurance companies must pay a claim if the claim has occurred during the insured period and has been delayed by claims processing or long court cases. There are two kinds of problematic claims for which reserves have to be built: IBNR (Incurred But Not Reported) reserves for claims, that are reported years after they occurred and IBNER (Incurred But Not Enough Reserved) for claims that have a very long regulation period like liability [175].

With IBNR, the total amount of claim size is not known at the end of the insurance period. This is especially true for persons' injuries (liability). Therefore a variety of mathematical methods for estimation of total loss amounts have been developed, one of the well known is the Chain Ladder Method (CLM).

A.M. Best International has supplied all of the data used in this chapter and granted permission for use of Insight Non-Life. Insight Non-Life contains the data of all major

non-life UK insurance companies that has been extensively validated and tested for correctness, making it the ideal data source for this chapter.

6.2 Claims Reserving

Claims Reserving is a vital topic in general insurance and serves the purpose of estimating the cost of claims to be paid out to the insured party by the insurer. Estimation of future events and their cost has mainly been part of advanced statistics and data modelling [176]. Improving claims estimation reliability and accuracy will increase the profitability of insurance companies and ensure their solvency in case of paying for claims as part of the contract. Most actuaries will use different methods to estimate the required claims reservation for specific groups of risk. Stochastic forecasting methods such as the Chain Ladder Technique with and without past claims numbers, Exponential Run-off and Curve Fitting have been well established and are used in most cases [125, 177, 178]. The purpose of this chapter is to apply neural network technology to insurance claims reserves estimation.

The insurance industry has the responsibility to meet the future claims of their policyholders with the result that the government forces companies to disclose information regarding their business activities under the Companies Act. For this purpose, general insurance companies secure large amounts for the reserve of outstanding claims. As for all insurance companies, the cost of their business lies in the future and the unknown extent of these figures is a major uncertainty. Producing the best estimate can be the difference between profitability and insolvency.

The problem of taxation of the whole of the claims reserve in the insurance business is that they are exempt from tax. Reserves to allow for possible adverse circumstances may be seen as a device adopted for the postponement of taxation properly due and seem negative from the government point of view. However, over reserving (or overprovision) is reducing the amount immediately available for distribution to

shareholders and reduces it's profitability and its future prospects in the insurance sector.

Insurance regulation has been well established over the past decades, preventing insurance companies from insolvency and subsequent collapse [179]. The Financial Services Authority (FSA) requires claims reserves to be broken down both by class of business and by year of origin. It's major interests are to protect the policyholder and therefore welcomes generous reserves, while the Inland Revenue is demanding a paring down of those same reserves so as to maximise taxable income. The existence of two contradictory requirements on the part of the Government apparently poses a dilemma when it comes to reserving. There is no absolute correct value for a claims reserve, since it depends on the purpose for which the reserve is required [125].

Statistical analysis of claims reserves is generally based on past and present experience. There is a minimum of information required for that process, in that statistical estimation of the reserves for very small classes becomes unreliable. For such classes, a claims assessor prepares a case-by-case approach for claims estimation. Many claims assessors have a life-long day-to-day claims experience, which uniquely equips them for the estimation task.

6.2.1 Claims Reserving for Different Types of Business

Insurance companies accept a wide variation of contracts and covers. Dependent on the type and nature of the risk, different reserving strategies need to be applied. Therefore, a general classification of the business categories into Types of Business is required. Types of Business can be distinguished between physical damage and liability; direct business and reinsurance; private and commercial business. Because a type of business can be composed of a mix of different risks, further subdivision into risk groups is required.

6.2.2 Types of Business

Insurance companies commonly classify insurance risks, other than reinsurance, by the following split in types of business:

Table 6.1 Insurance risks split by type of business.

No	Description	Characteristics
1	Accident & Health	Large number of policies, similar risks give homogeneity throughout this class.
2	Motor Vehicle	Large number of policies, risks must be split into risk groups for homogeneity.
3	Aircraft	Small number of contracts requires case-by-case analysis.
4	Shipping	Small number of contracts requires case-by-case analysis.
5	Goods in Transit	Small number of contracts requires case-by-case analysis.
6	Property Damage	Large number of policies, split into risk groups, claims settle within 2 years.
7	General Liability	Claims settlement can take 25 years and more, requires split into risk groups.
8	Pecuniary Loss	Large number of policies, split into risk groups, dependent on economic factors.

6.2.3 Claims Estimation Methods

An insurer who accepts a premium in one year may still be paying claims in respect of that policy many years later. This is because bureaucracy, court cases, long-term liabilities and so forth can delay claims payments. Therefore, the insurer needs to estimate the liability in order to have a basis for future premiums and claims reservation calculations. There are generally two distinct procedures the first being case-by-case estimation and the second being statistical analysis [125, 180, 181].

The case-by-case approach produces individual estimates, which is done by a claims assessor with experience of similar claims. If a large number of policies are available, sub-dividing similar claims into homogeneous risk groups can aid the assessor. It is then required to add an allowance for direct claims expenses, inflation, social or legislative changes and decide when payment is likely to occur. It is worth noting that even companies using an aggregated case-by-case approach to report future claims must use statistical analysis for taxation purposes.

Statistical analysis is a generic term covering almost any method that does not rely on examination of the individual claim file. Those range from simple ratio methods through a range of methods based on claims payment triangles. There is no single method, which is universally applicable. Commonly, several statistical methods are applied to the same set of data and their results compared. One of the major factors for the successful appliance of mathematical methods for estimation purposes is the availability of quality insurance data. Sometimes only limited information is available in the early years of a new business. In such case comparative methods across companies and years is necessary. To assist comparison between companies and years, data models of available data can be produced which can help in the process of comparing similar pre-existing risk groups of other companies with a new risk group introduced by new business. This chapter will concentrate on the construction of such data models with neural networks and compare the result with conventional statistical methods.

6.3 Data Preparation

The first and most obvious consideration is to determine what history data might be available. In order to generate a reliable mathematical estimation, history data of good quality on the specific type of business or risk group must be available.

The second consideration is the purpose of the estimate. If the estimate is required for published accounts a more cautious basis is required than that used in assessing premium rates. This will affect the desired accuracy of the estimate since the evaluation of premium rates can be changed on a daily basis, whereby published accounts are generally printed once a year. Additional differentiations are type of estimation such as long term, short term, interpolation or extrapolation.

A further consideration is the dependence on external circumstances, for example, future inflation, interest rate, unemployment, weather conditions and political

changes. If any of such information is available in data format, they should be included for the data model design and in the estimation process.

Producing an estimate with the help of a data model should not be restricted to one narrow range. If the range of the estimation is increased, the sensitivity to changes within the dependencies can be analysed. Highly sensitive dependencies can be pointed out and re-valuation of those can improve the overall prediction.

6.3.1 Types of Data used for Claims Reservation

As mentioned before, the ability to make good projections of past experience lies with the quality of historical data. Even with the best quality data available, any future projection will be subject to error. But the error can be reduced and confidence in the acquired results can be increased if the correct data or the correct combination of data items has been included in the modelling process. Furthermore, not only the data itself can influence the outcome, the data pre-processing such as the data presentation, data normalisation and denormalisation, data validation, data consistency and the data classification have significant involvement on the outcome of the estimation.

Historical data such as the number of claims reported, number of claims settled and the amounts paid out by way of settlement are the first benchmark data items beneficial for data modelling. Besides those data reflecting claims, data such as premium written or earned and measures of risk exposure are frequently available and may be included in the data model.

Since the future estimations are linked to time and classes of business, the history data needs to be split by year of origin and type of business. An insurance policy is not definitely restricted in time, some claims may occur years later; therefore the information on years of development is required within the history data.

6.3.2 Statistical Credibility of the Sample

The underlying principle of insurance is statistical in nature. A sufficient number of similar but independent risks is required to improve prediction within manageable margins and easing prediction of amounts payable in the next financial year. Hence an adequate premium can be set with some confidence in advance of the risk period itself. This is a result of what is popularly known as "the law of large numbers", which appears in statistical theory as the necessary relationship between the variance of a sample and its size [182].

Characterisation of claims for prediction can be archived by creating groups of data of similar but independent risks. Firstly, the data requires to be split into their main types of business. They are reflected in the supervisory authority classification and are law within the FSA. But the heterogeneity of many of these main classes is such to make further subdivision essential. The further the subdivision the greater the homogeneity in each of the resulting data groups.

Individual data groups can lose their statistical credibility because of lack of sample size, which in turns can cause a high variance. Data classes should have a similar risk profile and claims run-off. Similarities in claims run off can be detected if the overall tail length and development figures are alike. Generally, physical damage claims are settled within a few years, whereby major liability claims will take much longer.

If only a small number of data samples are available, different groups can be taken together for combined prediction and trend analysis if their ratio and business volume stays stable in the future.

If a large number of policies exist with the individual amount at stake relatively small, the conditions for statistical treatment are good. But if only a small number of policies exist within a class with large amounts at stake, case estimates (case-by-case analysis) are required [125].

6.3.3 Representation of Claims Data

Representing data in an understandable format is the first step in data analysis. The appropriate representation of data for a specific analysis can reveal certain trends just by graphing it as a simple line. Therefore, an appropriate data format should be considered if a successful prediction system is to be created.

Assuming a data sample, in which the risk classification, sample size and homogeneity is already established. What are the constraints on the data to describe a particular claims figure?

To begin with, there will be the claims amount paid during the course of the accounting year just past. This claims figure significance for projection purposes can be increased if the accounting year, length of the business run-off, the relative age of the claims and the relationship to premium income is known.

6.3.4 The Claims Triangle

The most common method for claims data representation is the claims triangle [183]. In the claims triangle claims data is displayed in Year of Development and Year of Origin on their x and y-axis respectively. The Year of Origin is the year in which the policy covering risk was taken. The Year of Development is the age of the insurance policy in years after a policyholder has taken it out.

For example, an insurance company has started to sell new policies of a new type of risk beginning in 1991. Policies sold in 1991 have caused claims to be paid out in 1991,1992...1996, whereby 1991 is the 1 Year of development, 1992 is the 2 Year of Development and so forth. Policies sold in 1992 have caused claims in 1992, 1993...1996, whereby 1992 is the 1 Year of Development, 1993 is the 2 Year of Development and so forth. The date in which the policies have been sold is the Year of Origin and the difference between the date in which claims occurred and the date in which the policy was bought are the Years of development. It is important to mention that this chapter uses a development year counting system based on 1 not on 0.

If the claims data is displayed in a data grid format displaying the Year of Development on the x-axis and the Year of Origin on the y-axis, the following triangular shape emerges:

Year of Development

Year of Origin

	1	2	3	4	5	6
1991	1001	854	568	565	347	148
1992	1113	990	671	648	422	
1993	1265	1168	800	744		
1994	1490	1383	1007			
1995	1725	1536				
1996	1889					

Figure 6.1 Data triangle of incremental claims figures.

Once the data has been put into the triangular format, it is very expressive of means for analysing and prediction of claims. Down the development year columns, the figures are indicating on how much money has been paid out in the first, second,...,sixth year after a policy has been sold. Across the origin year rows, the figures are indicating on how much money has been paid out for a policy sold in 1991, 1992...1996. Lastly, the diagonals can be seen to relate to the position in succeeding calendar years, with the lowest diagonal representing the calendar year immediately past. The sum of the figures in this diagonal is indicating on how much money has been paid in 1996, 1995...1991 (e.g. 1996: 1889 + 1536 +... = 5746).

There is a further variation of the table that is useful. Rather than looking at the year-by-year addition to the claims for each year of origin, cumulative development can be shown. The cumulative figures are obtained simply by adding the figures along each row. The process yields the triangle shown in figure 6.2.

If the data is analysed across the origin year rows, the figures are indicating on how much money in total for a specific development year has been paid out for a policy sold in 1991, 1992...1996. Summation of the diagonals will indicate how much

money has been paid out in total for the current type of business for a specific year of origin.

Year of Development

Year of Origin

	1	2	3	4	5	6
1991	1001	1855	2423	2988	3335	3483
1992	1113	2103	2774	3422	3844	
1993	1265	2433	3233	3977		
1994	1490	2873	3880			
1995	1725	3261				
1996	1889					

Figure 6.2 Data triangle of cumulative claim figures.

The triangular representation of data items is not just restricted to claims payment. Other information relating to claims payments can be displayed in the same format. Data such as Premium Income and Claim Numbers can be shown in the same way and used for the prediction task, whereby amounts paid out on claims are by definition the central quantity for reserving purposes. It is important to be clear as to the definition of the financial information involved e.g. are expenses, partial payments or fees included. A good data model requires data, which has been derived under similar criteria [184].

6.3.5 Economic Factors

Inflation is one of the major influences in financial forecasting. Neural Network can be trained with data where inflation is incorporated by implicit means [125]. But it may not be right to assume that past rates of inflation will continue to apply. Therefore, it can be assumed that historical data from the late 1980's, which contains high inflation rates, is not ideal for making claims forecasting for the late 1990's where inflation rates have been low [186]. Because car prices, repair costs and number of accidents have been fairly stable in the past, the insurance risk of cars can be assumed to be predictable. Therefore the risk group Private Car Comprehensive (PCC) has been chosen for the numerical experiment.

6.3.6 Claims Data Normalisation

Generally, data normalisation will result in a data set in which the value range lies in-between 0...1 for non-symmetrical sigmoid activation function or -1...+1 for symmetrical sigmoid or hyperbolic activation functions. This simple method for data normalisation is sufficient on a static data set in which global minimum and maximum are known.

Time series forecasting with financial data is not static. Whenever dynamic data forecasting is required, the global minimum and maximum figures are generally unknown. With claims data, the minimum can be assumed to be zero whereby negative claims are possible but out of scope for this chapter [187]. This assumption is not generally applicable for financial data since e.g. turnover data might be positive or negative. The maximum figure for claims normalisation cannot be found within the existing history data set since the data forecasting process may result in a new maximum. Therefore, the maximum figure contained within the training data set needs to be increased by an estimated factor. Initially, this factor could be the forecasted inflation for this particular type of business, found via mathematical progression or published expert opinion.

Application of such a factor in the normalisation process will result in a maximum figure of less than one in the training and testing data, avoiding saturation of the sigmoid activation function of the output neuron. Once the network has been trained and data prediction started, monitoring of the network output for saturation becomes essential.

If the normalised network output becomes close to one, the estimated inflationary factor has most probably been too small and output neuron saturation has occurred. In such a case, the inflationary factor needs to be adjusted appropriately and training of the network needs to be repeated. Besides the fact that the factor can be too small, it may be the case that it has been chosen too large. This can be detected by the network output being below a certain value, e.g. 0.6. Finding the ideal factor to avoid over- or under saturation can become a recurring process and can easily be automated.

6.3.7 Feature Selection Process

Defining which data items are relevant for the model in question is called feature selection. Selecting the right features to describe all pertinent dependencies is the first step to build up a fuller and more reliable data model. By definition, Incurred But Not Reported (IBNR) claims reserving requires a data model for its estimation. Generally, not only one mathematical approach should be used to produce a prediction result. The more different the prediction methods are in their background, the better the analysis of their confidence limits. But different methods require different features, which in turn reduces the ability of correlating the results.

Total claims paid (or incurred) is the amount of money paid to settle claims (generally including partial payments and expenses). The claim frequency is a figure reflecting the number of claims occurring during the same period as payments are made. Division of the total claims paid figure by the claims frequency is resulting in the average cost per claim figure. If a company has a large proportion of direct-writing business (low reinsurance ceded) the figures can be taken as gross. If a company has a large reinsurance proportion (low direct business), figures should be taken as net (gross-ceded).

Claim payments are dependent on how many losses can be recovered from reinsurance. The ratio between net premiums earned and gross premium written is called the retention ratio and determines the proportion of the income, which has not been used to pay for reinsurance. If a company changes its risk policy, a distortion in the historical data can be caused if the retention levels are changed drastically. In such a case, adjustments for changes in the retention limits must be made.

An important choice in claims development methods is whether to use paid or incurred loss data. The difference between paid losses and incurred losses are that paid losses are the actual paid amount, whereby incurred losses are the sum of paid and reserved money. If a claim has been reported and the claims figure is known (or estimated) but is not settled, the claims figure can be reserved since a settlement is expected. Such reserves are included in the incurred loss figure. If, for example, a

high percentage of claims occur close to the financial year-end and the money required for settlement has been reserved, the figures for paid losses and incurred losses would differ. Such difference is reflected in the ratio between the claims settled and claims reported which is called the claims settlement rate. If the claims settlement rate is lacking of stability across the financial periods, distortions in the projections can occur. If the claims settlement rate is constant, projections of paid losses and incurred losses are most likely to correlate, increasing confidence in the results. If high fluctuations are identified, a worst-case solution is more desirable [125, 188].

6.4 Claims Reservation with Chain Ladder Method

One important part of the business of a general insurance company is to forecast outstanding claims and setting up suitable reserves to meet these claims. The profits of insurance companies depend not only on the actual claims paid but also depend on the forecasts of the claims that will have to be paid.

The reserves that will be set aside to cover future claims to ensure the financial stability of the company and the stability of its profit and loss account need to be estimated in a reliable manner. There are a number of methods, which have proved useful in practice, one of which is extensively used and is known as the Chain Ladder Method (CLM) [125, 187].

6.4.1 Chain Ladder Method

This section will present the insurance data in the form of a claims triangle. It should be emphasised that this is for notational convenience only: there are no problems in extending the methods to other shapes of data. To reiterate, the year in which the policy has been written is often referred to as the underwriting year, accident year or year of origin. In the years after the policy was written the company may receive

claims related to that policy, and these claims are often referred to run off year or development year.

The data in table 6.2 shows the incremental run-off triangle of AXA Insurance plc for total claims. Since the SFA 1996 regulations limit the number of reported years to 10, a truncation of the triangle after the 10'th development year can be noticed. This slight derivation of the perfect triangle to a rectangle from 1989 onwards will not adversely affect any calculations.

Table 6.2 Incremental run-off triangle for AXA with actual 1999 data.

Year of Origin \ Development Year	1	2	3	4	5	6	7	8	9	10
1981	22277	10285	6565	3382	2552	336	667	324	267	229
1982	27669	12257	7451	4555	1681	2318	918	866	675	751
1983	26038	10963	7965	2807	3363	1226	716	210	341	199
1984	24594	11093	5299	5789	2946	2265	1306	757	378	528
1985	24274	9462	9247	7209	4494	2605	1753	929	651	580
1986	24008	11695	8712	7787	5020	3503	1964	1688	1055	625
1987	23081	14739	9606	7512	6028	3898	3198	2593	1723	171
1988	19373	10551	8917	7632	4490	4815	3241	1947	1978	1909
1989	35790	16960	15252	11216	9667	4935	2779	2410	1391	865
1990	63257	28691	23044	22279	14653	10191	6139	3788	3630	2872
1991	50348	26516	27625	18287	12760	6751	3648	3472	2347	
1992	42350	25735	18798	14394	9920	5642	6071	3258		
1993	33864	23350	15288	10877	5273	3146	1932			
1994	42833	26250	16261	9447	6351	3100				
1995	40277	25012	20030	19103	12761					
1996	61229	43086	48576	34089						
1997	89466	50737	41399							
1998	88171	79146								
1999	92442									

Run-off triangles can be presented in the form of incremental and cumulative claims forms for each development year. The data in figure 6.2 is shown as incremental figures. The main difference is that incremental figures decrease for higher run-off years whereby cumulative figures will increase.

The incremental claims relating to year of origin i and development year j will be denoted Z_{ij}, so that the observed data can be described in equation (6.1).

$$Z_{ij} : i = 1,..., t; j = 1,..., t - i + 1 \tag{6.1}$$

The neural network approach will use the incremental claim figures, but the CLM is applied to the cumulative claim figures, which are described in equation (6.2) and shown for AXA in table 6.3.

$$C_{ij} = \sum_{k=1}^{j} Z_{ik} \tag{6.2}$$

Table 6.3 Cumulative run-off triangle for AXA without 1999 data.

Year of Origin	Development Year									
	1	2	3	4	5	6	7	8	9	10
1981	22277	32562	39127	42509	45061	45397	46064	46388	46655	46884
1982	27669	39926	47377	51932	53613	55931	56849	57715	58390	59141
1983	26038	37001	44966	47773	51136	52362	53078	53288	53629	53828
1984	24594	35687	40986	46775	49721	51986	53292	54049	54427	54955
1985	24274	33736	42983	50192	54686	57291	59044	59973	60624	61204
1986	24008	35703	44415	52202	57222	60725	62689	64377	65432	66057
1987	23081	37820	47426	54938	60966	64864	68062	70655	72378	72549
1988	19373	29924	38841	46473	50963	55778	59019	60966	62944	64853
1989	35790	52750	68002	79218	88885	93820	96599	99009	100400	101265
1990	63257	91948	114992	137271	151924	162115	168254	172042	175672	
1991	50348	76864	104489	122776	135536	142287	145935	149407		
1992	42350	68085	86883	101277	111197	116839	122910			
1993	33864	57214	72502	83379	88652	91798				
1994	42833	69083	85344	94791	101142					
1995	40277	65289	85319	104422						
1996	61229	104315	152891							
1997	89466	140203								
1998	88171									

Past experience contained in the triangle is the history information that should be used for forecasting the data missing in the lower right hand triangle. Sometimes it is also useful to extend the forecasts beyond the latest development year (i.e. to the right of the claims run-off triangle) but the standard actuarial technique does not attempt to do this. This chapter will focus on the forecast of the 1999 development year only since

forecasting of subsequent years will follow the same procedure. The diagonal 1999 development year data to be forecast using CLM and neural networks can be seen shaded in table 6.2.

The CLM was developed from the theory that the amount of payments still to be made on a group of claims was related in a stable manner to the amount that has already been paid on those claims in earlier years. The basic CLM assumes that all external factors such as change in the rate of settlement of claims, alterations in the mix of business and inflation of claims costs can be ignored [125].

The CLM theory is based on development factors b_j. Development factors are ratios of cumulative payments in successive development years for each group of claims. The assumption is that the cumulative claims for each business year develops similarly by each development year, and estimates the development factors as ratios of sums of cumulative claims within the same development year. Thus the estimate of the development factor b for column j is shown in equation (6.3). Once the development factors b_j are known for progressive development years they can be used to fill in claims reservations for future years.

$$b_j = \frac{\sum_{i=1}^{t-j+1} C_{ij}}{\sum_{i=1}^{t-j+1} C_{ij-1}} \tag{6.3}$$

With the figures from table 6.3 and equation (6.3) the development factor b_2 for column 2 can be calculated as 1.549 as shown in equation 6.4.

$$b_2 = \frac{32562 + 39926 + ... + 140203}{22277 + 27669 + ... + 89466} = 1.549 \tag{6.4}$$

Summing each column in figure 6.3 and calculating the ratio by dividing the current column by the previous column will determine the b_j factors for the entire table. After the calculation of all development factors they can be used to estimate future

development years E_j by multiplication with the latest loss figure as shown in equation (6.5).

