1995

Ensemble optimised prior knowledge transfer in neural networks

Edward S. Dunstone
University of Wollongong

ENSEMBLE OPTIMISED PRIOR KNOWLEDGE TRANSFER IN NEURAL NETWORKS

A thesis submitted in fulfilment of the
requirements for award of the degree of

Doctor of Philosophy

(Computer Science)

from

UNIVERSITY OF WOLLONGONG

by

Edward S. Dunstone BE. (Hons)

Department of Computer Science
September, 1995

I hereby declare that I am the sole author of this thesis. I also declare that the material presented within is my own work, except where duly acknowledged, and that I am not aware of any similar work either prior to this thesis or currently being pursued.

Edward S. Dunstone

Abstract

This thesis investigates methods by which the prior knowledge that is encoded in groups or ensembles of trained artificial neural networks can be used to assist in the learning of new tasks. Standard methods for training neural networks use only the observed (*a-posteriori*) data, ignoring any other prior (*a-priori*) knowledge which may be available for a particular task. One form of this prior knowledge is the representations that have been learnt by other networks trained on similar problems. The use of such knowledge can improve both training times and generalisation by appropriately biasing the representational ability of the neural network for the new learning task.

Previous research on the transfer of knowledge between neural networks has concentrated largely on its direct or literal transfer from a single source network to a single target network. It is shown in this thesis that the knowledge encoded by multiple neural networks trained within the same problem "environment" can be used in preference to single source transfer to improve the biasing of the search space. This is known as *ensemble transfer*.

This neural network prior knowledge can be likened to points on a map of the representation space. Each trained neural network in the prior knowledge defines the location of a neural network solution that is appropriate for the learner's environment. The goal of an *ensemble transfer* algorithm is to efficiently use the prior knowledge from this map to bias the learning of the internal representation for any new tasks. An appropriate form for the storage of the prior knowledge that allows reliable optimisation is thus essential. This aspect is considered in detail with reference to transformation invariance in the network representations and symmetric regions in the representation space.

An *ensemble optimised transfer algorithm* is developed based on a compact version of the solution space without network symmetries. It is then tested in three different problem domains: character recognition; spoken digit identification; and image approximation. Advantages both in training speed and stability are demonstrated in all of these situations over an algorithm which uses *literal transfer* from the lowest error network in the prior knowledge (*ensemble literal transfer*).

Acknowledgments

I would like to thank my supervisors, John Fulcher, Philip Ogunbona and Ming Zhang, for their support and guidance in shaping my thesis. In particular John and Philip for providing detailed feedback on my drafts and Ming for his patient guidance during the early stages of my thesis. I am also grateful to the assistance, support and friendship extended by other staff and postgraduates in the Computer Science and Electrical Engineering Departments. In particular James Andrew, Justin Crowley, David Ranson, Greg Doherty, Les Ohlbach and Jonathan Gray. Many thanks to Bill Clark from the department of mathematics at Rutgers University for some stimulating correspondence about the nature of symmetries in directed graphs.

The love and support of my family have kept me going through the inevitable rough patches during my research. I am indeed indebted to my parents for, amongst other things, imbuing me with enough perseverance and dedication to see this thesis to completion.

To all the friends I have met outside my research (especially past flat mates and Juggling Club members), my heart felt thanks for helping to keep me sane. I have had some of the best years of my life with your help.

My sincere appreciation goes to the Center for Information Technology Research (CITR) and the Illawarra Technology Corporation for providing me with an excellent research environment and ready access good facilities. Also to the former directors of CITR; Ian Renieke for initially setting up the CITR Neural Network Research Group and Hugh Bradow for allowing my continued presence at the center after the end of my scholarship.

I gratefully acknowledge the financial support of SITA (Societe Internationale de Telecommunications Aeronautique) through a grant to investigate face recognition, under the management of Beth Jackson.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The acquisition of knowledge and its subsequent representation are fundamental to the ability to perceive and interact intelligently with one's environment. This interaction is greatly facilitated by the ability to learn from past events and then to use this prior knowledge to assist in the learning of related tasks. The ability to transfer prior knowledge, which is so readily apparent in human behaviour, would have obvious benefits for artificial learning systems.

There is an increasing awareness in the connectionist field of the importance of prior knowledge for minimising some of the problems associated with classical machine learning, as well as practically improving both learning speed and generalisation ability. This thesis shows how the prior knowledge encoded in the learned representations of multiple neural networks can be optimised for transferral to previously unseen tasks.

1.1 Motivation

Neural network research has historically concentrated on the direct learning of representations from training data. Prior knowledge however can play an important role in the learning process, yet there are currently few methods for its effective use. The research in this thesis provides a general method for providing this form of knowledge explicitly to neural networks, using a detailed understanding of the topology of the network solution spaces. This understanding allows the derivation of algorithms that can optimise the transfer of knowledge from ensembles of pre-trained networks.

There are several practical consequences of the use of prior knowledge. Its effect on knowledge acquisition can be thought of as restricting, or biasing, the space of possible solutions to that subset of the solution space consistent with the known prior knowledge. The bounding of the solution space means that less training data is required to find a good solution as there is a reduced search area. Similarly, the training speed should often be improved because the appropriate solution will be significantly closer to the initial starting point, on average, than if no prior knowledge is available.

The restricted weight space also provides the ability to 'focus' the attention of the learning algorithm. The prior knowledge can be used to 'ignore' parts of the solution

space that are not consistent with the required solution, but which may be able to exactly fit the training data. Even at the most fundamental level the learning of a function f which is discretely sampled from the space of functions $\mathcal{F}$ can be seen to be ill-posed because there are an infinite number of solutions that will fit f exactly. To accurately choose a particular function requires some prior knowledge about the problem environment. In standard signal processing the assumed prior knowledge is usually that there is a smooth or Fourier interpolation between the samples.

Often poor training data also leads to erroneous results, where the actual global minimum on this data may not be the desired solution. Instead it is the best local minimum that is consistent with any available prior knowledge that is desired. As an example consider a classic US army target detection problem, where the goal was to identify whether a picture of a forest contained a tank. Training the neural network resulted in perfect performance on the training set. However on closer inspection of the training data it was noticed that all the images of forests without tanks were taken on cloudy days, and all those with on sunny days. The learning algorithm had found the best solution for this problem, given the training data; it could tell if the sky was clear or cloudy, rather than the intended goal of tank identification. The appropriate information was available in the training data, but the classifier did not have enough prior knowledge to bias its search in correct area of solution space, ie. to concentrate only on those parts of the image that contain information relevant to identifying a tank.

Another consequence of the 'focusing' ability is a reduction in the effect that noise in the training data has on the learned solution. This is achieved through exactly the same biasing of weight space suggested for the tank problem above. Because the solution space has been restricted, random noise has a reduced effect. This is equivalent to saying that the entropy of the learning system has been reduced because the interpretation of the training data is limited by the prior probabilities on the neural networks solution space.

1.2 Ensemble Transfer

The current research on the direct transferral of knowledge between neural networks deals largely with transferring the knowledge encoded in a single neural network to a new task. Consider two related tasks x and y. Given a network which has already been successfully trained on task x, then this network solution can often be used to assist in the training of a new network to solve task y. In it simplest form, the solution to task x is simply copied directly as the initial conditions for learning task y, this is known as *literal transfer*.

This thesis extends the class of transfer algorithms to deal with the transfer from multiple network solutions or *network ensembles*, where instead of a single task x, there

exists a set of prior tasks X. This set of tasks, and the resultant neural network solutions, can be used to construct a map of solution space from which the appropriate prior knowledge for any newly presented tasks can be inferred. This is known as *ensemble transfer*. The simplest ensemble transfer algorithm is *ensemble literal transfer* (ELT) where the best solution found in X for the new task y, is copied for use as the initial learning state.

For more advanced transfer algorithms, the ability to locate the appropriate prior knowledge requires a compact representation for the structure of the solution space. There are at least two complications to forming such a representation; similarity and symmetry.

It is desired that the solutions in X be stored in such a way that, those tasks which are related by a simple transformations of the problem domain, should have solutions that are close to one another in the solution space. Unfortunately however, the general identification of similar tasks is a difficult problem because a good measure of similarity for one task environment may be very poor for another. The set of linear or affine transformation are however generally regarded as being similarity preserving across many environments and it is shown that this type of transform can be compensated for in the solution spaces of trained feed forward neural networks.

The topology of the solution or weight spaces in most neural network architectures contain a large set of symmetric regions, where different solution vectors possess exactly the same input-output characteristics. Each of these regions contains exactly the same set of network solutions. Training algorithms take no account of these symmetric regions, thus a learnt solution so a task could lie in any region. The number of symmetric regions grows so quickly that the volume of non symmetric solution space actually approaches zero as the size of the network goes to infinity [CLH93]. This means that we cannot easily compare learnt solutions to different tasks. It is highly likely that two networks trained on exactly the same problem with different initial weights or learning algorithms will end up with completely different weight sets. This occurs even the solutions are globally minimal because there are as many global minima as symmetric solutions. If it is unknown whether weight vectors lie in the same symmetry region, the distance between the solutions, in general, conveys no information about the similarity between the encoded problem domains. Weight vectors in the same symmetry region however may be close both in terms of their proximity in weight space and the similarity in network response.