In equation (6.5) C_{ij} is multiplied with (b_j-1) and because 1 is subtracted from b_j, the estimated figure E_j is incremental. If C_{ij} had been multiplied with b_j, E_j would have been cumulative. Because neural networks will forecast incremental figures, equation (6.5) is subtracting 1 from b_j to permit direct comparison of results.

$$E_j = C_{ij} \cdot (b_j - 1) \tag{6.5}$$

If more years are to be forecast, incremental forecasts can to be summed up to create cumulative C_{ij} figures or the subtraction of 1 can be omitted. Because one year of forecasting is used in this chapter, equation (6.5) is used. Thus, the estimation for 1999 for all contracts of the origin year 1996 is 152891 ·(1.158-1) = 24160. Table 6.4 shows all development factor ratios and claims forecasts for 1999 obtained by CLM.

Table 6.4 Cumulative run-off triangle for AXA with predicted 1999 data using CLM.

		Development Year									
		1	2	3	4	5	6	7	8	9	10
Year of Origin	1981	22277	32562	39127	42509	45061	45397	46064	46388	46655	46884
	1982	27669	39926	47377	51932	53613	55931	56849	57715	58390	59141
	1983	26038	37001	44966	47773	51136	52362	53078	53288	53629	53828
	1984	24594	35687	40986	46775	49721	51986	53292	54049	54427	54955
	1985	24274	33736	42983	50192	54686	57291	59044	59973	60624	61204
	1986	24008	35703	44415	52202	57222	60725	62689	64377	65432	66057
	1987	23081	37820	47426	54938	60966	64864	68062	70655	72378	72549
	1988	19373	29924	38841	46473	50963	55778	59019	60966	62944	64853
	1989	35790	52750	68002	79218	88885	93820	96599	99009	100400	101265
	1990	63257	91948	114992	137271	151924	162115	168254	172042	175672	**1790**
	1991	50348	76864	104489	122776	135536	142287	145935	149407	**2446**	
	1992	42350	68085	86883	101277	111197	116839	122910	**2685**		
	1993	33864	57214	72502	83379	88652	91798	**3100**			
	1994	42833	69083	85344	94791	101142	**5224**				
	1995	40277	65289	85319	104422	**9208**					
	1996	61229	104315	152891	**24160**						
	1997	89466	140203	**40165**							
	1998	88171	**48424**								
	b_j	**N.A.**	**1.549**	**1.286**	**1.158**	**1.088**	**1.052**	**1.034**	**1.022**	**1.016**	**1.010**

In table 6.4, the brackets and arrows shown graphically illustrate two ratio calculations referring to equation 6.3. Furthermore, the forecasting results of the CLM for 1999 are shown diagonally in bold and the development factors b_j are in bold at the bottom. One major limitation of the CLM is that no forecast for 1999 for the first year of development can be made, since no development factor b_j is available.
The CLM forecasting results are again listed in table 6.5 in a more convenient way. They will be used as benchmarks that shall be used to compare the results obtained with using neural networks. Because financial forecasting is an extrapolation procedure, linking will be used in an attempt to improve the forecast.

Table 6.5 CLM result analysis for 1999.

Year of Origin	Claims Occurred	CLM Prediction	Relative Error
1990	2872	1790	-60.47%
1991	2347	2446	4.04%
1992	3258	2685	-21.32%
1993	1932	3100	37.68%
1994	3100	5224	40.66%
1995	12761	9208	-38.58%
1996	34089	24160	-41.10%
1997	41399	40165	-3.07%
1998	79146	48424	-63.44%
1999	92442	NA	NA

Table 6.5 contains the forecasting results from table 6.4 as well as the actual loss figure from table 6.2 for reasons of comparison. The forecasting results from CLM seem to be relatively reliable since errors are not larger than 65%. This is caused by the fact that the chosen company AXA has a very large number of contracts and behaves relatively consistent e.g. no major mergers. With this, the law of large numbers applies where trends in data sets containing a large number of samples are more stable than data sets that contain a small number of samples [182]. CLM relies on a stable development and requires a fully developed early year.

6.4.2 Other Forecasting Methods

Claims reserving is not limited to the CLM, there are many other methods of calculating future claims. There are numerous methods available for claims reservation calculations such as Bornhuetter-Ferguson method, Reid's method, Bayesian forecasting methods, methods based on regression models and other less complex curve fitting methods [189]. Many of these methods can include inflation, IBNR, number of claims, claim size, cost per claim and many other factors relevant to the claims reservation process. A discussion of claims reservation methods other than the CLM is out of the scope of this thesis but City University and The Faculty and Institute of Actuaries have published a comprehensive guide on claims reserving called "Claims Reserving Manual" [125].

6.5 Claims Reservation with Neural Networks

Neural networks have been used successfully in several areas of the insurance industry's claims processing. Such areas include detection of fraudulent claims, improved claims processing, pricing and prediction of claims duration [190, 191] but nothing has been found on the application of neural networks for claims reservation.
This chapter contains two numerical examples on claims reservation. The first example on claims reservation will restrict data availability to one company only. Data availability is reduced in cases where certain types of business are written with contracts specific to one company or with low policy numbers. The second example on claims reservation will use data from two companies. Data availability is high in cases where many companies are writing business with similar contracts for a large number of contracts. Both numeric experiments will train two neural networks both networks will be linked and operated with a stimuli network in an attempt to improve forecasting. The forecasting results of the trained only, linked only and linked with stimuli networks will be compared.

6.5.1 Claims Reservation with Data from one Company

To compare neural network training and linking with CLM, data must be homogenous to a certain degree else CLM will perform badly. Therefore the risk group Private Car Comprehensive with incremental claims reported and outstanding excluding INBR from AXA Insurance Company plc has been chosen and its data is shown in tables 6.2, 6.3 and 6.4. The advantage of this risk group is that insurance contracts have not changed significantly over the years thus homogeneity is present.

6.5.1.1 Training Data Preparation

Because data from one company is to be analysed in isolation, two distinct domains are created from the run-off triangle from table 6.2 so that two neural networks can be trained. For this purpose, training patterns are created in two directions. The first set of training patterns are created to follow the data sequence along the x-axis, the second set will follow the data sequence along the y-axis. Whilst the data contained in the triangle will remain unchanged, the information held in each training pattern will describe a different domain.

Data records created along the x-axis will contain information about the development years. Data changes in each development year reflect claims management and the delay of claims e.g. court proceedings, claims handling and claims processing.

Data records created along the y-axis will contain information about the year of origin. Data changes in each year of origin reflect economic factors and the occurrence of claims e.g. inflation, traffic management, road conditions and cost of claims settlement.

Since financial data contains time dependent information, financial forecasting is categorised as time series forecasting [192, 193]. The neural network tries to learn what the data changes for each time unit are. Time units are represented by individual training patterns so that no explicit time information is required in the training data. All financial data used in both examples are typical time series that contain continuous data changes for a constant time interval e.g. one year.

Figure 6.3 shows how time series data is split into input and output windows. In the example below, four input neurons and one output are used, which required that the input window size must be four and the output window one. A data point that was used as the target output in a previous pattern becomes an input in the next pattern. Data points that were used as inputs are passed along to the next input changing the input patterns into a time series.

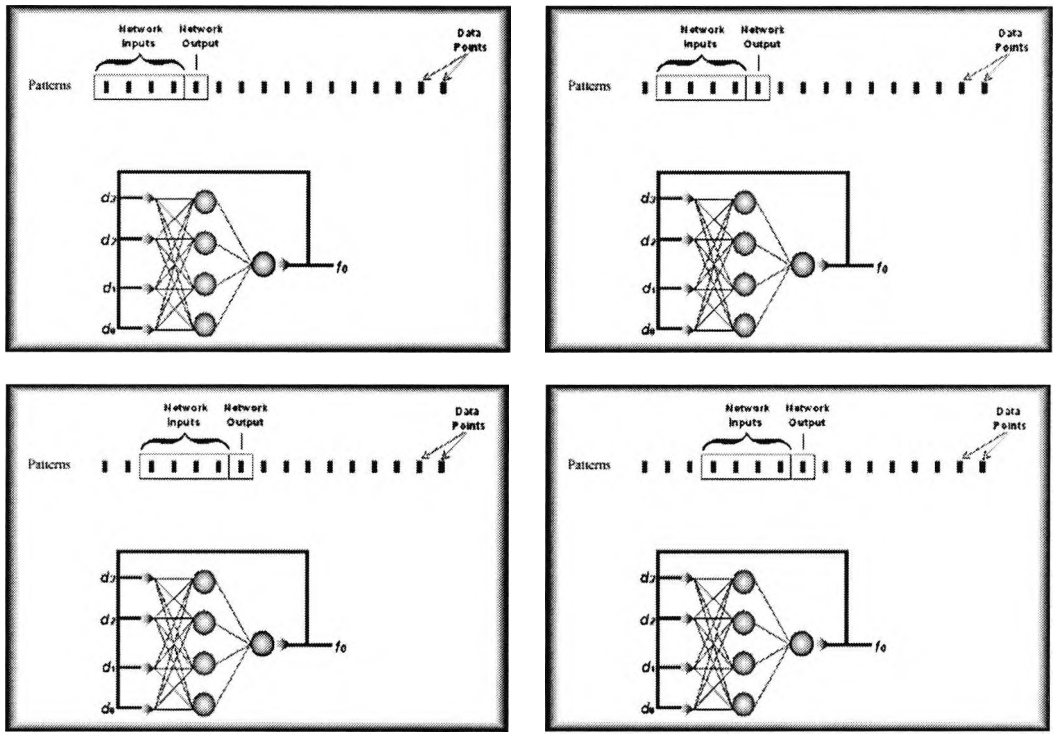


Figure 6.3 Training pattern generation for time series forecasting.

There are in principle two ways of training a neural network with time-delayed data. One way is to use a Time Delay Neural Network (TDNN). A TDNN has time delayed input neurons that shift data points from one input neuron to the next. This requires that the input neurons store data in memory in order to pass it on to the next. Another way is to use a normal MLP neural network where input and output vectors are shifted in the training data. With this, previous and next data points are stored within the training data file and no special TDNN software is required. Both methods are equivalent with the main difference being the location where input data is stored.

Several experiments with different sizes of input windows have been run and no relevant improvements on window sizes greater than five have been noticed. Therefore the window size of the input was set to five (five year history trail) with one output to predict (the next year's claim estimate).

Table 6.6 shows the cumulative claims run-off triangle from which training and testing data has been extracted. One sample pattern is shown outlined for each domain in x-and y direction referred to as the development and origin year direction. The first training pattern for development years starts at column 5 and ends at 9 with the output located at column 10. Subsequent patterns have been shifted to the left so that the next pattern input vector starts at column 4 and ends at 8 with the output located at 9.

Patterns for origin years will follow the same logic but in a vertical position. The first pattern input vector starts at row 1981 and ends at 1985 with the output at row 1986. Subsequent patterns will be shifted downwards so that the next pattern input vector will start at row 1982 and ends at 1986 with the output at row 1987.

Table 6.6 Creation of training data using time series window size of 5 inputs and one output.

Development Year

Year of Origin	1	2	3	4	5	6	7	8	9	10
1981	22277	32562	39127	42509	45061	45397	46064	46388	46655	46884
1982	27669	39926	47377	51932	53613	55931	56849	57715	58390	59141
1983	26038	37001	44966	47773	51136	52362	53078	53288	53629	53828
1984	24594	35687	40986	46775	49721	51986	53292	54049	54427	54955
1985	24274	33736	42983	50192	54686	57291	59044	59973	60624	61204
1986	24008	35703	44415	52202	57222	60725	62689	64377	65432	66057
1987	23081	37820	47426	54938	60966	64864	68062	70655	72378	72549
1988	19373	29924	38841	46473	50963	55778	59019	60966	62944	64853
1989	35790	52750	68002	79218	88885	93820	96599	99009	100400	101265
1990	63257	91948	114992	137271	151924	162115	168254	172042	175672	?
1991	50348	76864	104489	122776	135536	142287	145935	149407	?	
1992	42350	68085	86883	101277	111197	116839	122910	?		
1993	33864	57214	72502	83379	88652	91708	?			
1994	42833	69083	85344	94791	101142	?				
1995	40277	65289	85319	104422	?					
1996	61229	104315	152891	?						
1997	89466	140203	?							
1998	88171	?								
1999	?									

Continuing the creation of development and origin year patterns 55 and 85 patterns have been generated respectively. The development year pattern with the highest origin year will be of 1993 because five inputs and one output require six data points in horizontal direction therefore 1993 is the last data point available for the development year, as shown outlined in figure 6.5. It comes as a disadvantage that data from the last five years cannot be included in the training data.

Table 6.6 is not in the shape of an ordinary triangle because it benefits from additional history data from 1981 to 1988. If data from 1981 to 1988 would not have been available and table 6.6 would be a triangle, pattern creation for origin years would have stopped for origin year 1989 and development year 5. This would have caused that no training data for development years higher than 5 would be available for training. Because extensive history data has been provided, the training patterns for origin years include all 10 years. After creation of all patterns or vectors, they have been normalised by a vector based normalisation process that is dividing each vector component a_i by the vector length. This process is used to create normalised vectors with the length of 1 and is shown in its general form in equation 6.6.

$$a_{i_{norm}} = \frac{a_i}{\sqrt{\sum_i a_i^2}} \tag{6.6}$$

6.5.1.2 Training of Domain Networks

Two MLP neural networks have been trained with the normalised data from the previous section. The topology of both networks used has been chosen to be 5:10:1 with frozen outputs. All other important network parameters can be found in table 6.7. For ease of use, the neural network trained with development year data is referred to as x-direction and the one trained with origin year data is referred to as y-direction.

Training of both networks has been stopped when the generalisation has dropped or reached a plateau after a certain amount of training epochs. The data used for measuring the generalisation error are patterns containing 1999 data as target values.

Table 6.7 The parameters of the neural networks used in this section.

Description	X-Direction	Y-Direction
Input Neurons	5	5
Hidden Neurons	10	10
Output neurons	1	1
Activation Function	symmetric sigmoid	symmetric sigmoid
Initialisation	±0.7	±0.7
Learning Factor	0.01	0.01
Momentum	0.5	0.5
Number of training patterns	55	85
Number of testing patterns	5	10

Figure 6.4 shows the recall SSE_{trn} and generalisation error SSE_{gen} during training of the x-direction neural network. Training has been stopped after approximately 8000 iterations where the generalisation error reached the end of a plateau. The reason why training has not been stopped earlier when generalisation was low e.g. 0.2 is that the recall accuracy was too low e.g. 0.8. Whilst the generalisation error remained almost constant at 0.582 between 5500 and 8000 iterations, the recall error has fallen from 0.374 to 0.339.

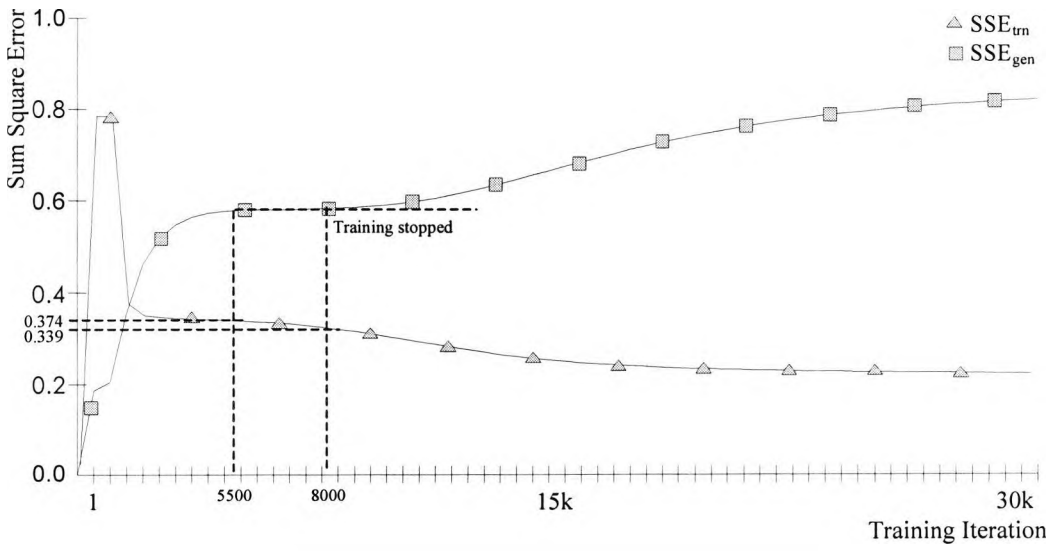


Figure 6.4 Recall and generalisation error of network trained with x-direction data.

Figure 6.5 shows how the recall SSE_{trn} and generalisation error SSE_{gen} change during training of the y-direction neural network. Training has been stopped after approximately 33,000 iterations where the generalisation error reached local minima. The reason why training has not been stopped earlier when generalisation was lower e.g. 0.6 is that the recall accuracy was too low e.g. 0.35. Whilst the generalisation error first reached an error of 0.719 at 28,000 and again at 33,000 iterations, the recall error has fallen from 0.202 to 0.191.

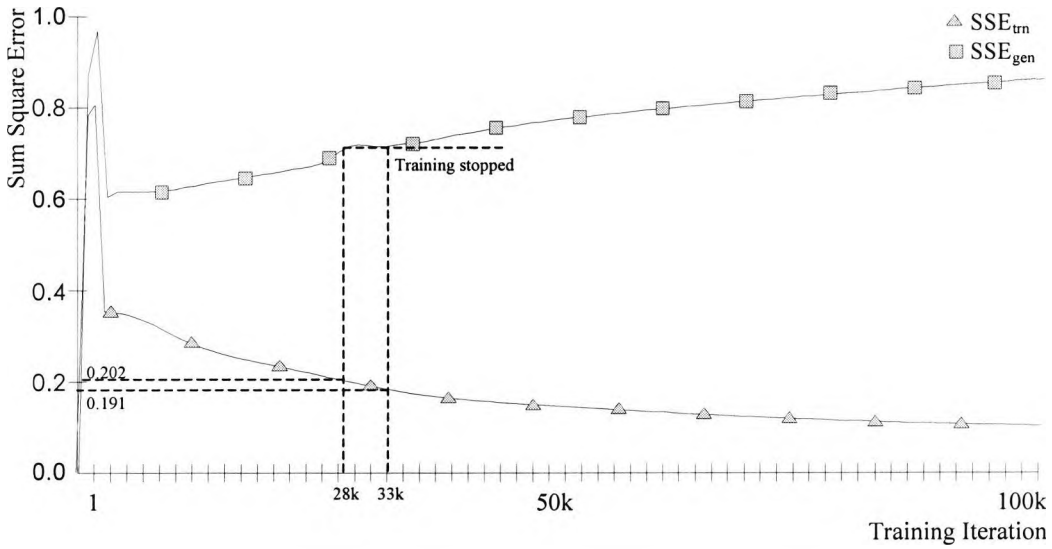


Figure 6.5 Recall and generalisation error of network trained with y-direction data.

After training, the generalisation results of both neural networks have been re-normalised and added as two differently shaded diagonal entries into table 6.8. The first diagonal entry is for the x-direction (x-dir) where forecasting starts at the 1994 origin year. This is because it is the first year where 5 consecutive origin years become available for creation of an input vector. The second diagonal entry is for the y-direction (y-dir) where forecasting is available for all years. Since history data for 1989 to 1981 has been available no origin year limitations exists.

Table 6.8 Extract of claims triangle with predictions from x and y directions.

Year of Origin										
1989	35790	52750	68002	79218	88885	93820	96599	99009	100400	101265
1990	63257	91948	114992	137271	151924	162115	168254	172042	175672	-2361
1991	50348	76864	104489	122776	135536	142287	145935	149407	-3522	2884
1992	42350	68085	86883	101277	111197	116839	122910	-2525	1115	
1993	33864	57214	72502	83379	88652	91798	-5379	786		
1994	42833	69083	85344	94791	1[illegible]	-9251	-386			
1995	40277	65289	85319	104422	NA	-1123				
1996	61229	104315	152891	NA	6317					
1997	89466	140203	NA	27736						
1998	88171	N[illegible]	21804							
x-dir	NA	58610								
y-dir	42011									

Table 6.9 shows a direct comparison between the occurred claims and forecasted claims for both x and y-directions after training.

It can be noted that the x-direction forecasts are particularly inaccurate. The reason might be that the development year information contained in the training data does not correlate well. A possible reason for this might be changes in jurisdiction, claims management, claims handling or claims processing occurred within the last 10 years.

Table 6.9 Forecasting results for x and y directions after training.

Year of Origin	Claims Occurred	x-Direction	Error	y-Direction	Error
1990	2872	-2361	-182.22%	2884	0.40%
1991	2347	-3522	-250.07%	1115	-52.48%
1992	3258	-2525	-177.51%	786	-75.87%
1993	1932	-5379	-378.46%	-386	-119.95%
1994	3100	-9251	-398.42%	-1123	-136.24%
1995	12761	NA	NA	6317	-50.50%
1996	34089	NA	NA	27736	-18.64%
1997	41399	NA	NA	21804	-47.33%
1998	79146	NA	NA	58610	-25.95%
1999	92442	NA	NA	42011	-54.55%

Unlike the poor performance for the x-direction, the y-direction performs reasonably well. It is noticeable that both forecasts perform worst for the year 1994 followed by

1993. A graphical illustration of the x-direction forecast is given in figure 6.6 and the y-direction forecast in figure 6.7. Only a slight parallelism in the trend for the years 1990 to 1992 can be noted in figure 6.6 whereby the prediction in figure 6.7 seems to follow the target values to some extent.

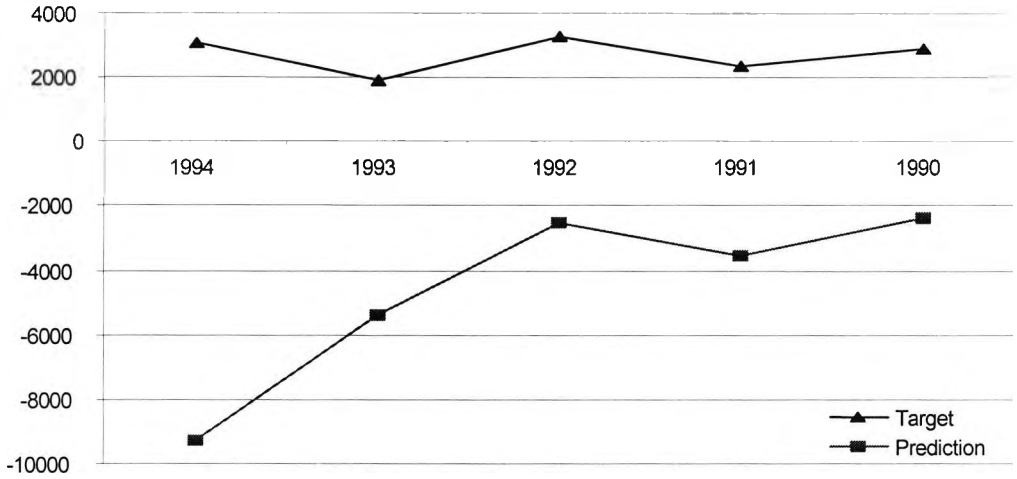


Figure 6.6 Prediction of claims reservation for x-direction.

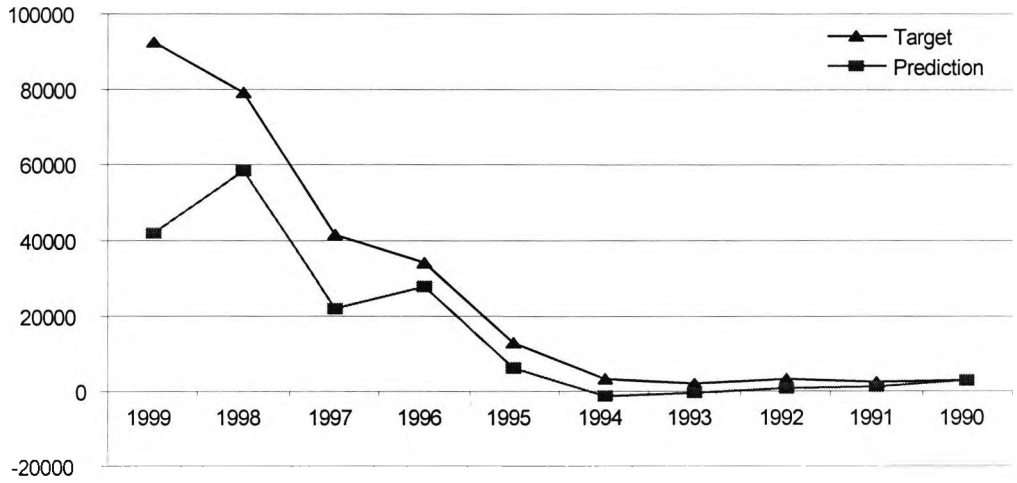


Figure 6.7 Prediction of claims reservation for y-direction.

The findings that the training for the y-direction performs better than the x-direction has been used in later sections where data from 2 companies will be used for training.

Table 6.10 summarises the recall and generalisation sum square errors of the forecast for both domains. To increase statistical reliability the experiment has been repeated 10 times up to this point and the averages, minimums and maximums for training and testing pattern are reported. Even if the average generalisation error SSE_{gen} for the x-direction 0.644 is lower than the y-direction 0.723, the RMSE errors are 0.644/5 = 0.129 and 0.723/10 = 0.0723. This shows that generalisation for the x-direction is less accurate than the generalisation for the y-direction, since the x-direction has only 5 testing patterns compared to the y-direction with 10.

Table 6.10 Performance benchmarks of both networks after training for 10 runs.

Description	x-Direction after training				y-Direction after training			
	Min	Max	Average	StDev	Min	Max	Average	StDev
SSE_{trn}	0.113	0.400	0.287	0.0753	0.191	0.328	0.265	0.0458
SSE_{gen}	0.573	0.839	0.644	0.0927	0.688	0.773	0.723	0.0304

Even if the x-direction error is high, it contains trend information. Linking of both networks might still be feasible to share some information between x or y-direction in an attempt to improve extrapolation for x and y-direction. For the purpose of linking, table 6.11 shows the hidden weight matrix of the network trained with x-direction data and table 6.12 shows the hidden weight matrix of the network trained with y-direction data. Please note that all particulars of the experiment refer to the first run.

Table 6.11 Hidden layer weight matrix of network trained for the x-direction.

Reference	x-Direction						
	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{1B}	Length
v_1	-0.7061	-0.5966	-0.5052	-0.2110	0.4972	0.0390	1.1844
v_2	-0.9418	1.3641	-0.1610	0.8043	0.9280	0.9029	2.2577
v_3	-1.1891	0.2811	0.0521	-0.0836	0.3082	1.0401	1.6369
v_4	-0.7141	-0.7614	-0.7024	-0.2328	0.7341	-0.0777	1.4772
v_5	-0.1264	-0.3282	0.4906	-0.4421	0.1188	0.3916	0.8528
v_6	-0.9531	-0.5827	-0.7651	-0.0577	-0.4088	-0.0391	1.4161
v_7	-0.9339	-0.6151	0.2446	-0.2631	-0.5830	0.0159	1.3114
v_8	-0.6809	-0.6078	-0.7497	-0.7840	0.3700	0.8829	1.7106
v_9	-0.6297	-0.8539	-0.0904	0.4080	0.0946	0.0046	1.1442
v_{10}	0.8189	-0.9967	-0.4257	-0.4598	-0.2967	1.0662	1.8115

Table 6.12 Hidden layer weight matrix of network trained for the y-direction.

Reference	y-Direction						
	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{1B}	Length
v_1	-0.4675	-0.5324	-0.1402	-0.3974	-0.6078	0.1928	1.0422
v_2	-0.9238	1.8511	1.6697	1.3047	-3.2385	1.1987	4.5492
v_3	5.1030	-2.1313	1.3624	0.9956	-0.3296	-0.3267	5.8005
v_4	-0.2072	-0.4222	-0.9357	-1.2531	-1.5552	2.1702	3.1298
v_5	-1.0807	-0.1954	-0.8365	-0.3476	-1.1061	1.4559	2.3172
v_6	-0.6659	-0.2585	-0.2307	-0.5447	-0.9239	0.8789	1.5767
v_7	-3.0340	2.4388	-2.2279	-0.6553	-0.3719	2.5248	5.2018
v_8	-2.4411	-0.9225	-3.3294	-3.0744	2.4209	3.2508	6.6163
v_9	-0.5268	-0.5480	-0.0362	-0.3643	-0.5239	-0.0542	0.9947
v_{10}	-0.0272	-2.0419	-0.9483	-1.3387	2.9135	1.4240	4.1686

Both networks have been trained with the algorithm developed in 4.6 to prevent neuron saturation. As a result no excessively large weights in any of the two weight matrixes can be found. If component values of weight vectors are spread over a wide range, e.g. ±20 instead of ±5, the probability of finding vectors pointing in similar directions is lower. Therefore equation 4.17 does not only prevent the creation of dominant neurons, it keeps neuron vectors closer together thus increasing the chances of finding neurons that can be linked.

6.5.1.3 Linking of Domain Networks

To combine the knowledge of both x and y-direction networks, linking as described in section 5.5.3 has been performed. Contrary to previous chapters, an acceptance angle of 20° instead of 10° has been used for linking since 5 input neurons have increased the dimensionality of the vectors from 3 dimensions in previous chapters to 6 dimensions (5 inputs + 1 bias). An increase in vector dimensions decreases the probability of vectors pointing in the same direction therefore an increase in the acceptance angle has been required to find sufficient numbers of neurons for linking. The linking process follows exactly the same process as demonstrated in section 5.5.3 and will therefore be described only in brief.

Table 6.13 shows the vectors, which have an angle difference below the acceptance angle of 20°. Both vectors listed in table 6.13 qualify for linking and table 6.14 shows the resulting vectors and their associated length correction factor F_2.

Table 6.13 Angles between weight vectors in ascending order.

Vector pair		Angle between vectors
x-Direction	y-Direction	
v_8	v_8	15.71°
v_7	v_9	19.96°

Table 6.14 Results of the combination of vectors with angles below 20°as listed in table 6.13.

Original vector references		Resulting vector v_{r1}						
x-Direction	y-Direction	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{1B}	Factor F_2
v_8	v_8	-0.5976	-0.2317	-0.8294	-0.7579	0.6242	0.7974	4.0270
v_7	v_9	-0.8619	-0.6390	0.2384	-0.3525	-0.6078	0.0498	0.7275

It can be observed that only two neurons comply with the acceptance angle constraint of 20°. Because of an increased number of input neurons more vector components are present in each weight vector, which has reduced the probability that they are pointing in similar directions in hyperspace.

6.5.1.4 Linking Analysis

Table 6.15 shows that the relative errors between components of the trained weights and the weights after linking. It can be observed that the relative errors are generally lowest for large weights and largest for small weights and that the error increases as the angle between the linked vectors grows. The table shows that the largest component error 61.89% of vectors v_8:v_8 (x-direction:y-direction) is much smaller than the largest component error -579.30% of v_7:v_9. The reason for this might be that the vector angle of 15.71° between v_8:v_8 is lower than the angle of 19.96° between v_7:v_9 and the lower the angle is, the better the match between the vector components.

Even with such a high error between vectors v_7:v_9, an attempt of linking both vectors seems feasible since more than half or all the other errors are less than 30%.

Table 6.15 Vector component change impact analysis.

Reconstructed Vectors	Vector Components					
	w'_{11}	w'_{12}	w'_{13}	w'_{14}	w'_{15}	w'_{1B}
x: v'_8	-0.5976	-0.2317	-0.8294	-0.7579	0.6242	0.7974
y: v'_8	-2.4065	-0.9329	-3.3400	-3.0519	2.5139	3.2112
x: v'_7	-0.8619	-0.6390	0.2384	-0.3525	-0.6078	0.0498
y: v'_9	-0.6270	-0.4648	0.1735	-0.2565	-0.4422	0.0363
Reconstructed Vectors	**Relative Errors**					
	$f(w_{11}, w'_{11})$	$f(w_{12}, w'_{12})$	$f(w_{13}, w'_{13})$	$f(w_{14}, w'_{14})$	$f(w_{15}, w'_{15})$	$f(w_{1B}, w'_{1B})$
x: v'_8	-12.23%	-61.89%	10.63%	-3.33%	68.74%	-9.68%
y: v'_8	-1.42%	1.13%	0.32%	-0.73%	3.84%	-1.22%
x: v'_7	-7.71%	3.88%	-2.52%	33.96%	4.26%	212.56%
y: v'_9	19.03%	-15.18%	-579.30%	-29.61%	-15.60%	-166.89%

Similarly to the analysis in previous chapters, table 6.16 lists the relative errors of the vector length. It can be observed that the higher error rests with the vectors v_7:v_9 since the angle between both vectors is larger than the angle between v_8:v_8.

Table 6.16 Vector length change impact analysis.

Vector	Original length	Reconstructed length		Relative Error
x: v'_8	1.7106	1.6427	($\|v_{rl}\|$)	-3.96%
y: v'_8	6.6163	6.6154	($\|v_{rl}\|*F_2$)	-0.01%
x: v'_7	1.3114	1.3054	($\|v_{rl}\|$)	-0.45%
y: v'_9	0.9947	0.9497	($\|v_{rl}\|*F_2$)	-4.52%

Tables 6.17 and 6.18 show the reconstructed weight matrixes for the x-direction and y-direction networks respectively. Rows that are framed on grey backgrounds are vectors that have been replaced with reconstructed vectors.

With the resulting weight matrixes, evaluation of the forecasting can be repeated for x and y-direction without stimuli network. Just as in previous chapters, input vectors from the x-direction are tagged as belonging 100% to A and input vectors from the y-

direction are tagged as belonging 100% to B. Once a stimuli network has been trained, domain memberships can be evaluated on a vector-by-vector basis.

Table 6.17 Hidden layer weight matrix of network trained for the x-direction after reconstruction.

Reference	x-Direction						
	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{1B}	Length
v_1	-0.70607	-0.59656	-0.50520	-0.21096	0.49720	0.03905	1.1844
v_2	-0.94179	1.36412	-0.16097	0.80429	0.92799	0.90291	2.2577
v_3	-1.18914	0.28106	0.05213	-0.08358	0.30823	1.04010	1.6369
v_4	-0.71408	-0.76140	-0.70238	-0.23280	0.73415	-0.07767	1.4772
v_5	-0.12638	-0.32824	0.49059	-0.44205	0.11879	0.39160	0.8528
v_6	-0.95311	-0.58273	-0.76507	-0.05775	-0.40877	-0.03911	1.4161
v_7	-0.86189	-0.63895	0.23842	-0.35251	-0.60780	0.04983	1.3054
v_8	-0.59759	-0.23166	-0.82940	-0.75786	0.62425	0.79740	1.6427
v_9	-0.62968	-0.85387	-0.09035	0.40803	0.09460	0.00457	1.1442
v_{10}	0.81887	-0.99672	-0.42566	-0.45983	-0.29666	1.06618	1.8115

Table 6.18 Hidden layer weight matrix of network trained for the y-direction after reconstruction.

Reference	y-Direction						
	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{1B}	Length
v_1	-0.46752	-0.53238	-0.14021	-0.39735	-0.60776	0.19281	1.0422
v_2	-0.92383	1.85108	1.66970	1.30472	-3.23850	1.19869	4.5492
v_3	5.10304	-2.13132	1.36235	0.99565	-0.32957	-0.32667	5.8005
v_4	-0.20722	-0.42217	-0.93568	-1.25313	-1.55524	2.17018	3.1298
v_5	-1.08066	-0.19543	-0.83652	-0.34756	-1.10606	1.45589	2.3172
v_6	-0.66587	-0.25852	-0.23066	-0.54465	-0.92392	0.87887	1.5767
v_7	-3.03399	2.43877	-2.22793	-0.65527	-0.37194	2.52485	5.2018
v_8	-2.40653	-0.93289	-3.34002	-3.05195	2.51386	3.21116	6.6154
v_9	-0.62703	-0.46484	0.17345	-0.25645	-0.44217	0.03625	0.9497
v_{10}	-0.02724	-2.04189	-0.94826	-1.33872	2.91345	1.42395	4.1686

After matrix reconstruction and input vector classification for x and y-direction, the new forecasting results are shown in table 6.19. This table can be compared with table 6.9 for an error analysis after linking. It shows that all errors for x-direction and most errors for y-direction have fallen and that only a few errors have increased. Figures 6.8 and 6.9 compare the predicted claims with the actual claims as they occurred.

Table 6.19 Forecasting results for x and y directions after linking.

Year of Origin	Claims Occurred	x-Direction	Error	y-Direction	Error
1990	2872	-1305	-145.42%	3693	28.59%
1991	2347	-2218	-194.49%	2103	-10.40%
1992	3258	-1057	-132.46%	3241	-0.51%
1993	1932	-3913	-302.53%	3227	67.01%
1994	3100	-6577	-312.16%	-2741	-188.40%
1995	12761	NA	NA	11497	-9.90%
1996	34089	NA	NA	37992	11.45%
1997	41399	NA	NA	32568	-21.33%
1998	79146	NA	NA	82406	4.12%
1999	92442	NA	NA	98242	6.27%

On comparison of figure 6.8 with figure 6.6, a shift of the forecasted claims towards the actual claims can be noticed. This shift has reduced the generalisation error from 0.339 to 0.209. Looking back to table 6.15 where vector component errors are listed, a high error in the bias values of v_7:v_9 of 212.56% can be found. Therefore it can be assumed that the high error of the x-direction vector bias value caused by linking has shifted the predicted claims towards the target claims.

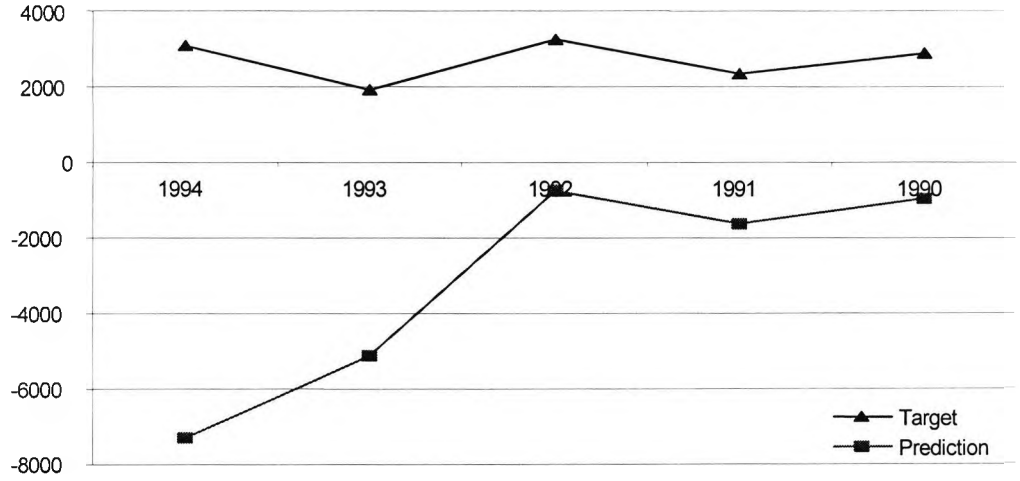


Figure 6.8 Prediction of claims reservation for x-direction after linking.

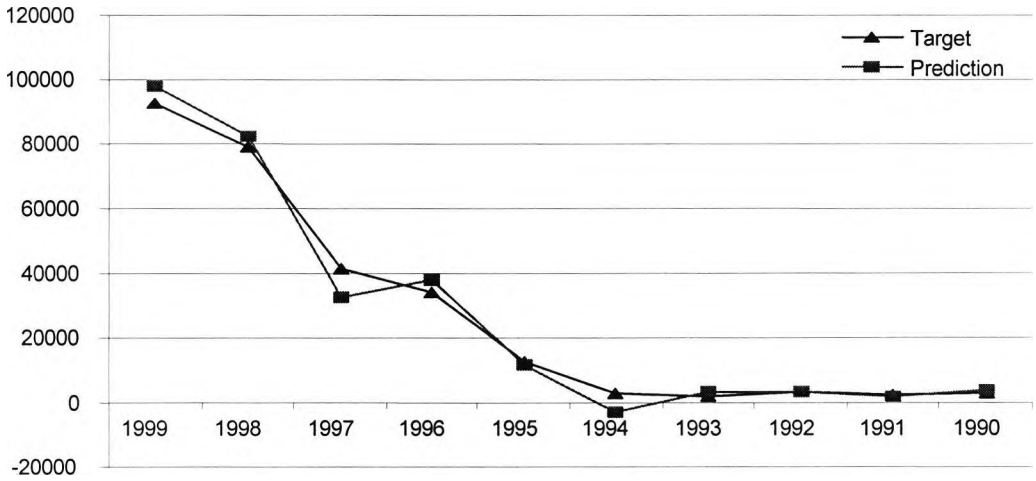


Figure 6.9 Prediction of claims reservation for y-direction after linking.

Comparing figures 6.9 and 6.7 shows that the errors for 1999-1995 have been reduced. This seems rather peculiar since no information for these years was present in the weight matrix of the x-direction network. Looking back to table 6.12 and table 6.15, the higher errors affecting the y-direction lie with v_9 components w_{13} and w_{1B}. For the reason that w_{1B} has only changed from –0.0542 to 0.0363 (-166%), its effect can only assumed to be lower than the effect of w_{13} with a change from –0.0362 to 0.1735 (-579%).

Table 6.20 summarises the forecasting results for both domains after training and linking for a total of 10 runs. It shows that the generalisation error has fallen on average by 14.24% and 13.79% for x and y-direction respectively, whereby the recall error has increased by 129.49% for the x-direction and 69.97% for the y-direction. The high percent increase of the recall errors is somewhat alarming but since reduction of the generalisation error is the objective, the increase in recall error appears to be acceptable. The largest drop in generalisation error encountered during 10 runs was 54% for x-direction and 47% for y-direction. But increases of 25% and 17% were also encountered.

Table 6.20 Performance comparison of trained and linked networks for 10 runs.

Description	x-Direction after training				y-Direction after training			
	Min	**Max**	**Average**	**StDev**	**Min**	**Max**	**Average**	**StDev**
SSE_{trn}	0.113	0.400	0.287	0.0753	0.191	0.328	0.265	0.0458
SSE_{gen}	0.573	0.839	0.644	0.0927	0.688	0.773	0.723	0.0304
	x-Direction after linking				**y-Direction after linking**			
SSE_{trn}	0.294	1.430	0.660	0.3756	0.335	0.659	0.451	0.1414
SSE_{gen}	0.362	0.755	0.553	0.1555	0.363	0.828	0.623	0.1267

6.5.1.5 Training of Stimuli Network

A stimuli network is used as a classification network within in a linked network environment. Its purpose is to classify incoming input vectors to domain memberships that can be used to control vector lengths for neurons that have been linked or not. Therefore, the stimuli network is actually controlling each neurons contribution to the overall network output.

Since the stimuli network must be aware of the entire input space, all training patterns that have been used to train x and y-direction neural networks must be utilised. As the stimuli networks awareness is only required within the input space, all target values of the training data can be omitted. Instead of using the target values for x and y-direction, they can be replaced with classification information.

Table 6.21 shows an extract of the training data used for training of x and y-direction, which will be referred to as domain A and B. It should be noted that two domains A&B require the stimuli network to have two outputs (out A and out B), one for each domain and 5 inputs since it uses the same input vector as the neural networks.

In table 6.21 the output values have been replaced with clustering information in such a way that input vectors used for training of domain A should generate 1-0 outputs and input vectors used for training of domain B should generate 0-1 outputs. This type of 1-0 and 0-1 clustering is limited and only indicates the physical pattern file location of each input vector. Input vectors of domain A are stored in a different pattern file than domain B therefore generate a different classification and do not represent the real domain membership that is required from the stimuli network.

Table 6.21 Stimuli network training data with clustered 1 or 0 outputs.

x-Direction							y-Direction						
in 1	**in 2**	**in 3**	**in 4**	**in 5**	**out A**	**out B**	**in 1**	**in 2**	**in 3**	**in 4**	**in 5**	**out A**	**out B**
0.59	0.46	0.29	0.17	0.11	1	0	0.26	0.23	0.26	0.25	0.43	0	1
0.62	0.34	0.29	0.25	0.14	1	0	0.20	0.15	0.32	0.49	0.43	0	1
0.67	0.39	0.22	0.11	0.16	1	0	0.32	0.21	0.37	0.35	0.39	0	1

The 1-0 clustering training for the stimuli network did not produce good training results because the classification of 1-0 for A and 0-1 for B did not represent the true classification for improving the network output. They simply signified the physical separation into 2 files but not their true classification. Instead of using 1-0 and 0-1 classifications, a simple brute force linear search for the optimal classification target has been carried out. Figure 6.10 shows how the stimuli network can be used to find close matches for domain memberships of A and B by incrementing the memberships in 0.01 steps and collecting the output of the network for each step. A linear search of this kind requires 1/0.01 * 1/0.01 = 10,000 iterations. Since the target values of all training vectors are known, the output value closest to the target value will determine the domain memberships for A and B.