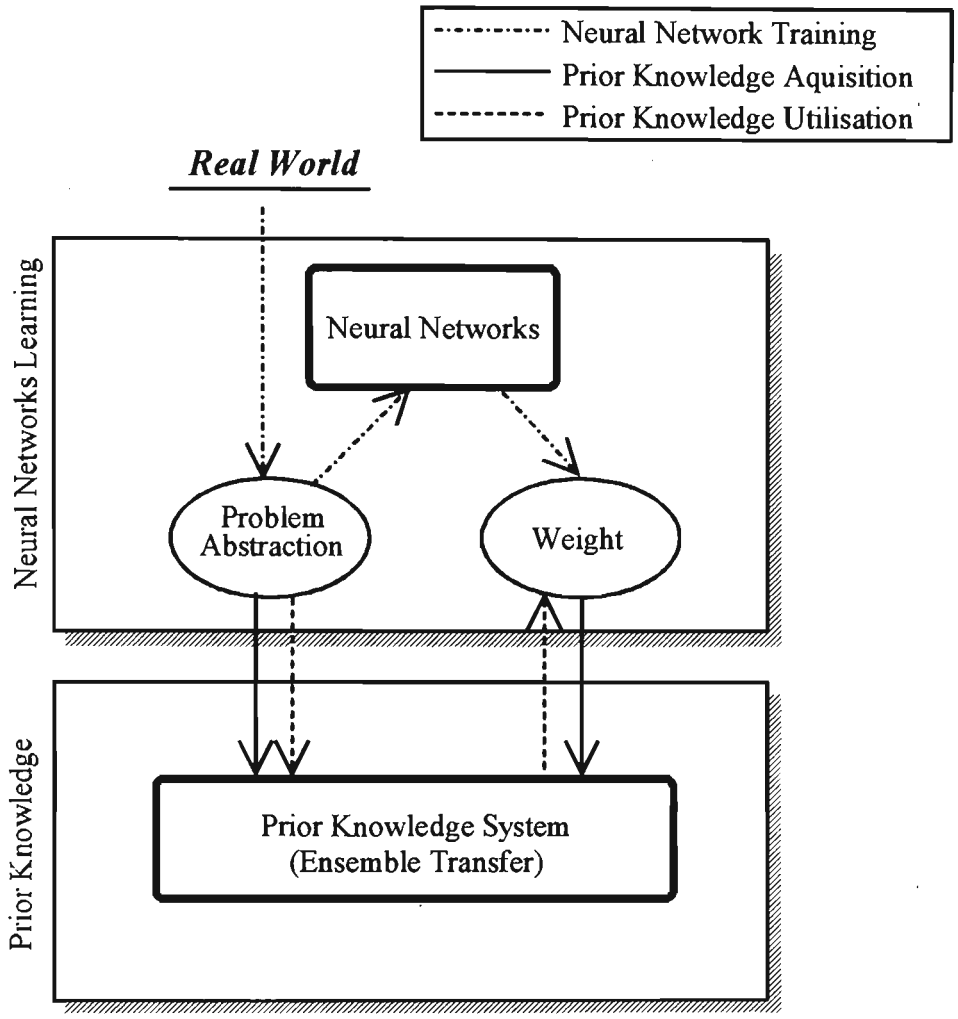


Figure 1.1: The use of prior knowledge via ensemble transfer

Transformation invariance and symmetric decomposition lead to a more compact representation of the volume of the neural network weight space. This form allows the mixing of the prior knowledge solutions for the calculation of an expected value for the weight vectors of a given training set. The calculated weight value is consistent with the available prior knowledge and can then used as the seed for further training. A diagrammatic representation of this type of system is provided in Figure 1.1.

1.3 Overview of the Thesis

There is emphasis in this thesis upon the internal representation that is acquired by a successfully trained neural network. The concepts derived from this examination are applied to the task of determining and using prior information. Little consideration is given to the actual learning algorithms used for training, because the overall effect of prior knowledge provides advantages which are independent of the learning algorithm

chosen. The magnitude of these effects will however be affected by the learning algorithm. Standard learning algorithms use the local topology of weight space to train, so any external information taking a more global view of the weight space topology, and thus restricting the solution domain, will be advantageous.

Chapter 2 starts by providing a general review of knowledge acquisition and representation in machine learning. Prior knowledge and its relationship to *a posteriori* knowledge is introduced, followed by a summary of previous attempts at the incorporation of prior knowledge in neural systems. Statistical inference, a general framework for the analysis of this knowledge in machine learning systems, is then discussed. This provides a basis for many of the terms used in this thesis. The more specific case of feed forward neural networks is then examined with regard to the density of solution in weight space. A simple set of one dimensional approximation examples is used to illustrate concepts relating to this density. The chapter is concluded by a brief overview of research into prior knowledge in the cognitive science and neurological disciplines.

In Chapter 3, the effects of similarity metrics in the problem domain on the weight space of neural networks are discussed. It is shown that affine transformations which take place in the problem domain can be compensated for in the network's weight space. Several relations are presented to achieve these transformations for two different forms of problem abstraction (classification and function approximation). These solution space transformations are used to provide a novel normalised form of weight space that is independent of applicable affine transformations. A new technique for image coding is also shown as part of this chapter. It provides a graphical demonstration of the effects of the affine transformations on the weight space.

A topological analysis of weight space is complicated by the presence of symmetric solutions, where the output response of the network can be the same whilst the internal weights are different. Chapter 4 introduces concepts relating to symmetry in the weight space and provides a detailed explanation of the effects on the topology of the solution space. A novel algorithm to decompose the symmetries based on the angles between weight vectors and simulated annealing, is also proposed. Some empirical studies are then undertaken to quantify the speed and efficiency of the given algorithm for practical problems. The effectiveness of this algorithm is examined empirically on two different problems.

After establishing the techniques for transform invariance (Chapter 3) and symmetry invariance (Chapter 4) in weight space, algorithms for the optimisation of this knowledge with ensembles of networks (*ensemble optimised transfer*) are developed in Chapter 5. A novel technique *network fusion*, which is a generalisation of two more specific algorithms, is demonstrated for mixing solutions on the neuronal level. To establish the

baseline optimisation parameters for these algorithms, they are tested using the same one dimensional approximation examples shown in Chapter 2. The performance considerations arising from the optimisation algorithm are then examined to allow a fair comparison with literal transfer.

Chapter 6 provides three example applications of the ensemble optimised transfer, character recognition, speaker recognition and image approximation. These examples are characterised using the techniques shown at the end of Chapter 5 and show significant performance improvements over ensemble literal transfer for prior knowledge sizes above an empirically determined bound.

The conclusion (Chapter 7) provides a perspective on avenues for future research and lists the most significant results from the thesis. The final section is devoted to potential applications of this research.

Appendixes A and B are devoted respectively to reviews on specific details of machine learning and network symmetries. The proofs for the theorems given in Chapters 3 and 4 are given in Appendix C. Appendix D contains pseudo code for the Ensemble Optimised Transfer, whilst Appendix E contains a diagrammatic representation of training data that is used in this thesis.

1.4 Glossary

Some of the main symbols used in this thesis are described below with the page number on which they are first described.

$\mathcal{H}$: The hypothesis space	14
h : an hypothesis	14
$\boldsymbol{X}$: Input Space	14
$\boldsymbol{Y}$: Output Space	14
$\boldsymbol{Z}$: Sample Space (or training data)	14
W : Weight space	17
$\mathbf{w}$: Weight vector	17
$\boldsymbol{X}_n$: Input to layer n in a network	17
L : Layer number	17
$\varphi(\mathbf{X}:\mathrm{w})$: Output response of network	17
k_{affine} : Function for affine transforms on the weight space	35
n_l : Number of neurons in layer l	50
N_s : Number of sign symmetries	50

Chapter 2

Knowledge Acquisition in Neural Networks

An initial promise of neural network techniques was to provide so called "black box" solutions to difficult problems. One need know nothing about the underlying model of the problem, since the network would be able to deduce this from the relevant training data, without outside assistance. This kind of process is called model-free inference or *tableau rasa* learning. For situations where it is too difficult or time consuming to derive an accurate mathematical representation for the physical model, such a system would be a panacea.

This aim however suffers from a major limitation known as the bias variance dilemma when it is applied to non trivial problems. In the neural network field this was first discussed comprehensively by Geman et al. [GBD92]. It is caused by the dual nature of the estimation of error on a problem. An incorrect model which has insufficient or inappropriate representational ability will have a high bias. On the other hand a model which is truly bias free must have a high variance because of its encoding flexibility, and hence will require a prohibitively large training set to provide a good approximation.

The dilemma is essentially that the more representational power a system is given, the more difficult it is to learn concepts correctly. Each system has an underlying process that is used to construct its internal model and as a consequence any solutions which are found will be naturally biased by the representational power of the learning system. Such bias includes the architecture type, connection topology, and perhaps most importantly the input and output representations. Consequently the estimation of these parameters relies on the prior knowledge or biases of the researcher about the problem, thus destroying the original goal of bias free learning.

To achieve low variance whilst simultaneously estimating a large number of parameters requires an impractical number of training examples. The solution to this problem lies in the proper use of prior information. As stated in [Bax95], *"it is often not the learning problem that poses the most difficulty but the identification of appropriate prior information"*.

2.1 Prior Knowledge

There are two types of information that can be used by a learning system to form a representation. *Posterior knowledge*, which is obtained after measurements have been made, and *prior knowledge*, which is knowledge obtained before measurements were made. Posterior knowledge is the kind of information usually associated with the training of neural networks, consisting only of directly observed training data. This information is usually explicitly available, and is easy to use. Prior knowledge is less frequently considered as a source of information for neural network training because the appropriate method of using such information is less obvious. The use of this information however can make a significant difference to the final learnt representation.

For example, consider a system that is required to distinguish class A from class B based on a single continuously varying parameter. Both sets posses identical Gaussian distributions but with different means. The variance of these distributions is such that there exists some degree of overlap. The optimal decision boundary that should be chosen by the classifier is the average of the distribution means. However if the training set contains a much larger proportion of A than B then, in the absence of any *a priori* information, the classifier will be incorrectly biased towards the class A. The theory of Bayes classifiers provides a more formal way of presenting this issue. Let the *a priori* probabilities of these two classes be P_A and P_B, and the *a posteriori* class conditional probability densities be p_A and p_B. For a sample x, the minimum error from Bayes theory is given by classifying it as class A if,

$$p_A(x) > p_B(x)\, P_B / P_A \tag{2.1}$$

If the *a priori* probability of a class is too low, then the optimal decision boundary may almost completely exclude that class.

The relationship between Bayes classification and neural networks is not directly obvious due to the non-linear nature of the activation functions in the neural network. It has been shown however that in the case where the networks output is constrained to [0..1] and given an appropriately sized network, an optimal Bayes classifier can be achieved at the limit of convergence [BB93].

2.2 Semantic vs Syntactic Knowledge

Prior knowledge can be further sub-divided into two classes; semantic and syntactic. *Syntactic knowledge* is information about the underlying physical or mathematical basis

of a problem. For most problems it is practically impossible to build a sufficiently accurate analytic model to represent the process. In some very simple problems however it is possible to learn such information. This information is known as determinations which are a general form of relevance knowledge, consisting of the information about the dependence among different features of the training set [MT94]. From current research it is not yet clear how a general procedure for automatically determining and acquiring this form of knowledge could be constructed.

Semantic knowledge is information about similar kinds of learning problems. It is this form of prior information which is most easily captured and utilised. In this thesis only this type of prior knowledge is considered in detail.

2.3 Techniques for Prior Knowledge Usage

For any learner with a fixed representational ability, learning can be viewed as a search through the range of implementable functions for the function that most closely approximates the desired problem. Methods of utilising the information contained in prior knowledge can thus be viewed as attempting to restrict or bias the space of implementable functions for a particular learner. This view of prior knowledge will be expanded in section 2.4.

Previous attempts at practical methods for the incorporation or transferral of prior knowledge in neural networks[1] can be divided into roughly three groups; *Weight techniques*, where the prior knowledge to be used is encoded in the weights of trained neural networks, *structural techniques* in which the prior knowledge is hard coded into the network architecture, and *learning techniques* which attempt to modify the way learning is conducted on the basis of prior knowledge.

One of the most serious problems in the comparison of transfer of training techniques is the way the performance is often measured relative to randomly initialised networks. This comparison is fraught with danger for all but the smallest networks. What sort of probability distribution should be used for the random initialisation of the weights, and how large a sample from the randomly initialised weight vectors is necessary in order to be guaranteed a statistically significant average training speed? These issues are not covered to any great extent in the literature on practical techniques for prior knowledge transfer. It is usually left to the reader to guess at the accuracy of any stated results, especially when the number of random trials is small. In this thesis all experiments involving a comparison have been done using a minimum of one hundred trials in an

[1] For conventional signal processing [Del89] provides a highly readable discussion of the use of a priori information, via set membership, for estimating the parameters in a linear system.

attempt to account for some of this variability, ultimately however one is bound by the huge computational requirements of training a network using real data over multiple runs. The issues raised by the measurement of performance of transfer algorithms will be considered further in Chapter 5.

2.3.1 Weight Techniques

Probably the simplest type of transfer is known as *literal transfer*, where the source weights are copied directly for use as the initial weights for training a new problem. Sharkey [SS93, JS95] has examined this situation for some simple classification problems and found that both *positive transfer* leading to decreased training times, and *negative transfer* with increased training times could occur over random initialisation, under certain circumstances. Negative transfer tended to occur when the type of output classification was changed between tasks, and positive transfer was more likely in the case where only the input classification was changed. All the networks considered in this thesis have either consistent inputs or outputs (eg for character recognition, a character will classified with the same output independent of the training set font). The consistent use of inputs and/or outputs is a necessary feature of training within the same problem environment.

Discriminality Based Transfer (DBT) [Pra93] uses the relevance of input layer hyperplanes (as determined by the weight values into the neurons) on the new task to determine whether the weights for each input layer neuron should be allowed to change. All other weights in higher layers are randomly initialised. The hyperplane relevance is determined by using an entropy measure or mutual information metric which relies on determining class boundaries in the training data. Enhanced training speed, relative to a small subset of randomly initialised networks, were shown across a range of real world classification tasks. DBT is constrained to systems in which a target class can be assigned for each input and so cannot be used for problems such as surface approximation or regression.

Some other weight based methods of utilising prior knowledge revolve around the use of traditional AI methods to generate appropriate weights [TM93]. These systems rely on the knowledge of a human expert to initialise weight vectors. In the same way the similarity between fuzzy systems and radial basis functions also allows the direct incorporation of fuzzy rules into neural type systems [GR94]. The direct incorporation of these "expert" rules into a neural network [MP95] is quite different from the transfer of prior knowledge amongst networks, so these techniques are not discussed further. Note that at the end of Chapter 4 some results are presented which may be helpful for AI systems (eg [BB90]) that are designed to explain the workings of networks.

2.3.2 Structural Techniques

A speculative paper by Schmidhuber deals with issues surrounding the embedding of 'meta'-levels in a neural network architecture [Sch93]. It is based on the idea that a network could examine and then change its own internal state by the use of appropriate feedback. No experimental results were given for this technique, however it is an interesting idea.

Brown et al. [BR92] demonstrates the use of a technique that constrains the weights within a hidden 'grey' layer according to the prior knowledge that is known about the desired function approximation. Good results are demonstrated for a single example where the non-linear dynamics of a control system are to be approximated. A method for constraining the weights on more general problems is not suggested within the paper.

Weight sharing, where several network connections are forced to share the same weight value, has been successfully applied to the problem of building symmetry invariance into networks. Improvements in both the generalisation ability and learning times for problems requiring such invariance have been found [Sha93].

It can be shown that the network structure and weights for approximating both linear and non-linear differential equations can be calculated by using a generalisation of the Taylor series [Coz95]. This technique allows the direct integration of any *a priori* knowledge about the differential equation to be factored into the network design. It is however limited to application domains that use differential equations, such as the prediction of time series.

Probably the most generally applicable of the structural techniques is problem decomposition. This is where a problem is broken down into smaller tasks and trained on separate smaller networks before being recombined into one large network. The methods given in the weight techniques can still be applied to these modular networks to yield even greater performance increases.

2.3.3 Learning Techniques

A 'meta' neural network (MNN) that learns to adjust the learning parameters by observing the changes in weights during training is shown in [NM92]. This technique is intended to allow the overall training speed to be increased on similar problems by getting the MNN to choose an optimum step size and direction vector for a gradient descent learning algorithm. On simple four bit parity and two class problems this technique was shown to have significant speed improvements, however no follow-up work has been published to date.

Abu-Mostaf [Abu93, Abu95] shows how hints can be integrated into the learning process by generating additional training examples from the prior knowledge. This can be an effective technique in environments where limited training data is available [AR91]. A variation on this method is used in section 3.6 to increase the diversity of the prior knowledge.

Baxter [Bax95] suggests that it is possible to learn a representation for the environment of a learner that is invariant to similarities in the environment. Essentially a single network is used to learn an approximate representation for the 'generic' functioning of the environment. To train a network for a specific problem from within this environment, the new network is placed 'on-top' of the environment network such that its inputs are connected to the environment networks outputs. These outputs are intended to be invariant under the similarity transforms applicable to a particular environment. The research is backed by some rigorous theoretical justification but is demonstrated only on toy problems.

2.4 Ensemble Transfer

The research in this thesis belongs firmly with the weight transfer techniques, however it is significantly different to any previous research in this field because it deals with the transfer of knowledge from more than a single source network. This paradigm will be called *ensemble transfer*.

One technique for achieving this ensemble transfer initially investigated, used the relationship between a low dimensional representation of the training set and the training weights to choose which literal transfer solution to use as the basis for further training [Dun94a] [DO95a]. This situation resulted in a peak speed-up of just over fifty percent compared to random initialisation over about 50 trials for image approximation, however the speed up was not very consistent between problems. It also proved difficult to generalise this technique to other application areas.

The techniques discussed in Section 2.3 are all of a practical nature, however there are also theoretical results that can be used. These allow the calculation of bounds on the amount of knowledge that is required for a certain level of confidence in decisions made from the knowledge space. One such technique that has been recently applied to neural networks is called probably approximately correct learning.

2.5 Probably Approximately Correct Learning

Probably Approximately Correct (PAC) learning theory is based on a mathematical framework constructed by Valient [Val84] using statistical inference theory. Its aim is to provide a solid mathematical foundation for research into theoretical issues in machine learning. The theory provides knowledge about the quantity of information required by a learner to form a good internal representation for a problem, independent of the computational complexity required to learn that representation. Most of the recent research in this area, particularly within the neural network field, owes its origins to work by Haussler [Hau92].