Searching through the domain space by using a brute force linear algorithm will result in meaningful target values for the training of the stimuli network. Table 6.22 shows input vectors in combination with improved A&B domain memberships where the overall network output is closest to the initial training target value.

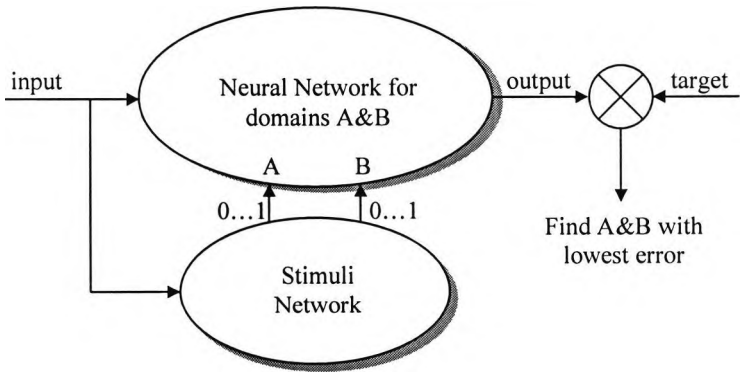


Figure 6.10 Generation of classification data by linear search.

Table 6.22 shows an extract of six training data patterns for x and y-direction. The stimuli network has been trained with 55 + 85 = 140 patterns each one with one output for domain A and B.

The linear search through 140 patterns with a granularity of 0.01 requires 140 * 10,000 = 1,400,000 iterations. With today's technology (2.6 GHz P4) this process took less than 25 seconds. Reducing the granularity from 0.01 to 0.1 can speed up this process, although this will cause less accurate domain memberships for A and B.

Table 6.22 Stimuli network training data with generated memberships.

x-Direction							y-Direction						
in 1	**in 2**	**in 3**	**in 4**	**in 5**	**out A**	**out B**	**in 1**	**in 2**	**in 3**	**in 4**	**in 5**	**out A**	**out B**
0.59	0.46	0.29	0.17	0.11	0.22	0.05	0.26	0.23	0.26	0.25	0.43	0.66	0.01
0.62	0.34	0.29	0.25	0.14	0.59	0.09	0.20	0.15	0.32	0.49	0.43	0.47	0.07
0.67	0.39	0.22	0.11	0.16	0.18	0.3	0.32	0.21	0.37	0.35	0.39	0.53	0.13

With the availability of the training data, the stimuli network will have to match the data insofar that it must have 5 inputs and 2 outputs. To find the best number of hidden neurons experiments with 5, 10 and 20 hidden neurons have been carried out and the best training performance has been found with 10 hidden neurons. Table 6.23 presents a list of all significant network parameters used for the stimuli network.

Table 6.23 The parameters of the stimuli network.

Description	**Setting**
Input Neurons	5
Hidden Neurons	10
Output neurons	2
Activation Function	symmetric sigmoid
Initialisation	±0.7
Learning Factor	0.01
Momentum	0.3
Number of training patterns	110
Number of testing patterns	30

Figure 6.11 shows how the recall SSE_{trn} and generalisation error SSE_{gen} change during training. Training has been stopped after approximately 295k iterations where the generalisation error reached a plateau. The reason why training has not been stopped at earlier is that the recall accuracy has fallen from 1.93 to 1.72 whilst the generalisation error remained nearly unchanged at about 0.83.

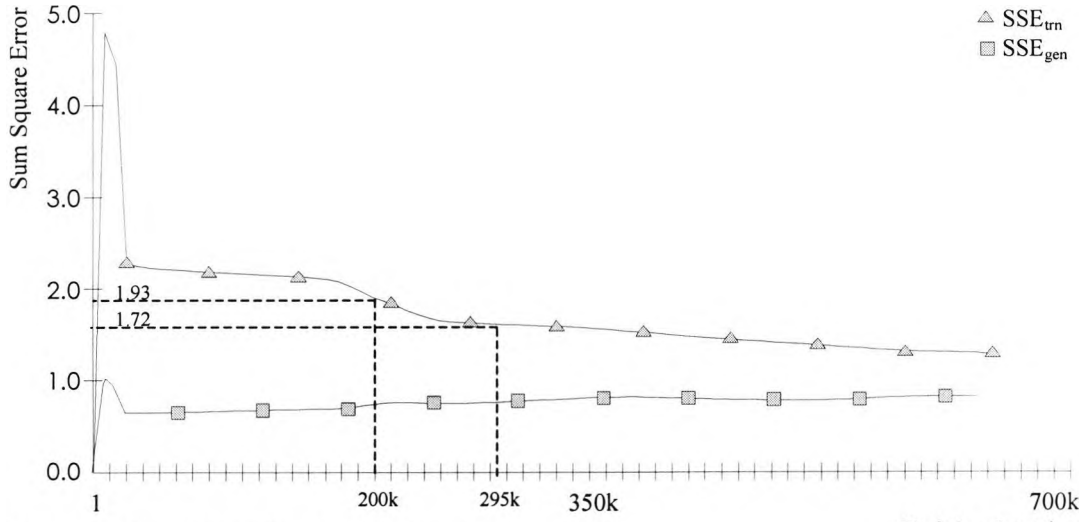


Figure 6.11 Training results of stimuli network.

Table 6.24 summarises the recall and generalisation sum square errors of both domain memberships for A and B. Even if the recall error SSE_{trn} for both domains is high, the RMSE is acceptable because of the high number of 110 training patterns.

Table 6.24 Performance benchmarks of stimuli network after training.

Description	Stimuli Network
SSE_{trn}	1.721
SSE_{gen}	0.830
RMSE for SSE_{trn}	0.164 (110 records)
RMSE for SSE_{gen}	0.151 (30 records)
Iterations	295,000

The linear search has yielded a much more reliable stimuli classifier system compared to a stimuli network that has been trained with data where the domain memberships were simply set to 1-0 and 0-1 for A and B correspondingly. With this, the trained stimuli network can now be used for classification of input vectors for linked domains.

6.5.1.6 Linking Results

To evaluate the impact of linking with utilisation of a stimuli network, the forecasting of claims reservation for 1999 has been repeated and is shown in table 6.25. If this table is compared with table 6.18 a reduction for all x-direction and half of the y-direction forecasts can be noticed.

Table 6.25 Forecasting results for x and y directions after linking.

Year of Origin	Claims Occurred	x-Direction	Error	y-Direction	Error
1990	2872	480	-83.30%	2876	0.13%
1991	2347	1444	-38.46%	2961	26.17%
1992	3258	616	-81.09%	2092	-35.78%
1993	1932	977	-49.45%	3470	79.61%
1994	3100	223	-92.79%	2977	-3.97%
1995	12761	NA	NA	10290	-19.36%
1996	34089	NA	NA	30856	-9.48%
1997	41399	NA	NA	42951	3.75%
1998	79146	NA	NA	71279	-9.94%
1999	92442	NA	NA	85313	-7.71%

On comparison of figure 6.12 with figure 6.8 a further shift of the forecasted claims towards the actual claims can be noticed. This shift has reduced the generalisation error from 0.211 to 0.187. Because the weights have remained unchanged since figure 6.8, this shift is attributable to the input vector classification of the stimuli network. A comparison of figures 6.13 and 6.9 shows further reduction of the y-direction forecasting errors.

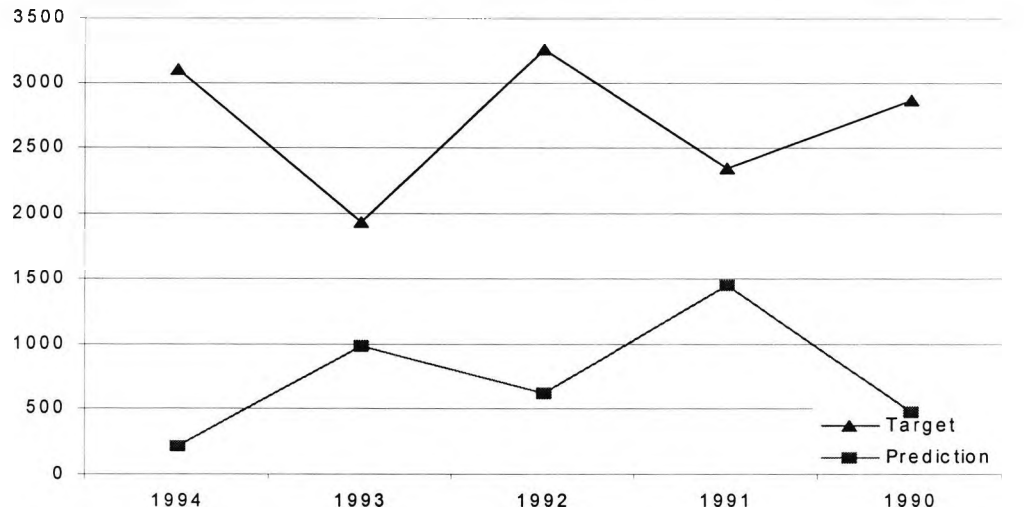


Figure 6.12 Prediction of claims reservation for x-direction after linking and stimuli network.

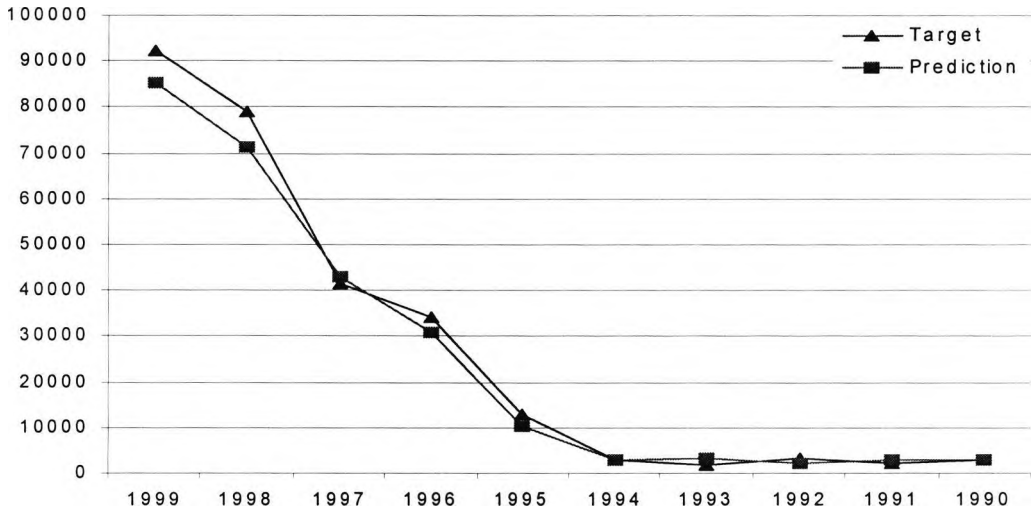


Figure 6.13 Prediction of claims reservation for y-direction after linking and stimuli network.

Table 6.26 summarises the recall and generalisation errors for 10 runs. All three stages encountered during the linking process are presented and figure 6.14 shows them graphically. The % changes for the generalisation error during linking and linking with stimuli were -14.24%, -42.53% for x-direction and -13.79%, -21.17% for y-direction.

Table 6.26 Performance comparison of trained, linked and linked with stimuli after 10 runs.

Description	x-Direction after training				y-Direction after training			
	Min	**Max**	**Average**	**StDev**	**Min**	**Max**	**Average**	**StDev**
SSE_{trn}	0.113	0.400	0.287	0.0753	0.191	0.328	0.265	0.0458
SSE_{gen}	0.573	0.839	**0.644**	0.0927	0.688	0.773	**0.723**	0.0304
	x-Direction after linking				**y-Direction after linking**			
SSE_{trn}	0.294	1.430	0.660	0.3756	0.335	0.659	0.451	0.1414
SSE_{gen}	0.362	0.755	**0.553**	0.1555	0.363	0.828	**0.623**	0.1267
	x-Direction after linking with stimuli				**y-Direction after linking with stimuli**			
SSE_{trn}	0.895	2.293	1.486	0.4069	1.168	2.567	1.690	0.3830
SSE_{gen}	0.141	0.584	**0.318**	0.1647	0.231	0.672	**0.491**	0.1534

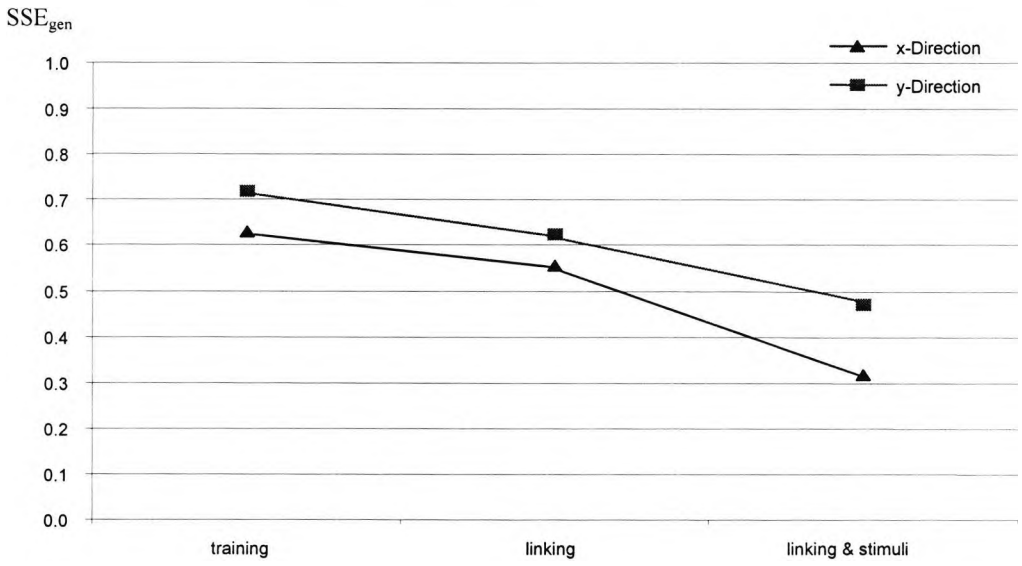


Figure 6.14 Prediction of claims reservation for y-direction after linking and stimuli network.

With the use of the stimuli network the recall errors of the x and y-directions have increased substantially. The reason for this is that the membership training for the stimuli network has been stopped to as soon as it performed well on the generalisation data, but not the data used for training and measuring the recall accuracy. Therefore the classification on the recall data is poor but good for the generalisation data, resulting in poor recall accuracy but improved generalisation. Since the objective has been to optimise the generalisation error, this has been achieved but with the price of a high recall error on the training data.

6.5.1.7 Comparison with Single Network

To supply a baseline comparison to the previous example, both datasets of x-direction and y-direction have been combined into one training file to analyse if divide and conquer did make a difference to the forecasting results.

For this purpose networks have been trained each with combined training data and generalisation data totalling to 140 training patterns and 15 testing patterns. The network used for the composite training was a standard 5:10:1 backpropagation MLP. All other important network parameters can be found in table 6.27.

Table 6.27 The parameters of the neural network used in this section.

Description	Both Domains
Input Neurons	5
Hidden Neurons	10
Output neurons	1
Activation Function	symmetric sigmoid
Initialisation	±0.2
Learning Factor	0.01
Momentum	0.3
Number of training patterns	140
Number of testing patterns	15

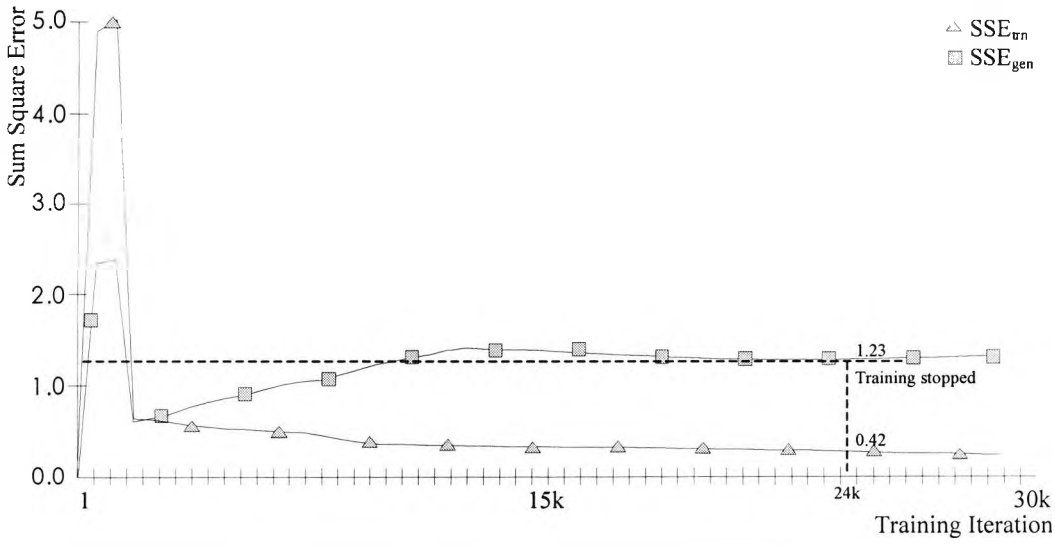


Figure 6.15 Recall and generalisation error of network trained with both domains.

Figure 6.15 shows the average recall SSE_{trn} and generalisation error SSE_{gen} during training of the composite neural network. Training has been stopped after approximately 24,000 iterations where the generalisation error reached the end of a plateau. To increase statistical reliability the entire experiment has been repeated 30 times and the averages, minimums and maximums for training and testing pattern are reported in table 6.28.

Table 6.28 Training results of single NN trained for both domains.

Description	Single neural network with both domains			
	Average	Min	Max	Std Dev
SSE_{trn}	0.4245	0.3571	0.4979	0.0418
SSE_{gen}	1.2289	1.1412	1.3873	0.0620

All forecasting results from the 30 networks trained have been de-normalised into numerical figures and are shown in table 6.29 for x-direction and in table 6.30 for y-direction. All relative errors refer to the difference between the averages and the actual Claims Occurred figures.

Table 6.29 x-Direction forecasting of single NN trained with both domains.

Year of Origin	Claims Occurred	Average	Min	Max	Std Dev	Error
1990	2872	3044	2961	3150	57	5.66%
1991	2347	-5945	-8184	-4516	696	139.48%
1992	3258	751	343	1032	176	-333.71%
1993	1932	-1009	-2068	-292	428	291.40%
1994	3100	-14798	-20414	-11680	1706	120.95%
1995	12761	NA	NA	NA	NA	NA
1996	34089	NA	NA	NA	NA	NA
1997	41399	NA	NA	NA	NA	NA
1998	79146	NA	NA	NA	NA	NA
1999	92442	NA	NA	NA	NA	NA

Figures that cannot be forecasted in the x-direction because of the window size restriction are marked NA.

Table 6.30 y-Direction forecasting of single NN trained with both domains.

Year of Origin	Claims Occurred	Average	Min	Max	Std Dev	Error
1990	2872	-4411	-6140	2872	1440	165.11%
1991	2347	-5945	-8184	-4516	696	139.48%
1992	3258	-6241	-7997	-5185	681	152.20%
1993	1932	-8433	-11920	-6515	1014	122.91%
1994	3100	-2198	-3331	-1175	619	241.03%
1995	12761	1682	918	2206	304	-658.77%
1996	34089	26293	25028	27230	549	-29.65%
1997	41399	22663	19275	25310	1833	-82.67%
1998	79146	57176	56242	58920	673	-38.43%
1999	92442	34391	27408	39302	3018	-168.80%

Comparing network training results from table 6.28 with table 6.26 requires the division of the average errors by the number of patterns since 55 training, 5 testing (x-direction), 85 training, 10 testing (y-direction) were used for table 6.26 and 140 training, 15 testing were used for table 6.28. Table 6.31 shows the RMSE to allow for an equal comparison of both tables.

Table 6.31 RMSE results from tables 6.26 and 6.28.

Type	Desc.	Patterns	Errors after training		Errors after linking		Errors after stimuli	
			Average	RMSE	Average	RMSE	Average	RMSE
x-Direction	SSE_{trn}	55	0.2875	0.00522	0.6598	0.01200	1.4861	0.02702
	SSE_{gen}	5	0.6445	0.12880	0.5527	0.11054	0.3176	0.06352
y-Direction	SSE_{trn}	85	0.2653	0.00311	0.4509	0.00530	1.6904	0.01989
	SSE_{gen}	10	0.7227	0.07230	0.6230	0.06230	0.4911	0.04911
$\sqrt{x^2 + y^2}$	SSE_{trn}	140	0.3912	**0.00279**	0.7992	0.00571	2.2508	0.01608
	SSE_{gen}	15	0.9683	**0.06455**	0.8328	0.05552	0.5848	0.03899
Composite	SSE_{trn}	140	0.4245	**0.00303**	NA	NA	NA	NA
	SSE_{gen}	15	1.2289	**0.08192**	NA	NA	NA	NA

It can be noticed that the training results of the composite network are worse than the sum of SSE errors for the x- and y-direction. The generalisation error of the composite network is also higher. Furthermore, if the composite network is compared to the linked or even to the linked with stimuli network, the composite network performed even worse.

6.5.2 Claims Reservation with Data from two Companies

In the previous section 6.5.1, data of one company has been used to create two distinguished domains by generating two distinct time series. In this section, data from two companies will be used to generate two distinct time series. It has been taken into account that both companies have been writing comparable business types in order to contribute knowledge to each other. For this purpose, Co-operative Insurance Society Ltd and Legal&General Insurance Ltd have been chosen since both write business that can be classified by the risk group: "Private Car Comprehensive, Claims reported and outstanding excluding INBR".

Probably the most likely scenario of training neural networks is where data from one or more company is used for training. It can then be used to produce a trained network for the purpose of forecasting. A problematic situation will occur if companies are to be added or removed frequently from the data model. With linking, the re-training of networks containing data from all companies can be avoided since only sub-networks need re-training. This section will discuss the framework for linking two companies.

6.5.2.1 Training Data Preparation

In section 6.5.1 where one company's data was used for creation of time series information in x and y-direction, training success for x-direction was moderate compared to y-direction. Consequently, the time series used for training of both companies in this section will be exclusively for the y-direction. Data for training and testing will very much follow the arrangement outlined in section 6.5.1.1 for the y-direction referred to as the year of origin. Table 6.32 and table 6.33 show incremental claims reported and outstanding excluding INBR for Co-operative and Legal&General. The creation of time series patterns for Co-operative from table 6.32 starts at 1981:1 with 14846 as the first input and ends with 25018 as the output. Subsequent patterns should be 8628...11680, 4670...11107 with the last one being 34028...47350. All data from 1999 should be forecasted and should therefore be separated from the training data. The first pattern in this file should start with 33770

as the first input and 61365 as the target output. Subsequent patterns should be 22923...27901, 22759...21894 with the last one being 116...1031. Since all patterns are normalised on a vector-by-vector basis, training and testing data sets can be normalised individually. If the data were to be normalised on a column basis, the minimal and maximal figures of the entire data set (or each column) would be needed, thus preventing the split into training and testing set prior to normalisation.