The most significant consequence arising from PAC theory is that the number of examples needed for robust learning increases with the complexity of task to be learnt. The more the learner knows about the problem domain, and consequently the more restricted the initial space of functions in which the target solution lies, the fewer possible solutions which need to be examined in order to learn it. Less examples are thus required to find an appropriate solution. In general, the number of examples needed to learn a given task increases in proportion to the number of potential solutions (Appendix A).

Many formal results have been published using PAC learning theory to provide bounds on the number of training examples required for a given level of generalisation in the presence of prior information (Appendix A.2). These bounds however are seldom of practical use because of the extreme generality of the theory. However despite this, or perhaps because of it, the bounds often provide a useful insight into the nature of both the information, and the model required for learning a good representation. At the core of machine learning theory is a field called statistical inference that provides definitions and relations between the available knowledge and the resultant inferences that can be drawn by a learner.

2.5.1 Introduction to Statistical Inference

This section provides a brief introduction to statistical inference to familiarise the reader with some of the terms used through out the thesis. A more comprehensive coverage may be found in Keifer [Kei87].

Consider a particular training set $\bar{z}$ that consists of a number of input samples x_i from the set X (the *input space)* and output samples y_i from set Y (the *output space*). The samples are chosen from a *sample space* Z, which consists of the mappings between X and Y, according to an unknown probability distribution P, which is fixed by the underlying problem domain. The task of a learner is to choose a hypotheses h which is in agreement with the training data from the set of all possible hypothesis $\mathcal{H}$ as constrained

by the representational ability of the learner. This is simply a restatement of the explanation given in Section 2.2 whereby a search is undertaken by the learner to find the best function to represent the training data. In neural network terms h represents the function that is computed by the weight vector **w** in a particular architecture. For a fixed network topology however h can be thought of as directly equivalent to the weight vector.

(2.2) $$h(\mathbf{X}) = \varphi(\mathbf{X},\mathbf{w})$$

Under classical training circumstances the empirical information is the only information the learner has about Z. On the training set with m samples, an empirical estimate of the hypothesis quality is usually measured using the mean square error,

(2.3) $$E(h',\vec{z}) = \frac{1}{m}\sum_{i=1}^{m}[h'(x_i) - h(x_i)]^2$$

Gradient Descent and other learning algorithms use this estimate as the basis for the generation of a hypothesis. This usually equates to the determination of appropriate weight values. This estimation on its own however is inadequate to determine a good approximation to Z unless the sample size is sufficiently large. Ideally we need to measure the *true error* which is :

(2.4) $$E(h',h) = \int_X [h'(x) - h(x)]^2 dP(x).$$

This equation requires knowledge about the probability distribution that the training samples are being drawn from. Unless we have a complete mathematical description for the model, this is impossible. If such a description did exist then teaching the system would be pointless as the known model could be used to make an exact calculations for the system.

Bootstrap techniques, where some of the training set is held in reserve for determining stoping conditions, breaks z into two disjoint sets. One set calculates the empirical error (eq 2.3) for the weight adjustments, and the other is used to obtain an estimate of the true error, so that training can be stoped when the true error is low. The estimate of true error should start to rise if the learner is generating an inappropriate representation, even though the empirical error may still be falling. In many cases this allows a better approximation to the true error, even though the full training set is not being used for the direct generation of the hypothesis.

The basis for the use of standard learning algorithms is the expectation that a low empirical error implies a small true error. This assertion is demonstrably incorrect for many problems given limited data. True PAC learning is thus provably impossible in a general learning situation without the use of prior knowledge [Hau92, Bax95]. This situation is a direct consequence of the bias variance dilemma discussed at the beginning of the Chapter.

2.6 Neural Network Parametric Model

The general machine learning theory discussed above is now restricted to feed-forward neural networks or multi-layer perceptrons (MLP's) and their associated hypothesis spaces. This structure has been partly chosen because of the extensive body of research results which exists regarding the theoretical issues surrounding it and also because of its widespread appeal in solving practical problems. With minor modifications it can be used to encompass a wide range of network paradigms including sparsely connected networks, radial basis function networks and recurrent networks. It does not however easily allow for network topologies with connections that skip layers such as cascade correlation [FL90], nor for inter-layer connections as in most self organising architectures [Koh89]. The flexibility MLP's for approximating arbitrary multi-variate functions is well documented [Che91], as are existence proofs showing that any well behaved continuous function can be approximated exactly given sufficient neurons and training time [CAS92, HKP91, Hec89].

2.6.1 Feed forward network architectures

Neural networks consist of sets of simple processing elements called neurons with weighted connections between them. A fully connected feed-forward neural network is one in which these neurons are assembled into layers, some of which are hidden and thus do not directly send or receive data from outside the network Each neuron in a hidden layer is completely connected to all the neurons in the layers above and below it. The following parametric description gives a set of definitions used in this thesis to describe the functioning of multilayer feed-forward neural networks. These definitions are based on notations used by Chen [Che91] and Hertz et al. [HKP91].

The multi-variate function $\varphi(x_1, x_2, \ldots x_n ; \mathbf{w})$ of dimension n which is computed by the neural network is produced by the interaction between the connection weights **w** and the non-linear activation functions at each hidden layer neuron. Consider a layered feed-forward network with L layers where connections go from the output of one layer to the

input of the next. A neuron j in layer l has a set of connections entering it from each neuron in the layer l-1 below and produces an output x_j^l , as part of the output for the entire layer X^l. The weights on these connections are called *fan-in* weights and those to the next layer *fan-out* weights. Let the fan-in weight from neuron i in layer l-1 to neuron j in layer l be given by w_{ji}^l. The set of weights in a layer then forms a weight matrix $W^l = [w_{ji}^l]$ of dimension $n_{l+1} \times n_l+1$. Each layer contains a bias node which provides a constant one output and receives no input, so the input to layer l is given by $X^l=(1, x_1^l, \ldots . x_{n_l}^l)$. The effect of the bias is to introduce a translation into the neuron activation function. The bias weight vector on layer l is given by $\left[w_{10}^l, w_{20}^l \ldots w_{n_l 0}^l\right]$ which will sometimes be denoted by the vector β.

Let N be the summation performed by each neuron before being passed through an activation function A then,

$$N_{(j)}^{l+1} = \sum_{i=1}^{n} w_{ji}^{l+1} x_i^l + w_{j0}^{l+1}, \qquad \text{where } l = 1,\ldots \text{ L-1} \tag{2.5}$$

$$\text{thus } N^l = W^l X^{l-1}.$$

For a particular network topology the *weight space* vector **w** is formed by the concatenation of the column spaces of the weight vectors for each layer. Thus,

$$\mathbf{w} = (\text{CS}(\text{W}^1)\ \text{CS}(\text{W}^2) \ldots \text{CS}(\text{W}^L)) \tag{2.6}$$

The output of each layer is given by $X^l = (1, x_1^l, \ldots . x_{n_l}^l)^{\text{T}}$ thus the input layer receives input of the form $X^0 = (1, x_1^0, \ldots . x_{n_l}^0)^{\text{T}}$ with the final output at layer L being $X^{L-1} = \varphi(X^0 ; \mathbf{w})$. The following recursive set of equations define the function computed by the network,

$$\begin{aligned} &X^0 = (1, x_1^0, \ldots . x_{n_l}^0)^{\text{T}} \\ &X^l = A_1(W^l X^{l-1}), \quad l = 1, \ldots \text{ L-1} \\ &X^{L-1} = \varphi(x_1, x_2, \ldots . x_n \ ; \mathbf{w}) \end{aligned} \tag{2.7}$$

Thus the final output is given as a series of cascaded affine transforms passed through some non-linear function A at each stage. Writing this in non recursive form illustrates this more clearly.

$$\varphi(X_o) = A_L(W^{L-1}A_{l-1}(\ldots.A_2(W^2A_1(W^1X^0)))..) \tag{2.8}$$

where X_0 = the input vector ; ie. $x_1, x_2 \ldots x_n$

2.7 Solution Volumes

The density of weight vectors that correspond to useable solutions in weight space, for a given problem environment, can usually be restricted by a priori knowledge about the computational ability of the network. If it is known, for instance, that very large weight values will saturate the networks activation functions, then the distribution of useful weights will usually be such that there are many more small weights than large ones. An activation function with finite asymptotic limits means that any increase in an individual weight value beyond a certain threshold (such that the activation function is always saturated) will have little or no effect on the computed function, so weight space can be seen to be effectively bounded. These effects will be examined empirically in the next section.

Consider a function $\rho_a(\mathbf{w})$ which gives the density of weight space due to the architectural considerations mentioned above. The volume of useable weight space may then be calculated as [HKP91],

$$V_0 = \int \rho_a(\mathbf{w}) d\mathbf{w} \tag{2.9}$$

This volume may be further restricted using results that will be presented in Chapters 3 (transform invariance) and 4 (symmetry decomposition). Removing symmetry from the weight space reduces the volume by the inverse of the number of symmetry regions N_s. The similarity transforms should increase the density of solutions by further restricting the volume, as certain classes of problem solutions are now represented in a more compact form. The new density of solutions $\rho_s(\mathbf{w})$ due to this compaction is dependent on the type of transformations which can be detected in the problem domain and subsequently compensated in the weight space. The weight space volume is hence,

$$V_0' = \frac{\int \rho_s(\mathbf{w}) d\mathbf{w}}{N_s}. \tag{2.10}$$

It is now possible to calculate the section of this space that provides good generalisation for a particular training set. Given a training environment with a target function f, an indicator function Ω can be specified that identifies what regions of the weight space will provide a generalisation such that the networks error is less than ε when compared to that function,

$$\Omega_f(\mathbf{w}) = \begin{cases} 1 & \text{if } \left|\varphi(\mathbf{X}_i; \mathbf{w}) - f(\mathbf{X})\right| < \varepsilon \\ 0 & \text{otherwise} \end{cases} \tag{2.11}$$

Then the volume of weight space containing solutions to f is,

$$V_0'(f) = \frac{\int \rho_s(\mathbf{w}) \Omega_f(\mathbf{w}) d\mathbf{w}}{N_s}. \tag{2.12}$$

A theoretical framework constructed by Schwartz et al [SSS+90] calculates the ratio between V_0 and $V_0(f)$ (without symmetry or similarity considerations) to determine the fraction of weight space that can implement f to the required error level. It then uses this result to determine the range of functions that are implementable for a particular architecture (*functional diversity*) from the entropy of this ratio. The theory goes on to show how the number of training examples p affects the generalisation ability. Essentially as more training examples are randomly chosen from the training set, with the underlying probability distribution given by f, then the volume of weight space $V_p(f)$ that is consistent with these training examples shrinks. At some number of training examples the network will always be able to generalise to a given error level on f. Thus the probability of randomly selecting a weight vector from weight space that is consistent with the p previous training vectors and receiving good generalisation can be determined. This then allows the determination of the average number of training vectors required for good generalisation.

2.7.1 Empirical Evidence for the Density function

To empirically examine the weight space density, an experiment was conducted to repeatedly train an ensemble of homogenous networks on a set of simple one

dimensional approximation problems. Each network had a single hidden layer with five neurons, a single input neuron and a single output neuron. The weight space thus contained sixteen dimensions. All neurons had unipolar sigmoid activation functions. The networks were trained for two thousand epochs, using the Levenberg-Marquardt learning rule [Hag93], on various exponentially modulated sine waves. One hundred training runs were performed per problem. The output for these networks can be seen in Appendix E.1. As the solutions found were all slightly different, the weight vectors reflect parts of solution space that encode functions within $V_0(f)$ to a given error level.

Taking a histogram of the best 50 weight vectors over all the problems yields the results shown in Figure 2.1. Other research into the general distribution of the weight values in weight space [Hin93,Hec89] has shown roughly Gaussian results, with a mean of zero. In contrast the observed data for this experiment is almost more like a symmetrical Cauchy distribution. Once the weights fall below a magnitude of 4, the frequency drops until it reaches a minimum for weight values of zero. This behaviour is explained by the fact that values which are very close to zero contribute less to functioning of the network than those that are slightly higher. A weight which is very close to zero effectively removes the connecting neuron in one layer from contributing to the relevant neuron in the layer above. In many learning situations this may be important in the calculation of the networks function, however it is reasonable to assume that these weights will be a minority when compared to the number mid-value weights. The two small peaks that occur at a weight magnitude of approximately fifteen are almost certainly artefacts of the type of training data, which had the same range of input [-5..5] and output data [0..1] for each problem.

This histogram gives us one view of the distribution of weights, but it is also possible to construct a more sophisticated visualisation of the weight space. Imagine a block of weight space with the origin at its center. In sections of the block where few useful solutions are found it is transparent, becoming denser the more solutions can be implemented per unit area. If a light is shone through this multi-dimensional block onto a wall, such that one of the blocks axes was parallel to the wall, the resulting shadow cast would be a 2D slice through the probability density function that was being calculated in equation 2.12. This slice of weight space does not reveal much about the total solution space. Let us now assume that the weight space block can be rotated about its axis, and that a strobe light is setup, so that when an axis of weight space lines up with the wall the light is turned on and the shadow is displayed. If the block is rotated quickly enough, a low dimensional projection or "shadow" of weight space is seen (Figure 2.2).

To construct this image every two element combination of the 16 dimensional weight vector ($C_2^{16} = 120$) was projected onto a 2D array. As a result the same weight is projected onto both axes under the different combinations and so the image is symmetric

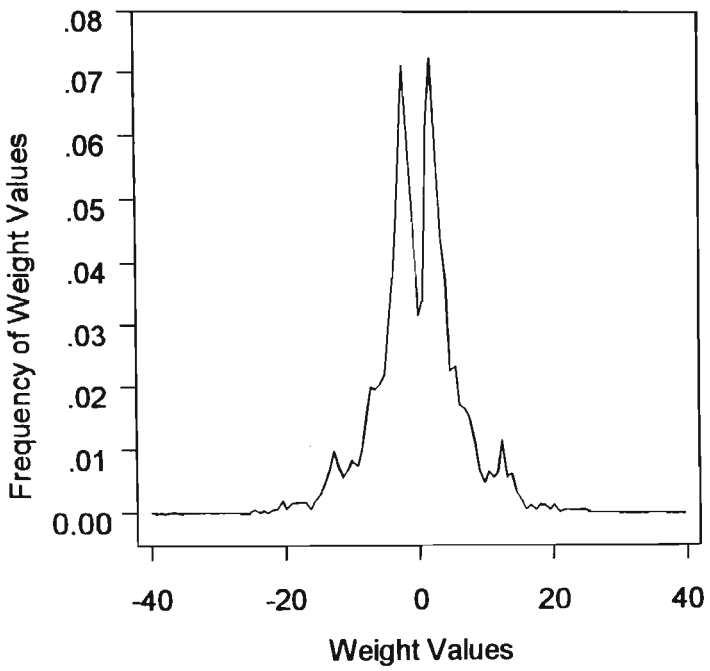


Figure 2.1 Histogram of weight values for 8 different tasks (Appendix E.1) each with 50 trained weight vectors.

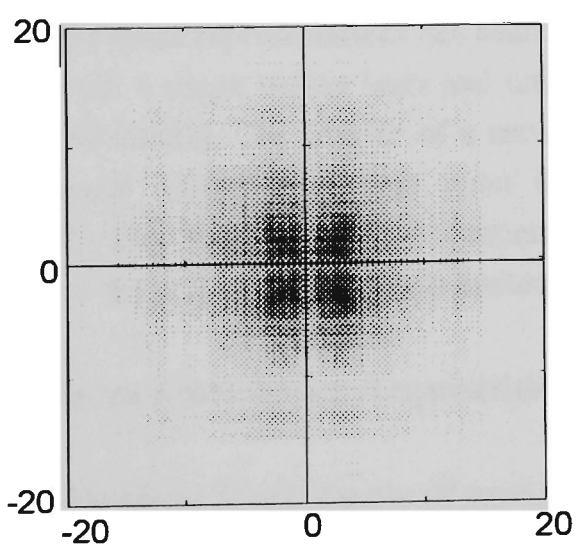


Figure 2.2 : Weight space shadow showing a two dimensional projection of the distribution of solutions in weight space.

about its positive diagonal axis. In keeping with the original histogram results most solution weights lie in a relatively compact region around the origin, with the density of solutions decreasing very near the weight axes.

The specific problems that were used to form these general shadows are shown in Appendix E.1. Each graph is the resultant output of the top 50 network solutions, accompanied by the corresponding shadow of the weight space. These examples will be used further in Chapter 4 and 5.

2.7.2 Weight Space Connectivity

To produce a map of weight space it would be convenient to have a 'canonical' representation for weight vectors. Unfortunately a true canonical representation of weight space is not possible, as this would imply that there was a one to one correspondence been each weight vector and a training vector. It should be clear from the formulas shown in section 2.7 that for a finite number of training examples there may be an entire volume of weight solutions that generalise well on a problem. The best that can be expected in practice is that $V'_0(f)$ should, for any f, exist on a compact connected subregion of the total weight space. It is desired that a particular type of problem be located in only one part of weight space, ie. no discrete symmetries, and that the volume of this weight space section can be localised. Whilst such a property is not proved for the compact representation proposed by equation 2.12, the empirical results on the mixing of solution vectors that will be shown in Chapter 5 provide evidence that this may be the case, at least for some practical situations.

The uniqueness of weight space representations has been demonstrated theoretically for feed-forward networks with a single hidden layer and tanh transfer functions, when symmetries are ignored [AS92,Sus92]. The weights of a network under these condition have been proven to uniquely determine a given input output map, however the assumption is made that the this mapping is both continuous and unbounded. The practical consequence of such theoretical results thus is limited.

2.8 Other Perspective's on Knowledge Acquisition

One of the motivations for research into the use of prior information is the empirical observations which have been made about the learning process in humans within the fields of psychology and cognitive science. It seems obvious that the use of prior information affects the way we learn. Each time a new word is heard or a new face is seen the brain draws on its prior knowledge about the world to classify and categorise these items quickly and efficiently. An analysis of the role of prior knowledge in human

learning can perhaps provide clues as to efficient mechanisms for its incorporation in machine learning systems and at the very least help to provide a context against which such systems may be judged.