Table 6.32 Incremental claims triangle for Co-operative.

Year of Origin	Development Year									
	1	2	3	4	5	6	7	8	9	10
1981	14846	8628	6470	4250	3012	2097	1468	1050	848	422
1982	14210	7409	5160	3844	3294	2611	1586	1087	976	384
1983	12245	6038	4505	3467	2594	1670	1182	638	418	299
1984	16073	6570	5059	4107	3646	3706	2681	2148	1455	965
1985	17306	10533	7416	6247	4837	3562	3492	1443	1005	116
1986	25018	11680	11107	8875	7381	7190	5349	4891	4194	4874
1987	34384	18729	16794	13742	10213	8101	6845	4115	2449	1475
1988	41929	21434	20014	18696	14530	12006	11257	7050	4079	2425
1989	46408	24967	22191	20284	16270	13562	9975	7194	5838	5462
1990	52032	30601	23004	17895	13215	8468	4394	2557	2318	1031
1991	46485	31425	28117	25181	19828	15215	10804	8567	6382	
1992	38394	25693	22759	20607	18191	12123	8780	2690		
1993	34028	22923	20765	14471	11602	7878	3271			
1994	33770	20467	14833	13149	9395	5644				
1995	35103	19002	15599	14781	14401					
1996	35115	19022	22483	24967						
1997	37958	22279	21894							
1998	47350	27901								
1999	61365									

The creation of time series patterns for Legal&General from table 6.33 starts at 1981:1 with 3932 as the first input and ends with 6039 as the output. Subsequent patterns should be 1927...3014, 1383...2227 with the last one being 7783...47350. The 1999 forecasting data should start with 7573 as the first input and 7177 as the target output. Subsequent patterns should be 4027...6231, 2782...2436 with the last one being 313...55. With this, the total number of training patterns created for each company is 85 and 10 patterns for testing.

After creation of all training and testing patterns, a vector based normalisation has been used as shown in equation (6.6).

Table 6.33 Incremental claims triangle for Legal&General.

Year of Origin	Development Year									
	1	2	3	4	5	6	7	8	9	10
1981	3932	1927	1383	1078	1264	193	94	41	29	17
1982	3049	1154	656	364	188	180	153	128	109	32
1983	3361	1372	1061	942	760	788	713	175	72	74
1984	5120	2306	1878	1350	754	577	566	357	195	155
1985	5241	2257	1610	1237	1005	607	678	564	330	313
1986	6039	3014	2227	1738	1595	972	428	170	107	48
1987	4925	2246	2357	1885	1668	1459	1325	1701	1713	343
1988	5200	2348	2051	2216	1749	840	617	138	43	69
1989	4695	2224	1794	1935	1367	741	468	451	113	78
1990	5767	2844	2233	1963	1118	484	365	182	58	55
1991	6780	3349	2250	1723	1405	990	753	441	-1	
1992	7795	3815	2782	2536	1693	502	634	190		
1993	7783	4027	3254	2300	967	609	113			
1994	7573	5027	3889	3344	1833	1326				
1995	5576	2861	2160	1581	747					
1996	6126	3887	3056	1990						
1997	6807	4426	2436							
1998	9338	6231								
1999	7177									

It can be noticed that the claims figures of Co-operative are about 6 times higher for the first 4 years of development compared to Legal&General. This is because Co-operative has on average 4 times more claims during the same interval than Legal&General (source: Insight Non-Life A.M. Best International).

6.5.2.2 Training of Neural Networks

Two MLP neural networks have been trained with the normalised data of both companies as referred to in the previous section. The topology of both networks used has been chosen to be 5:10:1 with the output layer set to 1 and frozen. All other important network parameters can be found in table 6.34.

Table 6.34 The parameters of the neural networks used in this section.

Description	Co-operative	Legal&General
Input Neurons	5	5
Hidden Neurons	10	10
Output neurons	1	1
Activation Function	symmetric sigmoid	symmetric sigmoid
Initialisation	±0.7	±0.7
Learning Factor	0.01	0.01
Momentum	0.5	0.5
Number of training patterns	85	85
Number of testing patterns	10	10

Figure 6.16 shows the recall SSE_{trn} and generalisation error SSE_{gen} during training of Co-operative. Training has been stopped after approximately 40,000 iterations where the generalisation error reached local minima of 0.1982. At this point the recall error was only 0.0586, which is very low indeed. With the low generalisation and recall errors, inclusion of information from another company seems almost unnecessary. Despite this, a second network has been trained with data from Legal&General for the purpose of linking.

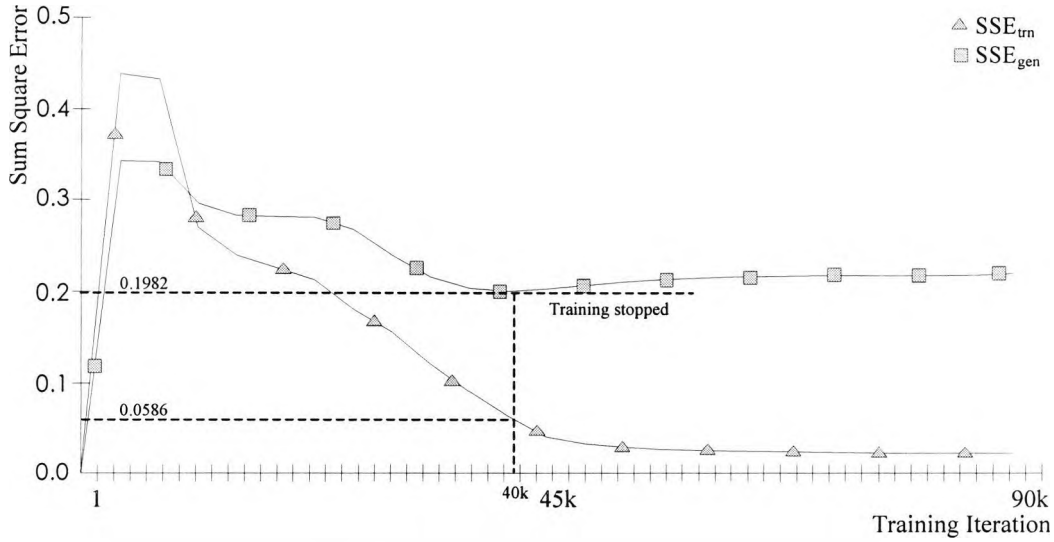


Figure 6.16 Recall and generalisation error of network trained with Co-operative data.

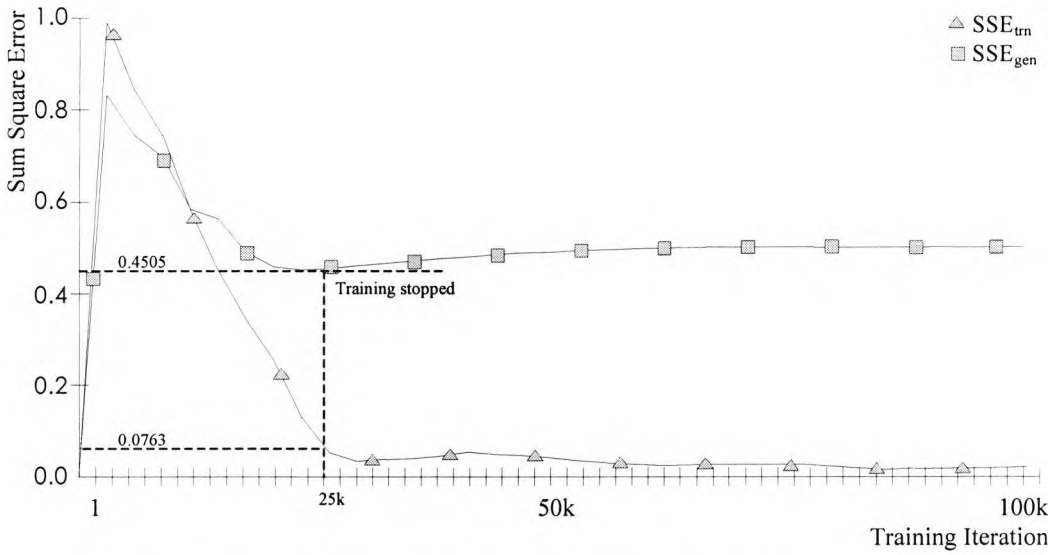


Figure 6.17 Recall and generalisation error of network trained with Legal&General data.

Figure 6.17 shows the recall SSE_{trn} and generalisation error SSE_{gen} during training of Legal&General. Training has been stopped after approximately 25,000 iterations where the generalisation error reached local minima of 0.4505. At this point the recall error was only 0.0786, which is again very low. The generalisation error of Legal&General is almost twice as high as the error for Co-operative. Therefore moving knowledge from Co-operative into Legal&General might be beneficial to Legal&General, but if any improvement on the generalisation error for Co-operative can be achieved is uncertain.

Table 6.35 shows a direct comparison between the occurred claims and forecasted claims for both insurance companies after training. It can be noticed that the highest error in the Co-operative forecast is only 48%, with an average of only 15.06%. On the other hand, Legal&General has high errors e.g. 100.21% and 68.07%, with an average of 32.02%, twice as high as Co-operative.

With both companies, the bulk of the higher errors are within the first four years from 1990-1993. This is probably caused by the low target values for these years. Because

the error is given as percentage, any forecast differing by a few hundreds will have a high impact on the percentage error.

Table 6.35 Forecasting results for Co-operative and Legal&General after training.

Year of Origin	Occurred	Co-operative	Error	Occurred	Legal&General	Error
1990	1031	1991	48.23%	55	145	62.11%
1991	6382	6434	0.81%	-1	485	100.21%
1992	2690	4199	35.94%	190	595	68.07%
1993	3271	5223	37.38%	113	283	60.04%
1994	5644	6511	13.31%	1326	1372	3.37%
1995	14401	13560	-6.20%	747	913	18.19%
1996	24967	24304	-2.73%	1990	1963	-1.38%
1997	21894	22955	4.62%	2436	2467	1.25%
1998	27901	27627	-0.99%	6231	6114	-1.92%
1999	61364	61156	-0.34%	7177	6922	-3.69%

Figures 6.18 and 6.19 are graphical illustrations of the numerical values from table 6.35 for Co-operative and Legal&General respectively. Both graphs show that the forecasting results for both insurance companies are excellent. In figure 6.18, only the years 1997, 1993 and 1992 show room for improvement and in figure 6.19, the years 1990-1993, 1995 and 1999 show room for improvement.

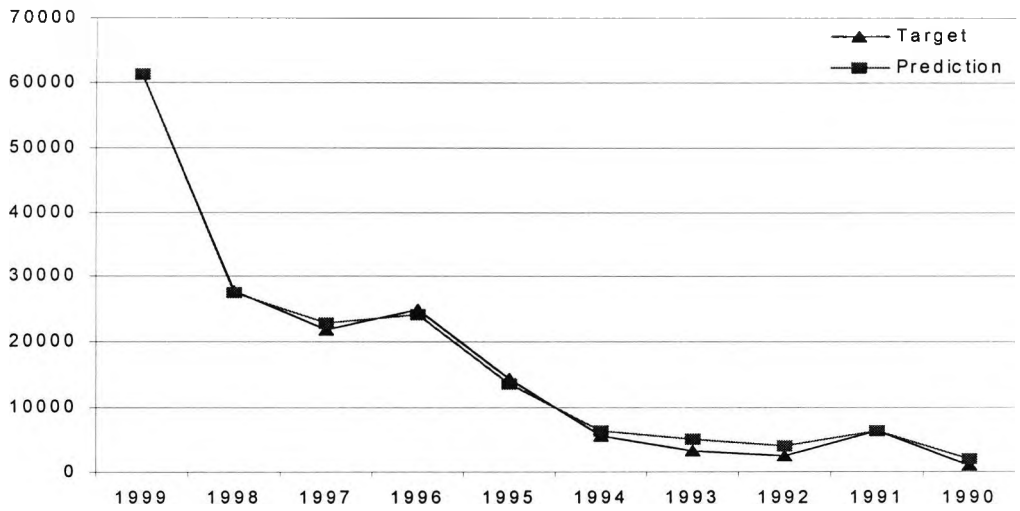


Figure 6.18 Prediction of claims reservation for Co-operative after training.

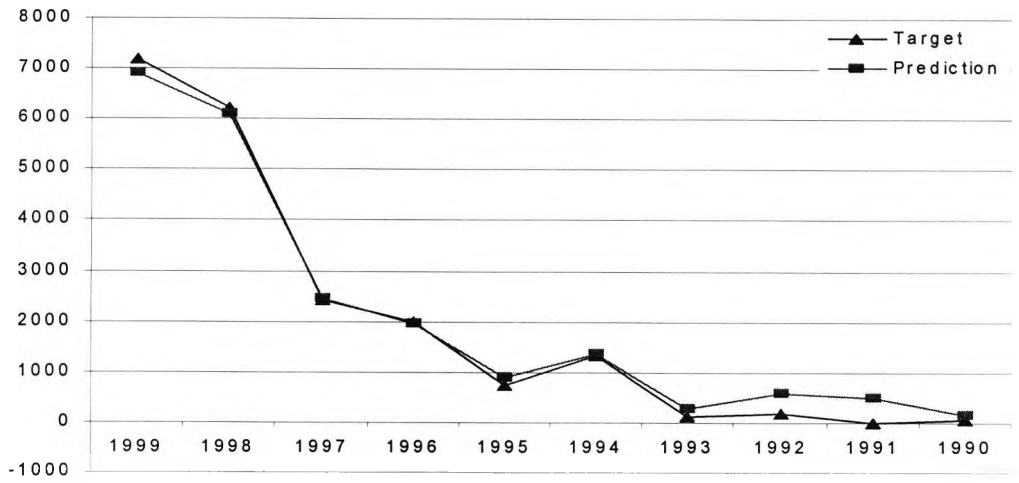


Figure 6.19 Prediction of claims reservation for Legal&General after training.

To increase statistical reliability the experiment has been repeated 10 times up to this point and the averages, minimums and maximums for training and testing pattern are reported. A summary of generalisation and recall errors after 10 runs for both companies can be found in table 6.36. Legal&General reached its minimum generalisation error point on average after 25,000 iterations, much earlier than Co-operative, which reached its minimum generalisation error point on average after 40,000 iterations. Continuous training of Legal&General beyond 25,000 iterations only yielded an increase in generalisation error as indicated in figure 6.17.

Table 6.36 Performance benchmarks of both networks after training for 10 runs.

Description	Co-operative after training				Legal&General after training			
	Min	Max	Average	StDev	Min	Max	Average	StDev
SSE_{trn}	0.0255	0.0667	0.0499	0.0158	0.0499	0.0863	0.0736	0.0108
SSE_{gen}	0.1338	0.2230	0.1878	0.0326	0.3519	0.5584	0.4504	0.0788

The weight matrixes of the hidden layer from both insurance companies are shown in tables 6.37 and 6.38. During training, the algorithm for the prevention of saturated neurons from section 4.6, equation (4.17) was used. Therefore no excessively large

weights can be found in both weight matrixes and the highest and lowest weights are in the region of ±6. It shall be repeated that weights connecting the hidden neurons to the output neurons are set to 1 and were frozen during training to prevent alterations.

Table 6.37 Hidden layer weight matrix of network trained for Co-operative.

Reference	Co-operative						
	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{1B}	Length
v_1	2.1495	-0.3596	2.0154	0.0173	-1.5016	1.1796	3.5296
v_2	-2.1826	-1.1504	-1.5728	-5.7996	-0.2293	5.7315	8.6659
v_3	-2.1204	-0.7700	2.2681	1.8108	1.7291	-0.6245	4.1099
v_4	0.1954	0.5565	-1.2248	-1.0643	-1.0793	1.1519	2.3394
v_5	-0.3229	0.1614	0.0433	0.2695	-0.1806	-2.0185	2.0765
v_6	-0.4960	0.0064	-0.1975	0.0336	-0.2105	-1.4475	1.5575
v_7	-1.6915	-1.4217	-3.1498	0.9425	1.6507	2.9953	5.2333
v_8	1.5662	0.4022	0.1878	-0.2764	-2.6136	2.5917	4.0341
v_9	-0.1234	-0.6458	-4.0680	1.2124	-2.0004	3.5142	5.8993
v_{10}	-0.2752	-3.5429	0.1869	-4.3828	-2.4116	5.9578	8.5547

Table 6.38 Hidden layer weight matrix of network trained for Legal&General.

Reference	Legal&General						
	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{1B}	Length
v_1	-3.2228	1.9822	-5.5717	1.3595	-4.0979	4.8785	9.3702
v_2	-4.0550	-2.6476	-0.3744	-6.7180	0.1078	5.7267	10.0763
v_3	0.1306	0.2276	-1.1178	0.3580	-1.4020	-0.7603	1.9976
v_4	-0.3927	0.6945	0.9167	-0.3111	-0.6912	0.3546	1.4756
v_5	0.6112	-0.8240	3.1986	-1.6984	-2.4395	1.7406	4.8114
v_6	2.3534	-3.4938	-1.3883	0.6541	0.8635	2.5418	5.2257
v_7	0.2291	2.4486	3.5602	0.3380	-0.9691	-0.0907	4.4480
v_8	-0.5930	-4.8081	-0.9721	2.3082	-0.7187	1.7282	5.7658
v_9	-0.7853	1.0394	-0.0427	1.1062	3.4995	-1.3572	4.1244
v_{10}	3.6956	1.6511	-2.8522	-2.4665	-0.7880	1.9100	5.9052

The next steps involve vector angle calculations and vector-by-vector comparison to find hidden neurons that can be linked. The acceptance angle has been set to 20° but might be reduced or increased in order to find at least two hidden neurons that can be linked to match the linking process from the previous section 6.5.1.

6.5.2.3 Linking of Domain Networks

The linking of both weight matrixes from tables 6.37 and 6.38 follows exactly the same procedures as demonstrated in 6.5.1.3 and former sections.

The first two vectors that have the closest angles are listed in table 6.39. It can be noticed that the vectors v_3 and v_{10} in the second row have an angle that exceeds the acceptance angle of 20° that has been agreed on initially. But for the purpose of consistency and comparability with section 6.5.1, where two neurons were linked, the acceptance angle has been increased to 22° so that two vectors can be linked.

Table 6.39 Angles between weight vectors in ascending order.

Vector pair		Angle between vectors
Co-operative	Legal&General	
v_2	v_2	15.25°
v_3	v_{10}	21.39°

Table 6.40 shows the resulting vectors and their associated length correction factors F_2. The negative vector length correction factor F_2 in the second row indicates that both vectors (v_3 and v_{10}) were pointing in opposite directions.

Table 6.40 Results of the combination of vectors with angles below 22°as listed in table 6.39.

Original vector references		Resulting vector v_{r1}						
Co-operative	Legal&General	W_{11}	w_{12}	w_{13}	w_{14}	W_{15}	w_{1B}	Factor F_2
v_2	v_2	-3.0547	-1.9714	-1.2061	-5.6537	-0.1711	5.1475	1.1686
v_3	v_{10}	-2.3627	-1.0377	1.9632	1.6569	1.0859	-1.2060	-1.4655

6.5.2.4 Linking Analysis

Table 6.41 shows the relative errors between vector components of the trained weights and the weights after linking. It can be observed that largest component errors of over 200% are within the Legal&General vector with the lower acceptance angle of 15.25°. This is somewhat unexpected because the higher errors can generally be

found with vectors that have a higher angle difference. On further inspection of table 6.41 it can be noticed that all errors are quite large if compared to table 6.14 from section 6.5.1.4. This can be seen as a warning that the information contained in both vectors may not necessarily be of the same kind.

Table 6.41 Vector component change impact analysis.

Reconstructed Vectors	Vector components					
	w'_{11}	w'_{12}	w'_{13}	w'_{14}	w'_{15}	w'_{1B}
Coop: v'_2	-3.0547	-1.9714	-1.2061	-5.6537	-0.1711	5.1475
L&G: v'_2	-3.5698	-2.3038	-1.4095	-6.6070	-0.2000	6.0155
Coop: v'_3	-2.3627	-1.0377	1.9632	1.6569	1.0859	-1.2060
L&G: v'_{10}	3.4626	1.5207	-2.8771	-2.4282	-1.5913	1.7674
Reconstructed Vectors	**Relative Errors**					
	$f(w_{11}, w'_{11})$	$f(w_{12}, w'_{12})$	$f(w_{13}, w'_{13})$	$f(w_{14}, w'_{14})$	$f(w_{15}, w'_{15})$	$f(w_{1B}, w'_{1B})$
Coop: v'_2	39.96%	71.37%	-23.31%	-2.52%	-25.38%	-10.19%
L&G: v'_2	-11.97%	-12.99%	276.45%	-1.65%	-285.51%	5.04%
Coop: v'_3	11.43%	34.76%	-13.44%	-8.50%	-37.20%	93.13%
L&G: v'_{10}	-6.30%	-7.89%	0.87%	-1.55%	101.93%	-7.46%

Table 6.42 lists the relative errors of the vector length change. It can be observed that the largest errors rest with the vectors v_3:v_{10} since the angle between both vectors is the largest.

Table 6.42 Vector length change impact analysis.

Vector	Original length	Reconstructed length		Relative Error
Coop: v'_2	8.6659	8.5535	($\|v_{r1}\|$)	-1.30%
L&G: v'_2	10.0763	9.9958	($\|v_{r1}\|*F_2$)	-0.80%
Coop: v'_3	4.1099	3.9866	($\|v_{r1}\|$)	-3.00%
L&G: v'_{10}	5.9052	5.8423	($\|v_{r1}\|*F_2$)	-1.07%

The reconstructed weight matrixes for both companies are shown in tables 6.43 and 6.44. Rows that are framed and shown on grey backgrounds are vectors that have been replaced with reconstructed vectors.

In the previous section an analysis on the linked network has been made without the use of a stimuli network. This step has been left out for this experiment and the linked network has been utilised directly with a stimuli network.

Table 6.43 Hidden layer weight matrix of Co-operative after reconstruction.