2.8.1 Motivations from Human Learning

From a cognitive science perspective, the role of prior knowledge in skill acquisition is to enable the learner to detect and correct errors [Ohl93]. The prior knowledge is thus used to influence the framework under which new skills are acquired. This creates an implicit coupling between learning and knowledge. Knowledge is acquired by learning but, in the process of learning, the acquisition of future knowledge is affected.

Prior knowledge impacts on this acquisition by allowing the detection of errors through "contradictions and conflicts" between the prior knowledge and the knowledge to be learned [Ohl93]. The more knowledge the learner has the more apparent these discrepancies become. The prior knowledge can then be used to identify the cause of the error.

When given completely unfamiliar problems to solve, people employ very general problem solving techniques that can be used with minimal information about the task environment. These techniques are called *weak methods* and include such skills as hill climbing and successive approximation some of which appear analogous to simplified artificial neural network training. These methods are general but inefficient and aim to provide some idea about the structure of the task environment, rather than giving complete or correct problem solutions. It has been suggested that all such weak methods derive from a single universal weak method [Lai86].

The function of weak methods is to provide learning opportunities. Skill acquisition begins with these general weak methods and finishes with domain-specific skills. This runs counter to the common observation that knowledge begins with specific examples and becomes more abstract with further experience. This finding is consistent however with stochastic machine learning where the learning is dependent on a temperature parameter. At the start of training, when the temperature is high, the learner does not attempt to select a particular part of the hypothesis space, rather a large proportion of this space is examined to find out which provides the best general fit. As training proceeds and the temperature is cooled more attention is paid to the specific training examples in order to find an optimal solution.

Another unexpected result in human learning [Ohl93] is that the rate of skill acquisition is faster the less the learner knows about the task. In machine learning terms this negative learning curve is equivalent to saying that the more prior knowledge is obtained the smaller the advantage, from a learning perspective, that is gained by

knowing this new prior knowledge. Interestingly these observations parallel some of the results found in Chapter 5.

Murre [Mur95] conducted a study into the plausibility of a variant of back-propagation as a model for human long term memory. The study demonstrated that a variant of radial basis function networks called ALCOVE [Kru92] could provide a good model of the transfer and interference results obtained from human subjects when learning new tasks. He also demonstrated that under some circumstances back-propagation also shows *hyper-transfer*, (which is not seen in humans) where the additional learning of an interfering task may actually improve the performance on the original task.

2.8.2 Neuro-physiological Basis for Prior Knowledge

Data on the biological storage and retrieval mechanisms for prior knowledge is still largely unknown. Thus there is intensive research that has been undertaken in various areas of the cerebral cortex, particularly the C3 region of the Hippocampus [LSS94], which has lead to some very interesting results on the retrieval and reinforcement of learned knowledge (see especially chapter one of [ON78]). While it is too early to determine precisely the role of prior information in these structures, small scale simulations currently are being constructed [BCS94] and may yield valuable results for the practical construction of machine learning systems in the future.

There is much debate currently as to the role of a-priori knowledge in neuro-physiological structures, with regards to the extent that the brain is genetically programmed versus environmentally conditioned [Gre95, OB94]. The visual cortex is a particularly active area of research in this regard, as this is one of the easiest system in which to study adaptation to new environments [Rol95].

2.9 Summary

This chapter has examined both theoretical issues in, and practical implementations for, the acquisition of knowledge in machine learning systems. The bias variance dilemma and Bayes theorem were used to highlight the importance of prior knowledge to overcoming theoretical limitations for machine learning. The definition of prior knowledge was then refined by explaining the distinction between semantic and syntactic prior knowledge.

Various practical techniques for using the prior knowledge were introduced. These were placed three broad categories; weight, structural and learning. Ensemble transfer was explained in the context of these techniques.

A mathematical basis for the link between the available knowledge and possible hypothesises was then shown though a brief introduction to PAC learning theory and statistical inference. Following this the specific functional operation of a feed-forward neural network was examined.

The concept of weight space density in neural networks was then linked to the network solution weight vectors. From this the volume of useable weight space was defined by the integral of this density over the weight space. Further compaction of the weight space is suggested through the removal of symmetries and the use of transformation invariance. An experiment to show the existence of the weight space density function was then conducted, and some inferences made about the nature of weight space. Weight 'shadows' were introduced to assist in the visualisation of this weight space density function.

Chapter 3

Transformation Invariant Weight Space

The transfer of prior knowledge requires that we have some measure to define the similarity between different tasks. The use of prior knowledge is thus intimately tied to the notion of similarity because it is this measure that determines what knowledge is relevant for a new problem. A general definition of this similarity metric is non-trivial, as a measure which is good for some problems may be very poor for others. For example the use of peak signal to noise ratio, whilst excellent for many simple signal processing problems, is a poor approximation to visual similarity in images [Jai81,PL94]. For ill-defined problems such as vision and speech such similarity metrics are extremely difficult to derive - for example witness the huge amount of research effort which has gone into speaker independent recognition, hand-written character recognition and face recognition. These areas are still undergoing intensive research and it seems likely that a practical solution for one problem domain will yield little information which is generally applicable to another. In other words, the environment in which a task exists defines the type of similarity function which is appropriate. The environment essentially defines a set of transformations which can operate on a signal and leave the similarity metric relatively unchanged.

The issues arising from this are covered at some length in Baxter [Bax95]. He maintains that the encoding of similarity functions can be achieved by learning a model for the environment in which the prior knowledge is to be used. Similarity is thus determined though the way the metric behaves when signals are "close". This is termed *representation learning*. In this paradigm a neural network is trained to learn a representation of the problem domain such that the outputs of this network are invariant under the similarity transformations relevant for that environment. New problems in the same task domain can then be solved by training a neural network from the outputs of the representation network. Baxter provides rigorous theoretical justifications for the advantage to be gained on learning bounds by using prior information in this way. In doing so a stronger generalisation criteria is introduced where the learner must not only

perform well on the entire problem domain but also have discovered how to learn from the environment in general.

There is an assumption however that the representation network is able to encode a suitably good model for the environment in which it operates. It is up to the user to determine the architecture which the representation network should have for a particular environment - an obviously non-trivial task for complicated problems, especially when determining the appropriate number of outputs required for similarity invariance in an environment. For the toy problems presented this was not so difficult, but it doubtful how well this technique would scale up for more difficult real world problems such as image and speech recognition.

Such environmental representations can be empirically observed to exist in human brains, and there is much physiological evidence to suggest that this representation is easily learnable, as humans appear to be able to acquire it from a relatively small number of examples [Lor91, Hal93]. However, given the current limited state of knowledge about the inner workings of the brain, this deduction is of little practical consequence for the construction of such systems in the near future.

3.1 Weight Space Similarity Transforms

The research reported in this thesis is quite different from that described by Baxter, as it discusses weight space *explicitly* rather than attempting to learn to encode environmental representations. Because of the difficulties in obtaining a good representation for the environments of real problems, the emphasis in this chapter is on a class of similarity preserving transforms which appear to be relatively general. These are linear or *affine* transformations. A word repeated at different times or at different speeds will usually be recognised as similar, as will an image which has been simply translated or scaled.

3.1.1 Environmental Consistency

An environment in this thesis is defined by the consistent use of either inputs or outputs. Consider a situation in which the task environment is to recognise a particular instance of some general recognition class. The environment must be defined such that the output space is consistent across all the instances. For example, if the environment is spoken digit identification, then the instance class may be the voice of a particular individual. In this case the network output corresponding to the word 'seven' should be the same in all training sets. In an approximation environment it is usually the input, rather than the output, space that we wish to be consistent across training sets.

Environment	Instance	Input Space	Output Space
Digit Identification (Classification)	A persons voice	*sampled sound of voice*	The digit spoken
Character Recognition (Classification)	A Font Type	*image of a character*	The character 'A'-'Z'
Surface Learning (Approximation)	An Image	Pixel Coordinates	*The intensity of the image at grid points*
Process Control (Approximation)	A particular system type	Inputs from system	*Outputs to control system*

Table 3.1 : Example Classification and Approximation Environments

Table 3.1. shows some example environments and related parameters. The underlined text indicates training data where it is desirable that consistency in representation is maintained. The italic text shows the space in which the problem specific training data is located. The problem space is then best described by that space in which no consistency is required.

For classification environments where there are multiple instances, a clear delineation of the problem specific training data is not as easy. For instance take a face recognition system where the network provides an output indicating what individuals face is present at its input. For this problem domain neither the input space nor the output space is likely to be consistent across different instances (unless the same people are used in the different training sets). It would be possible however to force some consistency by attempting to match similar face types to particular outputs.

This leads to the idea of regularity in an environment. The more regularity that is in an environment the less structural variation there is between different task instances. Thus, the larger environmental regularity the easier it will be to transfer knowledge between tasks. For instance, in printed character recognition, the variation between the characters in different fonts is relatively small. Characters are inherently two dimensional and binary. On the other hand, a system designed to recognise faces has a more diverse set of possible representations to cope with, and hence contains less regularity.

The reason for defining the environmental consistency requirements is twofold. Firstly the transfer of training between problem domains makes far more sense in an environment that is held to these consistent constraints. Secondly a knowledge of the

space in which problem specific training data is found is necessary for the efficient detection of transforms in the problem domain.

3.2 Transformation Invariance

To use prior knowledge by the transfer of training, it would be advantageous to be able to store the solution vectors in a state that is independent of any similarity preserving transformations.

Thus the solution to a single training vector is extended to be meaningful over the domain of problems that are covered by the similarity preserving transforms in that environment. Ideally a solution to one problem can then be directly used for any similar tasks without further training. A set of algorithms is thus required that can detect similarity preserving transforms f that occur in a particular training set z so that a reverse transform can be applied to the neural network hypothesis space $\mathcal{H}$. When a new task is presented, any transformations that can be detected are used to deform the stored solutions into an appropriate normalised state.

Consider two tasks, A and B, and their associated training sets z_A and z_B which are related by a similarity preserving transform f such that,

$$z_A \approx f(z_B).$$

To use the prior knowledge contained in a solution to task A to assist the training of task B, a method of performing the inverse transformation in the weight space is required. The two tasks are thus linked by paired transforms in the problem domain and in the weight or hypothesis space,

(3.1)

$$\begin{array}{ccc} z_A & \underset{f}{\overset{f^{-1}}{\rightleftharpoons}} & z_B \\ \downarrow & & \downarrow \\ h_A & \underset{k}{\overset{k^{-1}}{\rightleftharpoons}} & h_B \end{array}$$

For instance examine the simple classification problem shown in Figure 3.1. The lines through the image indicate the decision boundaries as may be determined by a small neural network trained on task A. It can be seen that the trained vector for task A is of little direct use to task B as it stands. In fact an initial weight configuration of this sort for task A would probably be detrimental to the learning processes [Pra93]. Task B is however *similar* to task A, only translated. Exactly the same decision boundaries can be used for task B by appropriately modifying the weights. It is simply necessary to determine, from some features of the training data, the magnitude of this translation.

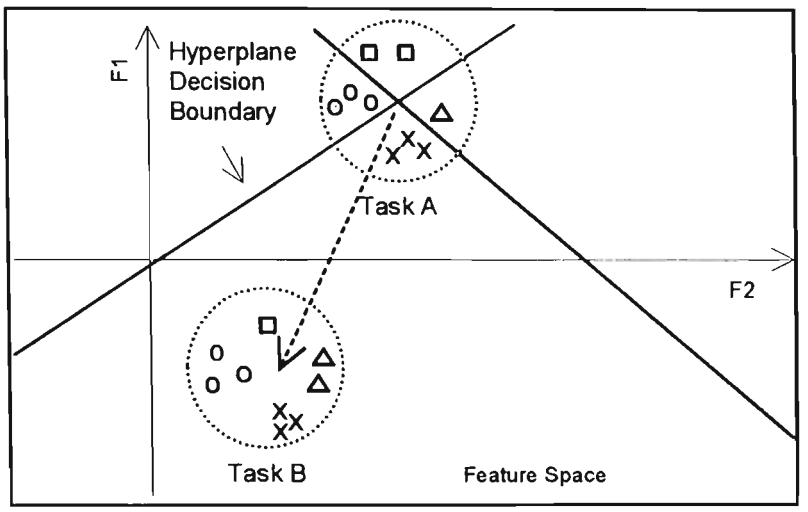


Figure 3.1 : Transferring weights from Task A to Task B.

For this problem an averaging of the feature vectors, to find the centers of the task clusters, would provide a good indication of the translation between the two tasks in the problem domain. This can be used to determine the required transformation to the network weights for a translation of the decision boundary. The prior knowledge which is contained in the solution to Task A has then been transformed to be directly relevant to Task B.

This situation can be generalised by requiring all solutions to be stored in a normalised form. New solutions which are to be used as prior knowledge must then be transformed such that they are consistent with the state of this normalised form. This is termed *Transformation Invariant Form*. In the above example the transform invariant form is such that the mean feature vector lies at zero. When a training set for the new task B is presented, its mean must be calculated and the weight vectors in the prior knowledge appropriately transformed so as to be consistent with the new problem environment. Thus the best match with the transformed weight vectors (ie. Task A) can then be used as the basis for further optimisation (Figure 3.2).

In summary, if $f_A(z_A) \approx f_B(z_B)$ then the correct prior information g_B for task B is,

$$g_B = k_B^{-1}\left(k_A\left(\hbar_A\right)\right) \tag{3.2}$$

Figure 3.3 graphically illustrates the meaning of equation 3.2 and shows the storage of the final solution $\hbar_B$ back onto the solution space map.

3.2.1 Solution Space Maps

The aim of compensating for the similarity transforms is to allow a single network solution to be relevant over an entire domain of transformed problem domains. For a

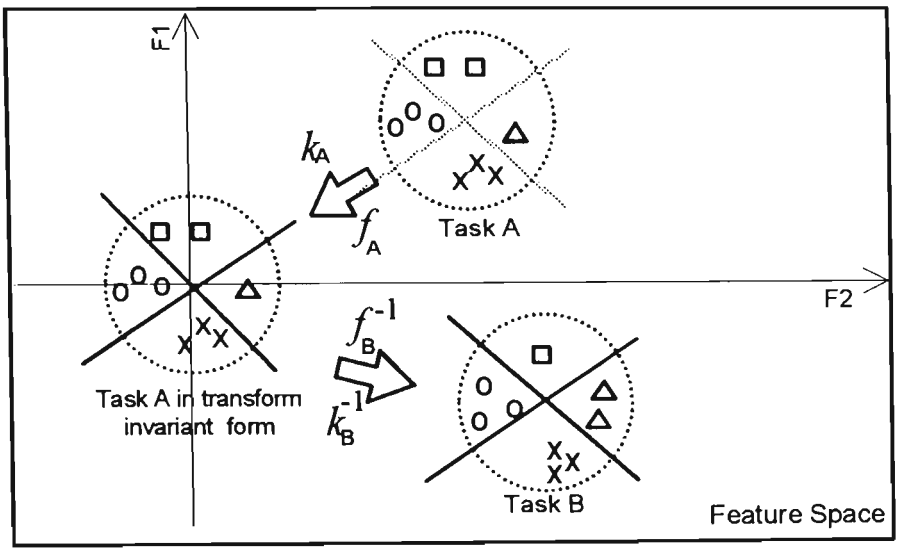


Figure 3.2 : Demonstration of normalised form for the simple example given in Figure 3.1.

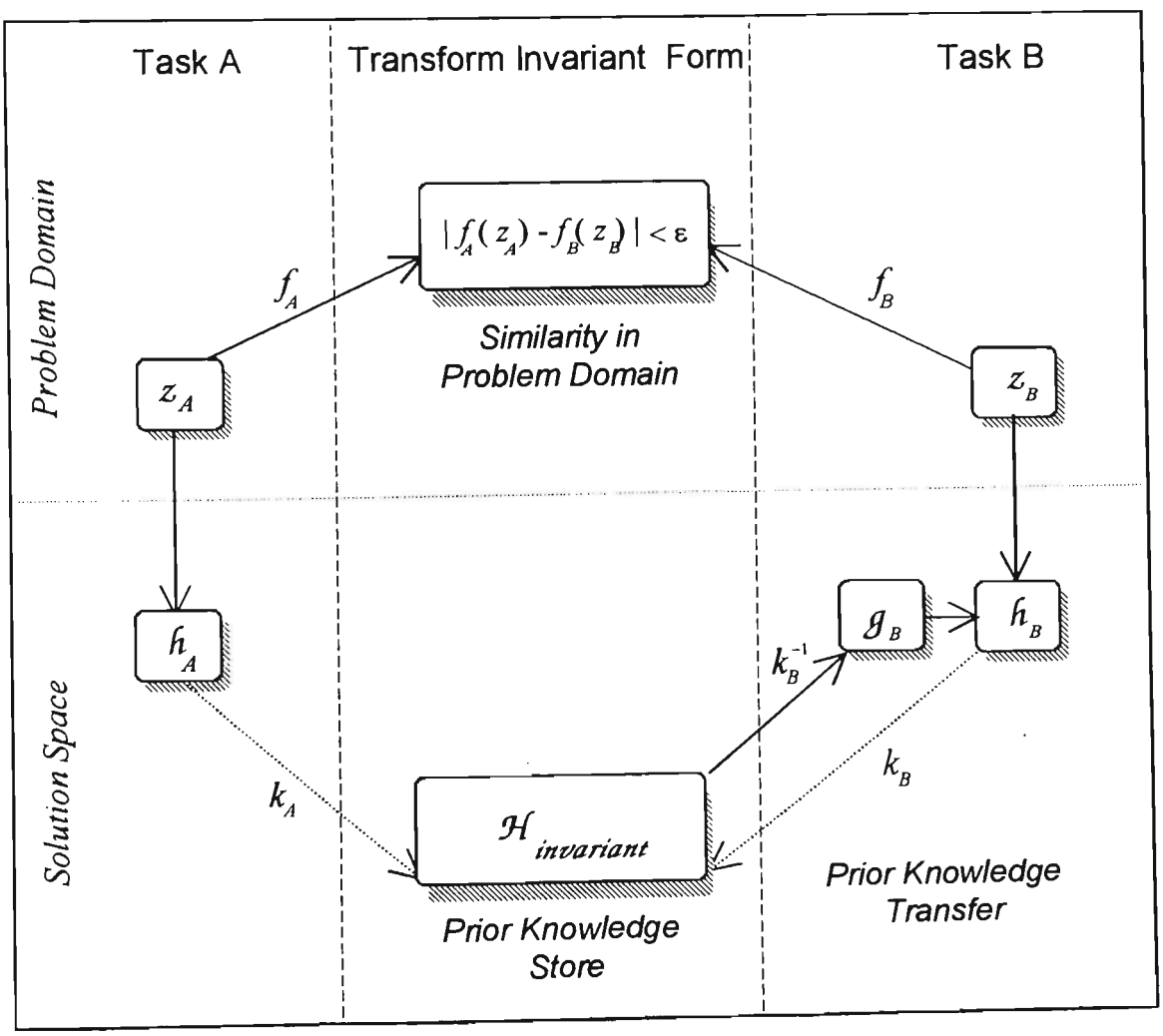


Figure 3.3 : Transfer of knowledge from task A to task B via transformation invariant form

given environment, if tasks A and B are similar, then ideally their position on the map of weight space should also be close. To this end it will be shown that the transformations which take place in the problem domain can be modelled and compensated for in the neural network's representation space. To achieve this goal, equations to calculate k and its inverse for affine transformations are derived for two commonly used forms of problem abstraction.