Reference	x-Direction						
	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{1B}	Length
v_1	2.1495	-0.3596	2.0154	0.0173	-1.5016	1.1796	3.5296
v_2	-3.0547	-1.9714	-1.2061	-5.6537	-0.1711	5.1475	8.5535
v_3	-2.3627	-1.0377	1.9632	1.6569	1.0859	-1.2060	3.9866
v_4	0.1954	0.5565	-1.2248	-1.0643	-1.0793	1.1519	2.3394
v_5	-0.3229	0.1614	0.0433	0.2695	-0.1806	-2.0185	2.0765
v_6	-0.4960	0.0064	-0.1975	0.0336	-0.2105	-1.4475	1.5575
v_7	-1.6915	-1.4217	-3.1498	0.9425	1.6507	2.9953	5.2333
v_8	1.5662	0.4022	0.1878	-0.2764	-2.6136	2.5917	4.0341
v_9	-0.1234	-0.6458	-4.0680	1.2124	-2.0004	3.5142	5.8993
v_{10}	-0.2752	-3.5429	0.1869	-4.3828	-2.4116	5.9578	8.5547

Table 6.44 Hidden layer weight matrix of Legal&General after reconstruction.

Reference	y-Direction						
	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{1B}	Length
v_1	-3.2228	1.9822	-5.5717	1.3595	-4.0979	4.8785	9.3702
v_2	-3.5698	-2.3038	-1.4095	-6.6070	-0.2000	6.0155	9.9958
v_3	0.1306	0.2276	-1.1178	0.3580	-1.4020	-0.7603	1.9976
v_4	-0.3927	0.6945	0.9167	-0.3111	-0.6912	0.3546	1.4756
v_5	0.6112	-0.8240	3.1986	-1.6984	-2.4395	1.7406	4.8114
v_6	2.3534	-3.4938	-1.3883	0.6541	0.8635	2.5418	5.2257
v_7	0.2291	2.4486	3.5602	0.3380	-0.9691	-0.0907	4.4480
v_8	-0.5930	-4.8081	-0.9721	2.3082	-0.7187	1.7282	5.7658
v_9	-0.7853	1.0394	-0.0427	1.1062	3.4995	-1.3572	4.1244
v_{10}	3.4626	1.5207	-2.8771	-2.4282	-1.5913	1.7674	5.8423

6.5.2.5 Training of Stimuli Network

Stimuli networks need to be able to classify incoming input vectors into domain memberships. All details about the stimuli network previously mentioned in section 6.5.1.5 will apply to this section. Again, the domain memberships of the input vectors have been generated by means of linear search as described earlier in section 6.5.1.5 and figure 6.10. Table 6.45 shows an extract of the training data used for the stimuli network training of both domains.

Table 6.45 Stimuli network training data with memberships generated by linear search.

x-Direction							y-Direction						
in 1	**in 2**	**in 3**	**in 4**	**in 5**	**out A**	**out B**	**in 1**	**in 2**	**in 3**	**in 4**	**in 5**	**out A**	**out B**
0.11	0.47	0.27	0.45	0.65	0.65	0.48	0.34	0.40	0.46	0.46	0.45	0.61	0.51
0.12	0.40	0.33	0.57	0.59	0.35	0.72	0.35	0.26	0.27	0.43	0.65	0.62	0.46
0.12	0.01	0.62	0.19	0.31	0.63	0.30	0.35	0.40	0.42	0.53	0.30	0.74	0.38

Because of the layout of the training data, the stimuli network required 5 inputs and 2 outputs. The number of hidden units used remained 10 thus resulting in a 5:10:2 network, unchanged to section 6.5.1.5. Table 6.46 is presenting a list of all significant network parameters used for the stimuli network.

Table 6.46 The parameters of the stimuli network.

Description	Setting
Input Neurons	5
Hidden Neurons	10
Output neurons	2
Activation Function	symmetric sigmoid
Initialisation	±0.7
Learning Factor	0.01
Momentum	0.3
Number of training patterns	130
Number of testing patterns	40

Figure 6.19 shows how the recall SSE_{trn} and generalisation error SSE_{gen} change during training. Training has been stopped after approximately 400k iterations where the generalisation error reached a minimum of 1.135. At this point the recall error was

1.847, which is relatively large. The recall error SSE_{trn} is generally increasing the more training patterns are used. To acquire a comparable figure that is independent on the number of patterns, the RMSE needs to be calculated. Table 6.47 shows a summary of generalisation and recall errors for the stimuli network. A comparison between tables 6.47 and 6.24 confirms that the performances of both stimuli networks are somewhat comparable.

This large number of 400k iterations was mainly caused by the low learning factor that was set to 0.01. Because the network was small and not many training records were used a 3.6 GHz P4, 2GB RAM computer (2004), the approximate training time needed was only 6 minutes.

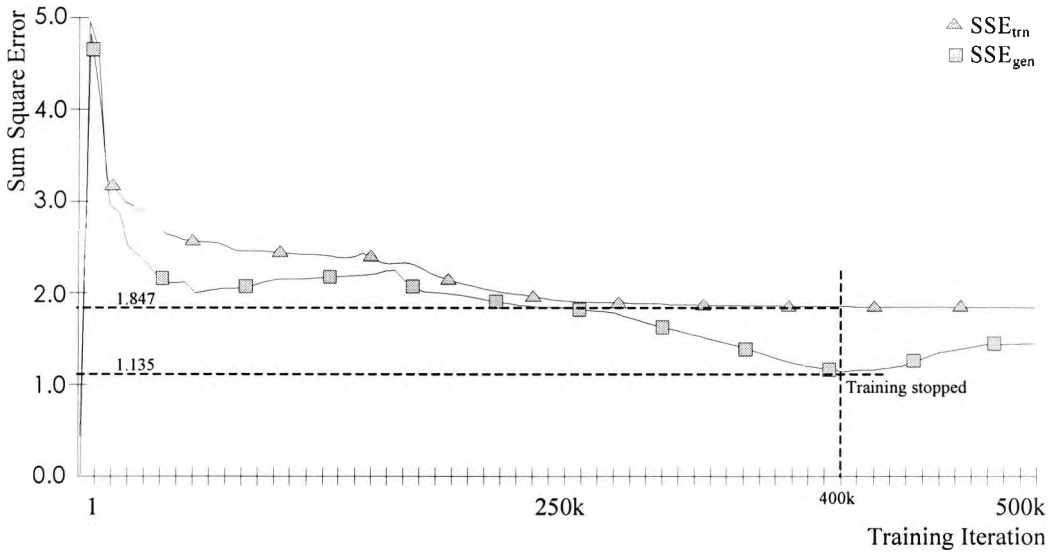


Figure 6.20 Recall and generalisation error of network trained with Legal&General data.

Table 6.47 Performance benchmarks of stimuli network after training.

Description	Stimuli Network
SSE_{trn}	1.847
SSE_{gen}	1.135
RMSE for SSE_{trn}	0.162 (130 records)
RMSE for SSE_{gen}	0.179 (40 records)
Iterations	400,000

6.5.2.6 Linking Results

To evaluate the impact of linking with utilisation of a stimuli network, the forecasting of claims reservation for 1999 has been repeated and is shown in table 6.48. If this table is compared with table 6.35 improvements can only be noticed in the early years from 1990 to 1994. A graphical illustration of the forecasting results listed in table 6.48 for both insurance companies is presented in figures 6.21 and 6.22.

Table 6.48 Forecasting results for Co-operative and Legal&General after linking.

Year of Origin	Occurred	Co-operative	Error	Occurred	Legal&General	Error
1990	1031	1868	44.81%	55	4	-1352.71%
1991	6382	5561	-14.77%	-1	11	109.43%
1992	2690	2093	-28.54%	190	286	33.65%
1993	3271	3902	16.18%	113	187	39.61%
1994	5644	8975	37.11%	1326	1548	14.33%
1995	14401	13387	-7.57%	747	403	-85.54%
1996	24967	33145	24.67%	1990	2242	11.24%
1997	21894	8046	-172.10%	2436	1984	-22.76%
1998	27901	35511	21.43%	6231	6033	-3.28%
1999	61364	69585	11.81%	7177	6527	-9.96%

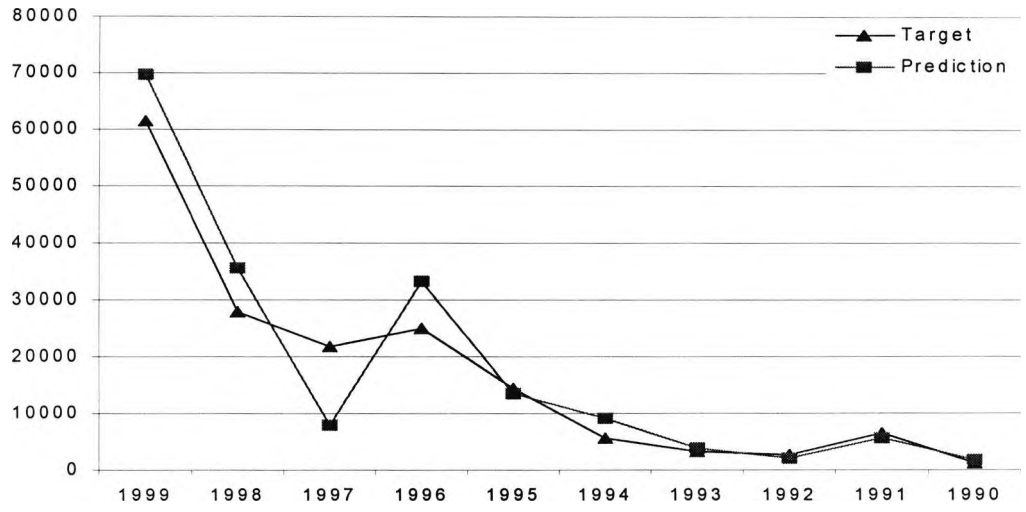


Figure 6.21 Prediction of claims reservation for Co-operative after linking and stimuli network.

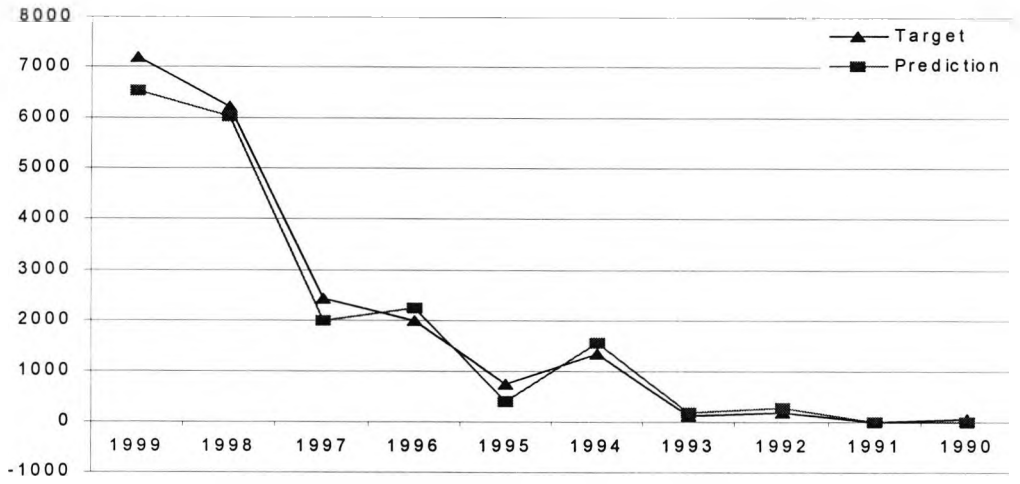


Figure 6.22 Prediction of claims reservation for Legal&General after linking and stimuli network.

Comparing figures 6.21 and 6.18 with utilisation of table 6.48 shows that only the forecasts for the years 1990, 1992 and 1993 have reduced generalisation errors of 3.42%, 7.40% and 21.20%. The years 1997 and 1996 have had the highest increase in errors of 167.48% and 21.94%. Comparing figures 6.22 and 6.19 with utilisation of table 6.48 shows that only the forecasts for the years 1992 and 1993 have reduced generalisation errors of 34.42% and 20.43%. The years 1990 and 1995 have had the highest increase in errors of 1290.60% and 67.35%.

Table 6.49 Performance comparison of trained, linked and linked with stimuli after 10 runs.

Description	Co-operative after training				Legal&General after training			
	Min	**Max**	**Average**	**StDev**	**Min**	**Max**	**Average**	**StDev**
SSE_{trn}	0.0255	0.0667	0.0499	0.0158	0.0499	0.0863	0.0736	0.0108
SSE_{gen}	0.1338	0.2230	**0.1878**	0.0326	0.3519	0.5584	**0.4504**	0.0788
	Co-operative after linking				**Legal&General after linking**			
SSE_{trn}	0.1056	0.4036	0.2392	0.0956	0.0824	0.8415	0.4726	0.3019
SSE_{gen}	0.1126	0.2765	**0.1928**	0.0582	0.3271	0.5450	**0.4226**	0.0889
	x Co-operative linking with stimuli				**Legal&General linking with stimuli**			
SSE_{trn}	0.4916	1.2953	0.7870	0.2351	0.5629	1.2054	0.8533	0.2265
SSE_{gen}	0.1390	0.4923	**0.2600**	0.1040	0.1319	0.4743	**0.2763**	0.1176

Even with the generalisation error reduced by 0.1741 (38.65%) from 0.4504 to 0.2763 , as shown in table 6.49, the forecasting results for Legal&General do not seem to have improved whatsoever. The reason the generalisation error has fallen is that the

main contributors to the error, shown bordered in table 6.50, have been reduced during linking, as indicated by arrows. This decrease in difference prior to denormalisation has not caused a relative percentage reduction after denormalisation. The reason for this is that the relative error has been measured with regards to the forecasted figure, not the target figure as show in equation (6.7). Therefore the improvements for the years 1990-1994 can be seen in the normalised network output but are not evident in the percentage figures.

$$Err_{rel} = \left(\frac{forecast - t\arg et}{forecast} \right) \tag{6.7}$$

Table 6.50 Error analysis for Legal&General.

Normalised target (neural network training target)	Neural network output after training	Neural network output after linking	Difference between target and output after training	Difference between target and output after linking
0.1250	0.3200	0.0087	0.1950 →	0.1164
-0.0006	0.3001	0.0068	0.3007 →	0.0074
0.1132	0.3417	0.1697	0.2285 →	0.0565
0.0946	0.2332	0.1559	0.1385 →	0.0613
0.6147	0.6300	0.6839	0.0153	0.0691
0.2434	0.2946	0.1331	0.0512	0.1103
0.3670	0.3625	0.4084	0.0045	0.0414
0.3511	0.3552	0.2902	0.0041	0.0610
0.5494	0.5412	0.5355	0.0082	0.0139
0.4200	0.4068	0.3860	0.0132	0.0339
Claims occured (target)	**Forecast after training**	**Forecast after linking**	**Error after training**	**Error after linking**
55	145	4	62.11%	-1352.71%
-1	485	11	100.21%	109.43%
190	595	286	68.07%	33.65%
113	283	187	60.04%	39.61%
1326	1372	1548	3.37%	14.33%
747	913	403	18.19%	-85.54%
1990	1963	2242	-1.38%	11.24%
2436	2467	1984	1.25%	-22.76%
6231	6114	6033	-1.92%	-3.28%
7177	6922	6527	-3.69%	-9.96%

6.5.2.7 Comparison with Single Network

To supply a baseline comparison to the previous example, both datasets of Co-operative and Legal&General have been combined into one training file to analyse if divide and conquer did make a difference to the forecasting results.

For this purpose networks have been trained each with combined training data and generalisation data totalling to 170 training patterns and 20 testing patterns. The network used for the composite training was a standard 5:10:1 backpropagation MLP. All other important network parameters can be found in table 6.51.

Table 6.51 The parameters of the neural networks used in this section.

Description	Both Domains
Input Neurons	5
Hidden Neurons	10
Output neurons	1
Activation Function	symmetric sigmoid
Initialisation	±0.2
Learning Factor	0.001
Momentum	0.3
Number of training patterns	170
Number of testing patterns	20

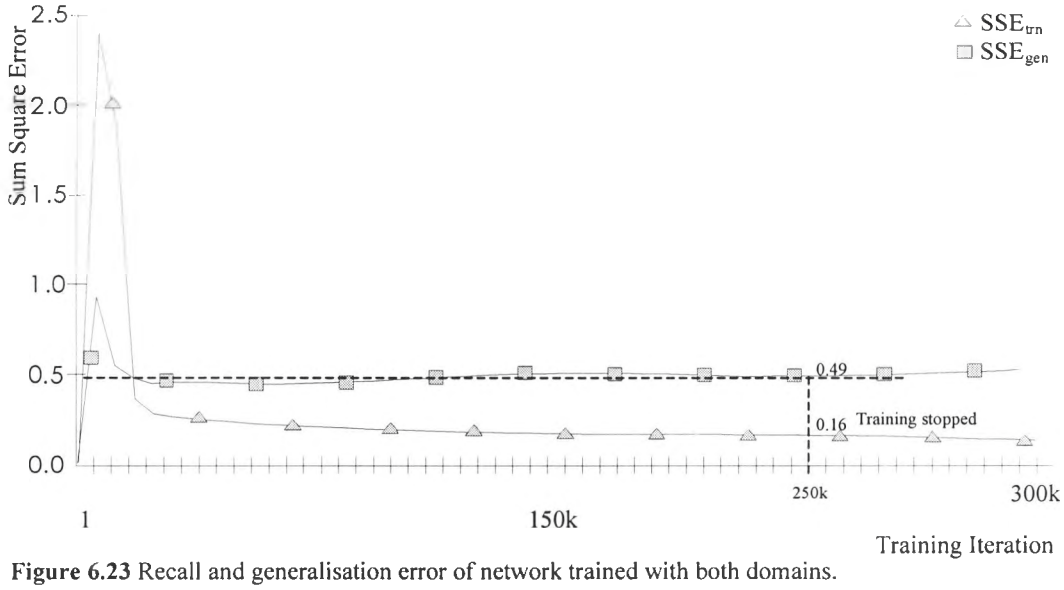


Figure 6.23 Recall and generalisation error of network trained with both domains.

Figure 6.23 shows the average recall SSE_{trn} and generalisation error SSE_{gen} during training of the composite neural network. Training has been stopped after approximately 250,000 iterations where the generalisation error reached the end of a plateau. To increase statistical reliability the entire experiment has been repeated 30 times and the averages, minimums and maximums for training and testing pattern are reported in table 6.52.

Table 6.52 Training results of single NN trained for both domains.

Description	Single neural network with both domains			
	Average	Min	Max	Std Dev
SSE_{trn}	0.1650	0.0877	0.2592	0.0541
SSE_{gen}	0.4977	0.3869	0.6241	0.0649

All forecasting results from the 30 networks trained have been de-normalised into numerical figures and are shown in table 6.53 for Co-operative and in table 6.54 for Legal&General. All relative errors refer to the difference between the averages and the actual Claims Occurred figures.

Table 6.53 Co-Operative forecasting of single NN trained with both domains.

Year of Origin	Claims Occurred	Average	Min	Max	Std Dev	Error
1990	1031	1231	576	2543	469	16.25%
1991	6382	6633	6019	7225	303	3.79%
1992	2690	3688	3178	4353	309	27.07%
1993	3271	4730	3519	6078	493	30.85%
1994	5644	7340	4567	8678	1049	23.10%
1995	14401	13657	12807	14666	488	-5.45%
1996	24967	22459	19451	24962	1333	-11.17%
1997	21894	20117	18739	23532	1194	-8.83%
1998	27901	26252	24791	28364	933	-6.28%
1999	61364	60663	58415	63103	1314	-1.16%

Compared to table 6.29, all years can be forecast since the data preparation for the Year of Development permits a trail long enough for creation of training patterns.

Table 6.54 Legal&General forecasting of single NN trained with both domains.

Year of Origin	Claims Occurred	Average	Min	Max	Std Dev	Error
1990	55	97	-9	180	45	43.26%
1991	-1	464	196	640	124	100.22%
1992	190	706	190	1192	229	73.09%
1993	113	277	240	319	22	59.24%
1994	1326	1229	1109	1303	52	-7.88%
1995	747	862	768	923	43	13.32%
1996	1990	1975	1737	2245	116	-0.75%
1997	2436	2259	2065	2388	91	-7.85%
1998	6231	5847	5367	6616	297	-6.56%
1999	7177	6804	6438	7238	185	-5.49%

Comparing network training results from table 6.49 with table 6.52 requires the division of the average errors by the number of patterns since 85 training and 10 testing patterns were used for Co-operation and Legal&General in table 6.49 and 170 training and 20 testing patterns were used for table 6.52. Table 6.55 shows the RMSE to allow for an equal comparison of both tables.

Table 6.55 RMSE results from tables 6.49 and 6.52.

Type	Desc.	Patterns	Errors after training		Errors after linking		Errors after stimuli	
			Average	RMSE	Average	RMSE	Average	RMSE
Co-operative	SSE_{trn}	85	0.0499	0.00059	0.2392	0.00281	0.7870	0.00926
	SSE_{gen}	10	0.1878	**0.01878**	0.1928	**0.01928**	0.2600	**0.02600**
Legal& General	SSE_{trn}	85	0.0736	0.00087	0.4726	0.00556	0.8533	0.01004
	SSE_{gen}	10	0.4504	**0.04504**	0.4226	**0.04226**	0.2763	**0.02763**
Co-op + L&G	SSE_{trn}	170	0.0889	**0.00052**	0.5297	0.00312	1.1608	0.00683
	SSE_{gen}	20	0.4880	**0.02440**	0.4645	0.02323	0.3794	**0.01897**
Composite	SSE_{trn}	170	0.1650	**0.00097**	NA	NA	NA	NA
	SSE_{gen}	20	0.4977	**0.02489**	NA	NA	NA	NA

It can be noticed that the training results of the composite network are worse than the summed errors for Co-operative and Legal&General (Co-op+L&G). Whilst the generalisation for Co-operative increases after linking and after stimuli, it reduces for Legal&General. Nevertheless, the combined generalisation error (Co-op+L&G) has been reduced after linking and after stimuli.

6.6 Conclusions

It can be concluded that claims reservation can be accurately forecast using MLP neural networks with five input one output time series training data. The training patterns created in the direction of year of origin (y-direction) seem to achieve better training results than patterns created in the direction of development year (x-direction). Vector based training pattern normalisation on 5:10:1 neural network have performed well and linking with acceptance angle of 20° seems to cause high loss of recall accuracy, although this was expected. The generation of stimuli training patterns by applying input vectors and target values by means of linear search produced very reliable domain classification.

In the case where data from one company was used, a comparison between CLM and the results of neural network training for origin year (y-direction) table 6.5 and table 6.18 shows that NN training has produced better forecasting results. Comparison between CLM table 6.5 and table 6.25 shows that linking has reduced the forecasting error even further. In the case where data from two companies was used, a comparison between the generalisation errors after training and after linking in table 6.49 shows only a slight improvement for Legal&General. Whereby the generalisation error of Co-operative has increased.

Table 6.56 summarises the changes in the generalisation errors after linking and after linking with stimuli network. The first example has caused a reduction of forecasting error in both instances for both domains. In the second example only domain B benefited. One explanation for this could be that the testing data was somewhat correlated to the training data and therefore an increase in the recall accuracy can cause an increase in the generalisation.

Table 6.56 Summary of generalisation error change for both examples.

	Linking without stimuli		**Linking without stimuli**	
	Domain A	**Domain B**	**Domain A**	**Domain B**
2-Directions	-14.24%	-13.79%	-42.53%	-21.17%
2-Companies	2.66%	-6.16%	34.82%	-34.63%

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis has introduced the first building blocks of a framework for a neural network linking process. Linking of knowledge contained in C-based source code and programming libraries has been so successful, that the application for linking of knowledge contained in neural network weight matrixes seemed overdue.

The linking process has been applied to vectors that represent knowledge held in each trained neuron thus utilising the existing and well-established field of vector algebra. Most beneficial has been the simplicity underlying the linking equations since only basic geometrical and vector algebra has been used.

The versatility of linking has been shown in the areas of pruning and in the combination of domain experts. Linking can be used for establishing the optimal size of a neural network by combination of similar neurons. Its purpose is to combine vectors that are holding similar knowledge and has been successfully applied to problems that could be described by a mathematical function and to real life data from the insurance industry. With this, knowledge of distinct domains has been linked for the benefit of improved generalisation.

7.2 Future Work

A number of possibilities for future work into linking have been described in each chapter of this thesis. These and other suggestions are listed in this section.

This section is a discussion of future work that is mainly relating to possible research into areas that represent a simplification or an enhancement. Simplifications have been made where it has been found to be important to continue the momentum in the research of linking without being sidetracked. Enhancements are suggested where a clear disadvantage exists in the methods of linking discussed in this thesis.

7.2.1 Simplifications

7.2.1.1 Combination of Output Layer Weights

The first simplification used in this thesis has been the setting of the output layer weights to 1 and freezing them so they do not change during network training. The requirement for this simplification originated from the need for non-linear activation functions that complicate the combination of the output layer weights. Therefore neural networks have been trained by updating their hidden layer weights but omitting the output layer weights. There are basically two possible solutions. The first one is a transformation of the output layer weights to 1 and the second one is the actual combination of the output layer weights. A preliminary transformation algorithm has been developed that successfully converted output layer weights to 1 by changing the weights of the hidden layer so that the network performance remained largely unchanged. Since this work has been mainly preliminary, it has not been included in this thesis.

7.2.1.2 Search for Similar Knowledge in Neurons

Another simplification that was used was the search for matching vectors that comply with the acceptance angle restriction and for linking. This search has been brute force where each vector has been compared with all other remaining vectors. Whilst this type of search is fine for small matrixes, the search time will increase exponentially if larger matrixes are used. This problem can easily be improved by using more sophisticated search algorithms

7.2.2 Improvements

7.2.2.1 Dynamic Pruning and Growing

Linking of neurons is a versatile process that combines neurons; only two possible scenarios, the pruning and combination of networks have been introduced in this thesis. Other areas such as dynamic pruning in combination with dynamic growing during training are possible future application areas for neuron linking. Dynamic pruning could combine neurons during the training that contain similar knowledge. In this instance, neurons are linked during network training and if found equivalent, linked so that one can be removed. Such dynamic pruning algorithms are generally used in conjunction with dynamic growing algorithms in cases where over-pruning occurred [118].

7.2.2.2 The Need for a Stimuli Network

The introduction of the stimuli network has been caused by the need to generate domain classification information. If two or more networks are linked that originate from the same domain, no stimuli network is required. In such a case, all networks involved in the linking will not need to store domain membership information. Because of this, the linking of networks from the same domain will be similar to the process that has been used for pruning but no such work has yet been undertaken.

7.2.2.3 Linking Multiple Networks

For the introduction of linking it was sufficient to demonstrate its paradigm with the use of two neural networks from two domains. But one of the real strengths of linking is that knowledge from multiple domains can be combined to form a complex structure. Linking of multiple domains will follow the same concept as demonstrated for two domains with the exception of the training of the stimuli network. The more domain memberships the stimuli network needs to distinguish, the lesser the accuracy with which each domain will be identified. Therefore the focus on linking of multiple domains may lie with the development of a more sophisticated stimuli network or the omission of the stimuli network as mentioned in 7.2.2.2.

7.2.2.4 Extension of Linking Equation

The linking equation has been derived in its basic form and proven to perform well for combining similar neurons. But the definition of similar neurons has been restricted to neurons that point in similar directions in hyperspace. This definition can be extended so that other neuron attributes are included such as the neuron sensitivity or relative error between weights. Such an extension to the definition of similar neurons can be included into the linking process by means of manipulation of the linking equation or insertion of an additional process.

7.2.2.5 Extension of Linking for Different Types of Networks

In this thesis, the linking process has concentrated only on MLP neural networks that were trained with the backpropagation algorithm. Naturally, linking is not restricted to MLP neural networks or backpropagation. For example, a very interesting subject the linking process could be applied to would be the reduction of dimensionality in a SOM network. Because the linking process is applied to vectors any expert system that uses vectors, globally referred to vector machines [194], can benefit from the linking process.

7.2.2.6 Extension of Linking for Different Types of Fields

Combination of similar vectors for the purpose of eliminating one in exchange for a single scalar factor is not restricted to the application of neural networks. Algorithms that reduce the amount of parameters required to store information are generally referred to as compression algorithms. Compression algorithms are a substantial part of communication since they reduce the required bandwidth for the transmission of the same information and therefore can increase transmission volume and speed. With this and the simplicity of the linking equation, it can be used to compress data prior to transmission for the purpose of reducing bandwidth.

Chapter 8

Bibliography

[1] H Schildt, "C the Complete Reference", second edition, Osborne McGraw-Hill (Textbook), 1990.

[2] P Perry, "Using Borland C++ 4", special edition, Que Corporation (Textbook), 1996.

[3] DJ Kruglinski, "Inside Visual C++", Microsoft Press (Textbook), 1993.

[4] A Ultsch, D Korus, "Integration of Neural Networks with Knowledge-Based Systems", University of Marburg, Artificial Intelligence in Medicine, Volume 934 of Lecture Notes on Artificial Intelligence, Springer Verlag, 1995.

[5] AH Tan, FL Lai, "Text Categorization, Supervised Learning, and Domain Knowledge Integration", KDD'2000 International Workshop on Text Mining, Boston, pp. 113-114, August 2000.

[6] AH Tan, "Integrating Rules and Neural Computation", RWCP Neuro ISS Laboratory, Proceedings ICNN'95, IEEE International Conference on Neural Networks, pp 1794-1799, 1995.

[7] CD Neagu, VP, "Modular Neuro-Fuzzy Networks Used in Explicit and Implicit Knowledge Integration", FLAIRS Conference, pp 277-281, 2002.

[8] GG Towell, JW Shavlik, "Refining Symbolic Knowledge using Neural Networks", Machine Learning, Volume 12, pp 321-331, 1994.

[9] S Garcez, K Broda, D M Gabbay, "Symbolic knowledge extraction from trained neural networks: A sound approach", Artificial Intelligence, pp 155-207, 2001.

[10] RS Michalski, S Chilausky, "Learning by being told and learning from examples", Journal of Policy Analysis and Information Systems 4, pp 126-161, 1980.

[11] TG Dietterich, RS Michalski, "A Comparative Review of Selected Methods for Learning from Examples", Machine Learning, Volume 1, pp 41-82, 1983.

[12] EB Messinger, LA Rowe, RR Henry, "A Divide-and-Conquer Algorithm for the Automatic Layout of Large Directed Graphs", IEEE Transaction on Systems, Man and Cybernetics, Volume 21, No. 1, pp 1-12, 1991.

[13] TG Dietterich, "The divide-and-conquer manifesto", Proceedings 11th International Conference Algorithmic Learning Theory, Springer Verlag, New York, pp 13-26, 2000.

[14] MI Jordan, RA Jacobs, "Hierarchical mixtures of experts and the EM algorithm", Neural Computing, Volume 6, Number 2, pp. 181-214, 1994.

[15] ME Ruiz, P Srinivasan, "Hierarchical neural networks for text categorization", Proceedings of the 22nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'99), pp 281-282, 1999.

[16] ME Ruiz, P Srinivasan, "Hierarchical text categorization using neural networks", Information Retrieval, pp 87-118, 2002.

[17] ME Ruiz, "Combining machine learning and hierarchical structures for text categorization", PhD Thesis, December 2001.

[18] E Black, J Lafferty, S Roukaos, "Development and evaluation of a broad-coverage probabilistic grammar of English-language computer manuals", Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics, pp 185-192, 1992.

[19] SK Riis, "Combining Neural Networks for Protein Secondary Structure Prediction", Technical University of Denmark, Proceedings ICNN'95, IEEE International Conference on Neural Networks, pp 1744-1749, 1995.

[20] C Boek, P Lajbcygier, M Palaniswami, A Flitman, "A Hybrid Neural Network Approach to the Pricing of Options", The University of Melbourne, Proceedings ICNN'95, IEEE International Conference on Neural Networks, pp 813-817, 1995.

[21] JA Benediktsson, J Larsen, JR Sveinsson, LK Hansen, "Optimized Combination, Regularization, and Pruning in Parallel Consensual Neural Networks", Proceedings of European Symposium on Remote Sensing, Volume 3500, Barcelona, Spain, pp. 3527-3538, 21-25, Sept. 1998.

[22] K Kirchhoff and J Bilmes, "Combination and joint training of acoustic classifiers for speech recognition", Proceedings of ISCA ASR2000 Tutorial and Research Workshop, Paris, 2000.

[23] J Waldemark, PO Dovner, J Karlsson, "Hybrid Neural Network Pattern Recognition System for Satellite Measurements", Proceedings ICNN'95, IEEE International Conference on Neural Networks, pp 195-199, 1995.

[24] G Auda, M Kamel, H Raafat, "Voting Schemes For Cooperative Neural Network Classifiers", Proceedings ICNN'95, IEEE International Conference on Neural Networks, pp 1240-1243, 1995.

[25] AJC Sharkey, NE Sharkey, "Combining Diverse Neural Nets", The Knowledge Engineering Review 12, pp 231-247, 1997.

[26] P Verlinde, "A Contribution to Multi-Modal Identity Verification Using Decision Fusion", PhD Thesis, Department of Signal and Image Processing, Telecom Paris, France, 1999.

[27] R King, M Ouali, AT Strong, "Is it Better to Combine Predictions?", Department of Computer Science, University of Wales, Protein Engineering 13, pp 15-19, 2000.

[28] R Maclin, JW Shavlik, "Combining the Predictions of Multiple Classifiers: Using Competitive Learning to Initialize Neural Networks", Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95), Montreal, Canada, pp.524-530. 1995.

[29] ZH Zhou, JX Wu, W Tang, "Combining Regression Estimators: GA-Based Selective Neural Network Ensemble", International Journal of Computational Intelligence and Applications, pp 341-356, 2001.

[30] C Perry, "An Integrated Developmental Connectionist Model of Orthography to Phonology and Orthography to Semantics Translation", Swinburne University of Technology Melbourne, Australia, Proceedings ICNN'95, IEEE International Conference on Neural Networks, pp 2960-2964, 1995.

[31] Y Satoshit, I Hidekiyot, N Yoshikazutt, "An Inverse Model Learning Algorithm Using the Hierarchical Mixtures of Experts", Proceedings ICNN'95, IEEE International Conference on Neural Networks, pp 2738-2742, 1995.

[32] B Sekerkiran, "A High Resolution CMOS Winner-Take-All Circuit", Proceedings ICNN'95, IEEE International Conference on Neural Networks, pp 2023-2026, 1995.

[33] BJ Jain, F Wysotzki, "Efficient pattern discrimination with inhibitory WTA nets", Proceedings in ICANN'01, Springer Verlag, pp 827-834, 2001.

[34] R Battiti, AM Colla, "Democracy in neural nets: Voting schemes for classification", Neural Networks 7, pp 691-707, 1994.

[35] ML Gargano, "Classifier voting in the neural networks." Proceedings in IJCNN 90, pp 388-391, 1990.

[36] BJ Jain, F Wysotzki, "Distance-based Classification of Structures within a Connectionist Framework", Forschungsbericht Nr. 763, Forschungsberichte der Universität Dortmund, FB Informatik, LLWA 01 - Tagungsband der GI-Workshopwoche, Dortmund, Oktober 2001.

[37] J Karhunen, L Wang, R Vigario, "Nonlinear PCA Type Approaches for Source Separation and Independent Component Analysis", Helsinki University of Technology, Proceedings ICNN'95, IEEE International Conference on Neural Networks, pp 995-1000, 1995.

[38] B Lerner, H Guterman, M Aladjem, I Dinstein, "Feature Extraction by Neural Network Nonlinear Mapping for Pattern Classification", The 13'th International Conference on Pattern Recognition, ICPR13, Vienna, Volume 4, pp 320-324, 1996.

[39] LR Tucker, RC MacCallum, "Factor Extraction by Matrix Factoring Techniques", Chapter 8, Exploratory Factor Analysis (Textbook), pp 180-215, 1997.

[40] A Hyvärinen, "Survey on Independent Component Analysis", Helsinki University of Technology Laboratory of Computer and Information Science, Neural Computing Surveys, Volume 2, pp 94-128, 1999.

[41] A Hyvärinen, E Oja, "Independent Component Analysis: Algorithms and Applications", Helsinki University of Technology, Neural Networks, pp 411-430, April 1999.

[42] WS Sarle, "Neural Network Implementation in SAS Software", Proceedings of the 19th Annual SAS Users Group International Conference, pp 1551-1573, 21 April 1994.

[43] SR Waterhouse, AJ Robinson, "Classification Using Hierarchical Mixtures of Experts", Cambridge University Engineering Department, IEEE Workshop on Neural Networks for Signal Processing IV, pp 177-186, 1994.

[44] W Jiang, MA Tanner, "On the Approximation Rate of Hierarchical Mixtures-of-Experts for Generalized Linear Models", Neural Computation, Volume 11 Number 5, pp 1183-1198, July 1999.

[45] EP Xing, MI Jordan, RM Karp, S Russell, "A hierarchical Bayesian Markovian model for motifs in biopolymer sequences", Advances in Neural Information Processing Systems (NIPS) 15, 2003.

[46] K Chen, L Xu, H Chi, "Improved learning algorithms for mixture of experts in multiclass classification", Neural Networks 12, Pergamon Publisher, pp 1229-1252, 1999.

[47] K Chen, D Xie, H Chi, "Speaker Identification Based on The Time-Delay Hierarchical Mixture of Experts", Peking University, Beijing 100871, China, Proceedings ICNN'95, IEEE International Conference on Neural Networks, pp 2062-2066, 1995.

[48] SR Waterhouse, "Classification and Regression using Mixtures of Experts", PhD Thesis, Department of Engineering, University of Cambridge, UK, 1997.

[49] J Fritsch, M Finke, A Waibel, "Adaptively Growing Hierarchical Mixtures of Experts", Carnegie Mellon University, Pittsburgh, pp 459-465, 1997.

[50] P Pucar, "Modelling and Segmentation using Multiple Models", PhD Thesis, Department of Electrical Engineering, Linköping University, Sweden, 1996.

[51] AV Rao, D Miller, K Rose, A Gersho, "Mixture of Experts Regression Modeling by Deterministic Annealing", IEEE Transactions On Signal Processing, Volume 45, Number 11, pp 2811-2820, November 1997.

[52] RA Jacobs, MI Jordan, SE Nowlan, GE Hinton, "Adaptive Mixture of Experts", Neural Computation, Volume 3, pp 79-87, 1991.

[53] JT Kwok, "Support Vector Mixture for Classification and Regression Problems", Proceedings of the International Conference on Pattern Recognition (ICPR), Brisbane, Australia, pp 255-258, August 1998.

[54] S Lawrence, AC Tsoi, AD Back, "Function Approximation with Neural Networks and Local Methods: Bias, Variance and Smoothness", Australian Conference on Neural Networks, ACNN 96, pp 16-21, 1996.

[55] P Smagt, F Groen, "Approximation with neural networks: Between local and global approximation", University of Amsterdam, Proceedings ICNN'95, IEEE International Conference on Neural Networks, pp 1060-1064, 1995.

[56] O Boz, "Knowledge Integration and Rule Extraction in Neural Networks", PhD Thesis, EECS Department, Lehigh University, US, 9 October 1995.

[57] YJ Moon, SY Oh, "On an Efficient Design Algorithm for Modular Neural Networks", Department of Electrical Engineering, Pohang University of Science and Technology, Korea, Proceedings ICNN'95, IEEE International Conference on Neural Networks, pp 1310-1315, 1995.

[58] WS Sohand, CK Tham, "Modular Neural Networks for Multi-service Connection Admission Control", Computer Networks, Volume 36, Number 2-3, pp 181-202, May 2001.

[59] M Fun, M Hagan, "Modular Neural Networks for Friction Modelling and Compensation", Proceedings of the IEEE International Conference on Control Applications, pp 814-819, 1996.

[60] AP Engelbrecht, "Sensitivity Analysis for Decision Boundaries", Neural Processing Letters, Volume 10, pp 253-266, 1999.

[61] AP Engelbrecht, HL Viktor, "Rule Improvement Through Decision Boundary Detection Using Sensitivity Analysis", Department of Computer Science, University of Pretoria, South Africa, pp78-84, 1999.

[62] AP Engelbrecht, "Sensitivity Analysis for Selective Learning by Feedforward Neural Networks", Fundamenta Informaticae XXI, IOS Press, pp 1001-1028, 2001.

[63] D Cubanski, D Cyganski, "Multivariate classification through adaptive Delaunay-based C^0 spline approximation", IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 17, pp 55-66, April 1995.

[64] TA Estlin, "Using Multi-Strategy Learning to Improve Planning Efficiency and Quality", Artificial Intelligence Laboratory, The University of Texas, TX 78712, PhD Thesis, May 1998.

[65] ZQ Liu, F Yan, "Case-based Diagnostic System Using Fuzzy Neural Network", Computer Vision and Machine Intelligence Lab, The University

of Melbourne, Australia, Proceedings ICNN'95, IEEE International Conference on Neural Networks, pp 3107-3112, 1995.

[66] J Utans, GR Gindi, "Neural network approach to object recognition and image partitioning within a resolution hierarchy", AAAI 1996 Workshops, Portland, Oregon, pp 238-246, 5 August 1996.

[67] DG Melvin, CT Spracklen, "Fusion of artificial neural networks and rule based systems for reasoning in noisy domains", Intelligent Engineering Systems Through Artificial Neural Networks Volume 2, ANNIE, ASME Press, New York, pp 963-968, November 1992.

[68] R Poli, M Brayshaw, "A Hybrid Trainable Rule-based System", Technical Report CSRP-95-3, School of Computer Science, The University of Birmingham, March 1995.

[69] P Bonaventura, M Gori, M Maggini, F Scarselli, "A Hybrid Model for the Prediction of the Linguistic Origin of Surnames", IEEE Transactions on Knowledge and Data Engineering, pp 760-763, 2003.

[70] DJ Dailey, P Harn, PJ Lin, "ITS Data Fusion", ITS Research Program, Research Project T9903, College of Engineering, University of Washington USA, April 1996.

[71] J Freeman-Hargis, "Rule-Based Systems and Identification Trees", Online Internet Tutorial, http://ai-depot.com/Tutorial /RuleBased.html, 2003,

[72] JP Ignizio, "An Introduction to Expert Systems: The Development and Implementation of Rule Based Expert Systems", McGraw Hill, (Textbook) 1 September 1990.

[73] S Haykin, "Neural Networks A Comprehensive Foundation", Prentice Hall, Upper Saddle River, New Jersey, (Textbook) 1999.

[74] R Callan, "The Essence of Neural Networks" Prentice Hall, Upper Saddle River, New Jersey, (Textbook) 1999.

[75] K Gurney, "An Introduction to Neural Networks", UCL Press Limited, (Textbook) 1999.

[76] A Blum, "Neural Networks in C++", John Wiley & Sons Inc, (Textbook) 1992.

[77] IW Sandberg, JT Lo, CL Francourt, JC Principe, "Nonlinear Dynamical Systems: Feedforward Neural Network Perspectives", John Wiley Publishers, 1 February 2001.

[78] L Fausett, "Fundamentals of Neural Networks: Architectures, Algorithms and Applications", pp 80-86, Prentice-Hall, 1994.

[79] J Hertz, A Krogh, R Palmer, "Introduction to the Theory of Neural Computation", Addison-Wesley Publishing Co, Redwood City CA, (Textbook) 1991.

[80] TR Shultz, WC Schmidt, "Modeling Cognitive Development with a Generative Connectionist Algorithm", Chapter 5 in Developing cognitive competence: New approaches to process modelling, Hillsdale, NJ Lawrence Erlbaum Associates, (Textbook) 1995.

[81] E Cox, "The Fuzzy Systems Handbook, A Practitioners Guide to Building, Using, and Maintaining Fuzzy Systems", Academic Press, (Textbook) 1994.

[82] L Zadeh, "Fuzzy Sets", Information and Control, Academic Press, Volume 8, New York, pp 338-353, (Textbook) 1965.

[83] T Terano, K Asai, M Sugeno, "Applied Fuzzy Systems", Academic Press Limited, (Textbook) 1997.

[84] J Sjöberg, "Non-Linear System Identification with Neural Networks", Linköping University, Printed in Sweden by Linköpings Trycker, (Textbook) 1995.

[85] JH Holland, "Adaptation in Natural and Artificial Systems", University of Michigan Press, Ann Arbor, Michigan USA, (Textbook) 1975.

[86] JH Holland, "Adaptation in Natural and Artificial Systems: An introductory analysis with applications to biology, control and artificial intelligence", MIT Press, Cambridge, MA, second edition, (Textbook) 1992.

[87] JH Holland, "Genetic algorithms", Scientific American, Volume 267, pp 44-50, July 1992.

[88] DE Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley, (Textbook) 1989.

[89] L Davis, S Coombs, "Genetic Algorithms and communication link speed design: theoretical considerations", Proceedings of the 2nd international conference on Genetic Algorithms, pp 252-256, 1987.

[90] L Davis, "Adapting operator probabilities in Genetic Algorithms", Proceedings of the 3rd international conference on Genetic Algorithms, pp 61-69, 1989.

[91] L Davis, "Bit climbing, representational bias and test suite design", Proceedings of the 4th International Conference on Genetic Algorithms, pp 18-23, 1991.

[92] L Davis, "Handbook of genetic algorithms", Van Nostrand Reinhold, New York, (Textbook) 1991.

[93] Z Michalewicz, "Genetic Algorithms + Data Structures = Evolution Programs", 2nd edition Springer - Verlag, (Textbook) 1994.

[94] M Kamber, R Shinghal, DL Collins, GS Francis, AC Evans, "Model based 3D segmentation of multiple sclerosis lesions in magnetic resonance brain images", IEEE Transactions on Medical Imaging, pp 442-453, 1995.

[95] YS Tsai, PH King, MS Higgins, "An Expert-Guided Decision Tree Construction Strategy: An Application in Knowledge Discovery with Medical Databases", American Medical Informatics Association Annual Fall Symposium AMIA'97, pp 208-212, 1997.

[96] AK Jerebko, JD Malley, M Franaszek, RM Summers, "Multiple Neural Network Classification Schemefor Detection of Colonic Polyps in CT Colonography Data Sets", published in PubMed, Acad Radio, Volume 10, Number 2, pp 154-60, February 2003.

[97] ZH Zhou, Y Jiang, "Medical Diagnosis with C4.5 Rule Preceded by Artificial Neural Network Ensemble", IEEE Transactions on Information Technology in Biomedicine, Volume 7, pp 37-42, 2003.

[98] K Racine, "Design and Evaluation of a Self Cleaning Agent for Maintaining Very Large Case Bases (VLCB)", Queen's University, Master Thesis, 1995.

[99] A Stranieri, J Yearwood, J Zeleznikow, "Tools for World Wide Web based legal decision support systems", Proceedings of the 8th International Conference on Artificial Intelligence and Law, pp 206-214, 2001.

[100] JR Quinlan, "Induction of decision trees", Machine Learning, pp 81-106, 1986.

[101] B Wilson, "Induction of Decision Trees", an Internet tutorial, location: http://www.cse.unsw.edu.au/~billw/cs9414/notes/ml/06prop/id3/id3.html, 2003.

[102] P Hancock, "Coding strategies for genetic algorithms and neural nets", University of Stirling, Centre for Cognitive and Computational Neuroscience Department of Computing Science and Mathematics, PhD Thesis, 1992.

[103] K Tumer, J Ghosh, "Analysis of decision boundaries in linearly combined neural classifier", Pattern Recognition, Volume 29, pp 341-348, 1996.

[104] G Rogova, "Combining the results of several neural network classifiers", Neural Networks, Volume 7, pp 777-781, 1994.

[105] S Hasham, B Schmeiser, "Improving model accuracy using optimal linear combination of trained neural networks", Technical Report SMS92-16, School of Industrial Engineering, Purdue University, 1992.

[106] D Opitz, R Maclin, "Popular ensemble methods: An empirical study", Journal of Artificial Intelligence Research, Volume 11, pp 169-198, 1999.

[107] NE Sharkey, AJC Sharkey, GO Chandroth, "Neural nets and diversity, Proceedings of the 14th International Conference on Computer Safety, Reliability and Security", pp 375-389, 1995.

[108] TK Ho, JJ Hull, SN Srihari, "Decision combination in multiple classifier systems", Conference PAMI'94 , Volume 16, pp 66-75, January 1994.

[109] E Alpaydin, "Multiple networks for function learning", Proceedings of the IEEE International Conference on Neural Networks, pp 9-14, 1993.

[110] S Hashem, B Schmeiser, Y Yih, "Optimal linear combinations of neural networks: An overview", Proceedings of the IEEE International Conference on Neural Networks, pp 1507-1512, 1994.

[111] RA Jacobs, "Methods for combining experts' probability assessments", Neural Computation, Volume 7, pp 867–888, 1995.

[112] C Bishop, "Neural Networks for Pattern Recognition", Clarendon Press, Oxford (Textbook), 1995.

[113] J Kittler, M Hatef, R Duin, J Matas, "On combining classifiers", IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 20, pp 226-239, March 1998.

[114] TA Bale, "Modular Connectionist Architectures and the Learning of Quantification Skills", Artificial Intelligence Group, University of Surrey, Guildford, PhD Thesis, September 1998.

[115] E Alpaydin, "Multiple networks for function learning", Conference IJCNN'93, pp 9-14, 1993.

[116] LK Hansen, P Salamon, "Neural network ensembles", IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 12, pp 993-1001, 1990.

[117] G Thimm, E Fiesler, "Neural Network Pruning and Pruning Parameters", Institut Dalle Molle d'Intelligence Artificielle Perceptive, Switzerland, 1st Online Workshop on Soft Computing, 19-30 August 1996.

[118] X Liang, "Network Expansion and Network Compression: Further Discussion on Structure Variation Methods", Peking University, Beijing, China, Proceedings ICNN'95, IEEE International Conference on Neural Networks, pp 680-685, 1995.

[119] AP Engelbrecht, I Cloete, "A Sensitivity Analysis Algorithm for Pruning Feedforward Neural Networks", Stellenbosch University, South Africa, International Conference on Neural Networks (ICNN '96). Sheraton Washington Hotel Washington, DC, USA, 2-6 June, pp 1001-1028, 1996.

[120] R Reed, "Pruning algorithms - a survey", IEEE Transactions on Neural Networks, Volume 4, Number 5, pp 740-747, 1993.

[121] C Bucila, J Gehrke, D Kifer, W White, "DualMiner: A Dual-Pruning Algorithm for Item sets with Constraints", 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Canada, pp 104-143, July 23 2002.

[122] P Ponnapalli, KC Ho, M Thomson, "Formal Selection and Pruning Algorithm for Feedforward Artificial Neural Network Optimization", IEEE Transactions on Neural Networks, pp 964-968, 1999.

[123] AL Prodromidis, SJ Stolfo, "Cost Complexity-based Pruning of Ensemble Classifiers", Knowledge and Information Systems, pp 449-469, 2001.

[124] DG Myers, "Psychology", Hope College Michigan, fourth edition, Worth Publishers, New York, (Textbook) 1999.

[125] "Claims Reserving Manual", The Faculty and Institute of Actuaries, London, Volume I+II, 1997.

[126] Bostock, Chandler, "Pure Mathematics", Stanely Thornes, England, (Textbook) 1996.

[127] T Harris, L Gamlyn, "Intelligent On-Line Multiple Sensor Diagnostics For Steam Turbines In Power Generation", Proceedings ICNN'95, IEEE International Conference on Neural Networks, pp 686-691, 1995.

[128] M Hattori, M Hagiwara, "Knowledge Processing System using Multidirectional Associative Memory", Faculty of Science and Technology, Keio University, Japan, Proceedings ICNN'95, IEEE International Conference on Neural Networks, pp 1304-1309, 1995.

[129] A Vahed, C Omlin, "Rule extraction from recurrent neural networks using a symbolic machine learning algorithm", Tech. Rep. US-CS-TR-4, Computer Science Department, University of Stellenbosch, Stellenbosch 7600, South Africa, pp 712-717, 1999.

[130] J Sietsma, R Dow, "Creating artificial neural networks that generalize", Neural Networks, Volume 4, pp 67-79, 1991.

[131] T Windeatt, G Ardeshir, "An Empirical Comparison of Pruning Methods for Ensemble Classifiers", Springer Verlag lecture notes, http://link.springer.de/link/service/series/0558/bibs/2189/21890208.htm, pp 208-217, 2001.

[132] TR Shultz, JL Elman, "Analyzing cross connected networks", Advances in Neural Information Processing Systems 6, NIPS'93, San Mateo, pp 1117-1124, 1994.

[133] M Riedmiller, "Advanced supervised learning in multilayer perceptrons - from backpropagation to adaptive learning algorithms", Computer Standards and Interfaces, Special Issue on Neural Networks, Volume 5, pp 265-278, 1994.

[134] B Liu, W Hsu, Y Ma, "Pruning and summarizing the discovered associations", Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining, San Diego, CA, pp 125-134, August 1999.

[135] AS Weigend, DE Rumelhart, BA Huberman, "Generalization by weight-elimination with application to forecasting", Advances in Neural Information Processing (NIPS'90), San Mateo, CA, pp 875-882, April 1991.

[136] J Furnkranz, G Widmer, "Incremental Reduced Error Pruning", Machine Learning: Proceedings of the 11th International Conference, pp70-77, 1994.

[137] G Castellano, AM Fanelli, M Pelillo, "An Iterative Pruning Algorithm for Feedforward Neural Networks", IEEE Transactions on Neural Networks, Volume 8, Number 3, pp 519-531, 1997.

[138] AP Engelbrecht, IC Loete, "Feature Extraction from Feedforward Neural Networks using Sensitivity Analysis", International Conference on Advances in Systems, Signals, Control and Computers, pp 221-225, 1998.

[139] AP Engelbrecht, I Cloete, JM Zurada, "Determining The Significance Of Input Parameters Using Sensitivity Analysis", International Workshop on Artificial Neural Networks, Spain, Springer-Verlag, pp 382-388, 1995.

[140] S Waugh, A Adams, "Pruning within Cascade-Correlation", Department of Computer Science, University of Tasmania, Proceedings ICNN'95, IEEE International Conference on Neural Networks, pp 1206-1210, 1995.

[141] F Castiglione, "Forecasting Price Increments Using an Artificial Neural Network", Advances in Complex Systems, Volume 4, Issue 1, pp 45-56 2001.

[142] Y Hirose, K Yamashita, S Hijiva. "Backpropagation algorithm which varies the number of hidden units", Neural Networks, Volume 4, pp 61-66, 1991.

[143] C Balkenius, "Generalization in Instrumental Learning", Proceedings of the 4th International Conference on Simulation of Adaptive Behavior (SAB96), pp 305-314, 9-13 September 1996.

[144] C Balkenius, J Morén, "Dynamics of a Classical Conditioning Model", ICANN'98, Perspectives in Neural Computing, Springer-Verlag, pp 41-56, 1998.

[145] CW Anderson, D Hittle, A Katz, R Kretchmar, "Synthesis of Reinforcement Learning: Neural Networks, and PI Control Applied to a Simulated Heating Coil", Journal of Artificial Intelligence in Engineering, Volume 11, pp 423-431, 1997.

[146] M Fenner, "The Nervous System, Pt IV Questions And Answers", MAF Newsletter, Volume 7, http://www.dinc.com/maf/1998/nl_05_07.htm, July 1998.

[147] D Haan, AW Young, F Newcombe, "Neuropsychological Impairment of Face Recognition Units", The Quarterly Journal of Experimental Psychology, Volume 44A, Number 1, pp 141-175, 1992.

[148] AK Warzecha, M Egelhaaf, "Neuronal Encoding of Visual Motion in Real-Time", Motion Vision - Computational, Neural, and Ecological Constraints, Springer Verlag, Berlin Heidelberg New York, pp 239-277, 2001.

[149] JM Zanker, J Zeil, "Motion Vision - Computational, Neural, and Ecological Constraints", Springer Verlag, Berlin Heidelberg (Textbook), New York 2001.

[150] N Kasabov, G Clarke, "A template-based implementation of connectionist knowledge based systems for classification and learning", Advances in Neural Networks, Volume 3, Ablex Publishing Company, pp 137-156, 1995.

[151] N Kasabov, M Fedrizzi, "Fuzzy neural networks and evolving connectionist systems for intelligent decision making", Proceedings of the 8th International Fuzzy Systems Association World Congress, Taiwan, pp 30-35, 17-20 August 1999.

[152] N Kasabov, "Decision support systems and expert systems", published in Handbook of brain study and neural networks, MIT Press, pp 21-26, 2002.

[153] TD Gedeon, "Data Mining Of Inputs: Analysing Magnitude And Functional Measures", International Journal of Neural Systems, Volume 8, pp 209-218, 1997.

[154] X Yao, "A new simulated annealing algorithm", International Journal of Computer Mathematics, Volume 56, pp 161-168, 1995.

[155] K Lagus, I Karanta, JY Jääski, "Paginating the generalized newspaper - a comparison of simulated annealing and a heuristic method", Parallel Problem Solving from Nature (PPSN IV), pp 594-603, Springer Verlag, Berlin-Heidelberg, 1996.

[156] RA Jacobs, "Methods for combining experts' probability assessments", Neural Computation, Volume 7, Number 5, pp 867-888, September 1995.

[157] K Kirchhoff, JA Bilmes, "Combination and joint training of acoustic classifiers for speech recognition", ISCA ITRW Workshop on Automatic Speech Recognition - Challenges for the new millennium (ASRU2000), pp 17-23, 2000.

[158] R Clemen, R Winkler, "Multiple Experts vs. Multiple Methods: Combining Correlation Assessments", Decision Analysis, November 1, 2002.

[159] M Su, M Basu, A Toure, "Multi-domain Gating Network for Classification of Cancer Cells Using Gene Expression Data", Proceedings of IJCNN 2002, May 12-17, pp 286-289, 2002.

[160] Y Won, PD Gader, "Morphological Shared-Weight Neural Network for Pattern Classification and Automatic Target Detection", University of Missouri - Columbia, Proceedings ICNN'95, IEEE International Conference on Neural Networks, pp 2134-2138, 1995.

[161] M Anthony, P Bartlett, Y Ishai, J Shawe-Taylor, "Valid generalisation from approximate interpolation", Combinatorics, Probability and Computing, Volume 5, pp 191-214, 1996.

[162] R Mahonyt, J Mooret, L Dailey, "Locally C1 Interpolation of Functions on an Arbitrary Simplex Mesh using a Simple Feed-forward Perceptron", Proceedings ICNN'95, IEEE International Conference on Neural Networks, pp 1662-1667, 1995.

[163] D Ensley, DE Nelson, "Extrapolation of Mackey-Glass data using cascade correlation", Simulation, Volume 58, Number 5, pp 333-339, May 1992.

[164] M Kayama, T Okamoto, "A navigation system based on self organizing feature map for exploratory learning in hyperspace", Transactions of the Institute of Electronics, Information and Communication Engineers, pp 561-568, 2000.

[165] M Kayama, T Okamoto, "A semantic map approach to a navigation system for exploratory learning in hyperspace", IEEE International Conference on Systems, Man, and Cybernetics (SMC'99), Volume 3, pp 839-844, 1999.

[166] DF Hougen, M Gini, J Slagle, "Partitioning Input Space for Reinforcement Learning for Control", Proceedings of the IEEE International Conference on Neural Networks, pp 755-760, June 1997.

[167] D Kidner, M Dorey, D Smith, "What's the point? Interpolation and extrapolation with a regular grid DEM", Proceedings of the 4th International Conference on GeoComputation, Mary Washington College Fredericksburg, Virginia, USA, pp 940-1008, 25-28 July 1999.

[168] A Browne, "Representation and Extrapolation in Multilayer Perceptrons", Neural Computation, Volume 14, Number 7, pp 1739-1754, 2002.

[169] E Barnard, L Wessels, "Extrapolation and Interpolation in Neural Network Classifiers", IEEE Control Systems Magazine, pp 50-53, October 1992.

[170] MA García, "Efficient Surface Reconstruction from Scattered Points through Geometric Data Fusion", IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, Las Vegas, USA, pp 559-566, 1994.

[171] J Marti, C Bunn, "Automated path planning for simulation", Proceedings of the Conference on AI, Simulation and Planning, AIS94, pp 122-128, 1994.

[172] R Sharda, D Steiger, "Using Artificial Intelligence to Enhance Model Analysis", The Impact of Emerging Technologies on Computer Science and Operations Research, Kluwer Academic Publishers, Norwell, MA, pp 263-279, 1995.

[173] A Bull, "Technical reserves in non-life insurance with particular reference to motor insurance", The Journal of the ASTIN and AFIR Section of the International Actuarial Association Volume V, Part 2, pp 177-198, May 1969.

[174] A Halpert, S Weinstein, C Gonwa, "Evaluating reserves in a changing claims environment", CAS Forum Fall 2001, pp.193-237, http://www.casact.org, 2001.

[175] T Mack, "A Simple Parametric Model for Rating Automobile Insurance of Estimating and IBNER Claims Reserves", The Journal of the ASTIN and AFIR Section of the International Actuarial Association, Volume 21, Number 1, pp 93-109, April 1991.

[176] "Introductory Statistics with Applications in General Insurance", London – New York, Cambridge University Press, (Textbook) 1983.

[177] T Mack, "Which Stochastic Model is Underlying the Chain Ladder Method?", XXIV ASTIN, Volume 15, No 2/3, pp 133-138, Colloquium in Cambridge 1993.

[178] BE Ollodart, "Loss Estimates Using S Curves: Environmental and Mass Tort Liabilities", Casualty Actuarial Society Forum, pp 111-132, Winter 1997.

[179] MB McKnight, "Reserving for Financial Guaranty Products", Casualty Actuarial Society Forum, pp 256-279, Fall 2001.

[180] RF Wiser, JE Cookley, A Gardner, "Loss Reserving", Foundations of Casualty Actuarial Science (4th Edition), Casualty Actuarial Society, Chapter 5, pp 197-285, (Textbook) 2001.

[181] E Pinto, DF Gogol, "An Analysis of Excess Loss Development", Proceedings Casualty Actuarial Society, Volume 74, pp 227-255, 1987.

[182] T Apostol, "Introduction to Analytic Number Theory", Undergraduate Texts in Mathematics, Springer-Verlag, New York, (Textbook) 1976.

[183] F Pierson, "Using the whole triangle to estimate loss reserves", CAS Forum 1994, http://www.casact.org, pp 11-44, 1994.

[184] DF Gogol "Using expected loss ratios in reserving", CAS Forum, pp 241-243, http://www.casact.org, Fall 1995.

[185] JP Evans, "Can Long Tailed Lines of Business Really Afford Higher Loss Ratios?", 2002 CAS Winter Forum, http://www.casact.org, Winter 2002.

[186] "UK Data Archive", Internet data resource, http://www.data-archive.ac.uk.

[187] RJ Verrall, Z Li, "Negative incremental claims: Chain ladder and linear models", Journal of the Institute of Actuaries, Volume 120, Actuarial Research Reports, pp 171-183, 1993.

[188] PD England, RJ Verrall, "Stochastic Claims Reserving in General Insurance", Presented to the Institute of Actuaries, pp 443-544, 28 January 2002.

[189] E De Alba, "Bayesian estimation of outstanding claim reserves", 6th International Congress on Insurance Mathematics and Economics hosted by CEMAPRE, ISEG, Lisbon, 2002.

[190] DR Clark, "Basics of Reinsurance Pricing", CAS Study Note and Tutorial, http://www.casact.org, 1996.

[191] S Goonatilake, P Treleaven, "Intelligent Systems for Finance and Business", University College London, John Wiley Publishers (Textbook), December 1995.

[192] J Yao, CL Tan, "A Study on Training Criteria for Financial Time Series Forecasting", Proceedings of International Conference on Neural Information Processing, Shanghai, China, pp 772-777, 14-18 November, 2001.

[193] J Yao, CL Tan, "Guidelines for Financial Forecasting with Neural Networks", Proceedings of International Conference on Neural Information Processing, Shanghai, China, pp 757-761, 14-18 November, 2001.

[194] N Cristianini, J Shawe-Taylor, "An Introduction to Support Vector Machines and other kernel-based learning methods", Cambridge University Press, (Textbook) 2000.

[195] D Grossman, T VanDeGrift, "Machine Learning Ensembles: An Empirical Study and Novel Approach", CSE 573 - Artificial Intelligence Course, University of Washington, Seattle, WA 98195, 2000.

[196] Z Zhi-Hua, W Jianxin,T Wei, "Ensembling neural networks: Many could be better than all", Artificial Intelligence Volume 137, pp 239-263, 2002.

[197] T S Rognvaldsson, "A Simple Method for Estimating the Weight Decay Parameter", http://citeseer.nj.nec.com/67062.html, OGI Technical Report CSE 96-003, 1996.

[198] A Krogh, J A Hertz "A simple weight decay can improve generalization", Advances in Neural Information Processing Systems, Volume 4, pp 950-957, 1992.

[199] J F Kolen, J B Pollack, "Back-Propagation Without Weight Transport", The Proceedings of the IEEE World Congress on Computational Intelligence, 26 June - 2 July, pp 345-351, 1994.

[200] L Breiman, "Bagging predictors", Machine Learning, Volume 24, pp 123-140, 1996.

[201] D Elizono, E Fiesler, "A Survey of Partially Connected Neural Networks", International Journal of Neural Systems, Volume 8, pp 535-558, December 1997

[202] M Rychetsky, S Ortmann, M Glesner, "Pruning and Regularization Techniques for Feed Forward Nets applied on a Real World Data Base", Symposium on Neural Computation, NC'98 Proceedings, Vienna, Austria, pp 824-829, 1998.

[203] L Orlandi, F Piazza, A Uncini, A Ascone, "Dynamic pruning in artificial neural networks", IV Italian Workshop on Parallel Architectures and Neural Networks, pp 199-208, May 1991.

[204] J Sietsma,R J F Dow, "Neural net pruning - why and how", IEEE International Conference on Neural Networks, San Diego, Volume 1, pp 325-333, 1988.

[205] M Mozer, P Smolensky, "Skeletonization: A technique for trimming the fat from a network via relevance assessment", Advances in Neural Information Processing Systems 1, Volume 1, pp 107-115, 1989.

[206] D Plaut, S Nowlan, G E Hinton, "Experiments on learning by backpropagation", Technical Report CMU-CS-86-126, Carnegie Mellon University, Pittsburg, PA, 1986.

[207] G E Hinton, "Connectionist learning procedures", Artificial Intelligence, Volume 40, pp 185-234, 1989.

[208] L Wu, J Moody, "A Smoothing Regularizer for Feedforward and Recurrent Neural Networks", Neural Computation Volume 8, Number 3, pp 463-491, 1996.

[209] H C Rae,P Sollich, A CCCoolen, "On-line learning with restricted training sets: An exactly solvable case", J.Phys.A Math.Gen.32 pp 3321-3339,1999.

[210] Y Le Cun, GE Hinton, "Improving the Convergence of Back Propagation Learning with Second Order Methods", Proceedings of the 1988 Connectionist Summer School, Los Angeles, pp 1608-1612, 1988.

[211] Y Chauvin, "Dynamic behavior of constrained back-propagation net-works", Advances in Neural Information Processing Systems (NIPS) 2, pp 642-649, 1990.

[212] Y Le Cun, J S Denker,S A Solla,. "Optimal Brain Damage", NIPS 2, pp 598-605, 1990.

[213] T Kavzoglu, C A Vieira, "An Analysis of Artificial Neural Network Pruning Algorithms in Relation to Land Cover Classification Accuracy", Proceedings of the Remote Sensing Society Student Conference, Oxford, UK, pp 53-58, 1998.

[214] B Hassibi, D G Stork, "Second Order Derivatives for Network Pruning: Optimal Brain Surgeon", Advances in Neural Information Processing Systems 5, pp 164-171, 1993.

[215] J Moody, "Economic Forecasting: Challenges and Neural Network Solutions", Paper presented at the International Symposium on Artificial Neural Networks, Hsinchu, Taiwan, December 1995.

[216] AP Engelbrecht,L Fletcher,I Cloete, "Variance Analysis of Sensitivity Information for Pruning Multilayer Feedforward Neural Networks", IEEE International Joint Conference on Neural Networks, Washington DC, USA, paper 379, 1999.