3.3 Detecting Affine Transformations in the Training Data

The detection of transformations in training data is a topic which could easily occupy a thesis to itself. As the emphasis in this chapter is upon the transformations possible in the network weight space rather than in the problem domain, the treatment of this topic is necessarily brief. There is a degree of circularity in the practical use of these solution space transformations, as without a method of detecting transforms in the problem domain they are obviously impossible to correct solution space. Thus some practical examples where this detection can be done are now shown.

Transform detection in the training data has been restricted in this chapter to examples from image processing and computer vision. Many of the techniques described here however can be adapted for use in other problems that can be similarly formulated. For example speech can be represented as a 2D spectrogram (see section 6.2).

It is important to emphasise that for the transformations to be corrected in the weight space they must be an integral part of the task environment. For a face recognition system it will normally be desired that, at the very least, it is invariant to translations in the image. There is thus no advantage in checking or correcting for translations in the training data when transferring training to a problem where the face images are all taken from the same view, as this is not a distinguishing feature of that task. However if a new system is to be trained to recognise faces from a different angle, the transfer of training would benefit from being able to align features to be consistent with the available prior knowledge. A more obvious example is learning an approximation to facial images (Section 3.4) where any transformations that can be compensated for from previous problems are immediately relevant, as the task domain is the image itself.

Problem domains can be divided into at least[2] two main classes, classification and approximation. Provided that there is environment consistency (Section 3.1.1) then we

[2] Hakin [Hak94] pp 66-67, divides learning tasks in six categories Approximation, Association, Pattern Classification, Prediction, Control and Beamforming

know which part of the training data (input or output) to examine to determine similarity preserving transformations.

As an example consider the sets of character recognition data for different fonts as used in chapter 6.1 (see Appendix E.2). In this case an effective method of determining affine transformations is to place a bounding box around the mean of the training set (so getting the average character shape) and then to relate the relative distortion of this box to the type of transformation k that must be performed in the solution space. The bounding box is found by contouring the mean image at a level of 10% of the maximum output and then determining the rectangle that contains this contour. The coordinates of the corners of the rectangle provide information about the average amount of translation, scale and shear which is present in the particular training set (Figure 3.4).

The detection of affine transforms in images is a problem which has yet to be solved efficiently for the general case. There are many techniques however that will work well with certain classes of inputs. The radon or Hough transform is one such method which can be used to detect rotation and scale transformation in images provided they are well centred. It works by transforming the image coordinates from Cartesian space, to a space defined by the pixels angle and log distance from the center of the image.

More sophisticated methods have been developed using biologically motivated filters, such as Gabor filters. These filters, which are known to model the simple cells found in the visual cortex of mammals, have a number of properties that make them optimal for processing real images [Dua85]. They are created from Gaussian modulated sine waves which can be at different frequencies and spatial orientations and hence essentially compute a short time windowed Fourier transform on the image. A neural model that uses these filters is known as a dynamic link architecture [LVV+93]. It demonstrates very impressive performance at detecting both affine and perspective transformations on real images.

Ideally if any three or more features can be consistently located in an image then all affine transformations on that image can also be determined, assuming no non-linear transforms. Such a technique using Gabor Filters has been developed [Dun95b] for rapidly detecting and normalising affine and perspective transforms in facial images, regardless of most expression and lighting changes. Three facial points could be found consistently using this technique; the end of the nose and the two eyes (Figure 3.5).

In some situations it is possible to determine what transformation has been placed on the training data by using some external information about the problem. For example a single surface learning network (see Section 3.6.1) has difficultly encoding complicated images because the decision regions extend the entire length of the image and thus have unwanted global effects. One method around this problem is to break the image into smaller pieces and train separate networks to encode these smaller subsections of the

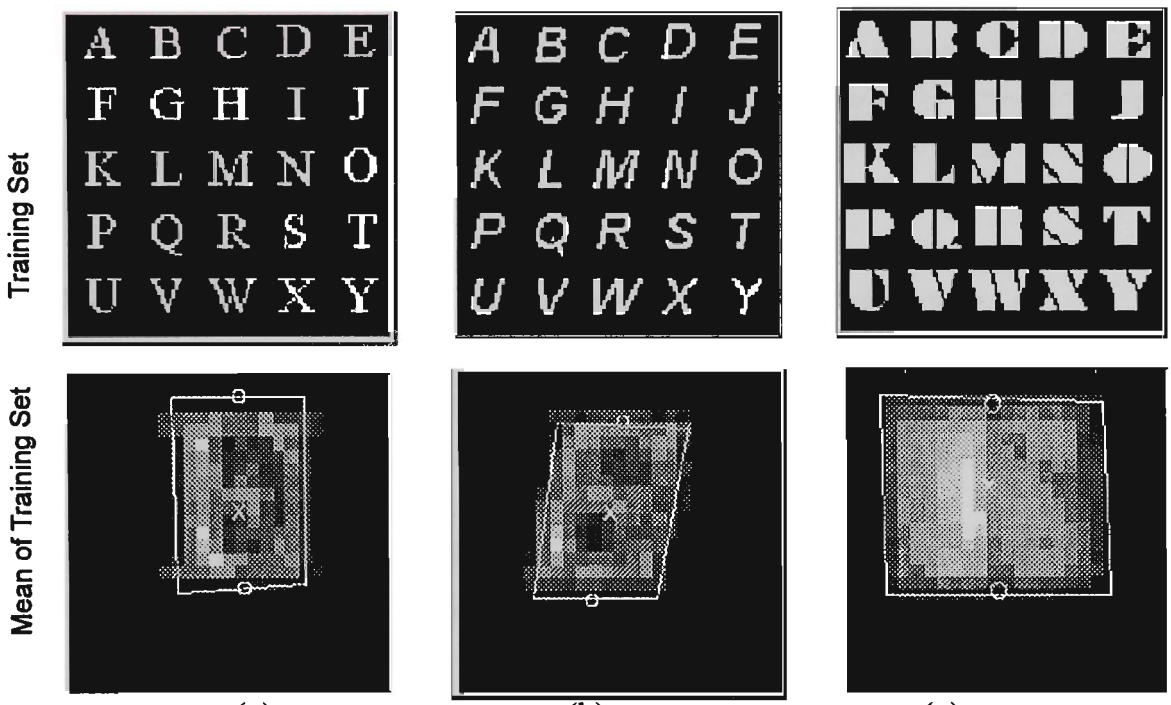


Figure 3.4 : Detection of affine transforms on charater recognition training data.

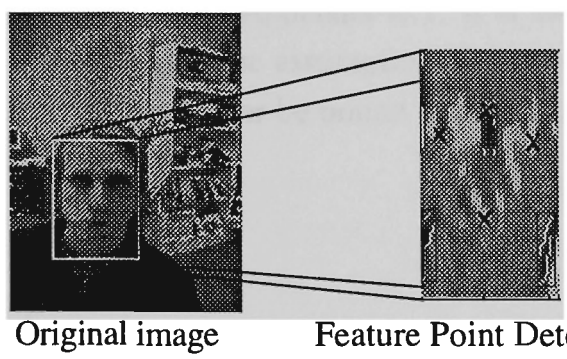


Figure 3.5 : Detection of affine transforms in more complicated computer vision data.

image [Dun94a, DO95]. If we wish to use prior knowledge gained from the image encoding of other network's to speed up a networks training then the appropriate affine transforms on the network weight vectors can be easily calculated by the knowledge of the position of the image subsection used for training.

3.4 Affine Transformation Invariance in the Weight Space

The effect that affine transformations in the training data have on the neural network's learned representation is of interest if k_{affine} is to be calculated. It can be shown that such solution space affine transformations can only exist in the first layer of the neural networks weight space, W^1. This is a direct consequence of equation 2.6. However, since the following sections rely on this property, a formal statement is now given.

THEOREM 3.1 : *In a finite MLP network topology, where the input is connected to the first layer only, affine transformations T_A in the problem domain Z can be achieved by a function $k_A(\mathrm{T}.\mathbf{w})$ where $\mathrm{T}.\mathbf{w} = W^l$, if and only if $l=1$.*

T is a binary membership vector for the individual weights in weight space. For the purposes of this theorem it constrains k_A to operate on the weights of only one layer. The proof for this theorem may be found in Appendix C.1. It is almost certain that a more general proof also exists which relaxes the assumption $\mathrm{T}.\mathbf{w} = W^l$. This would make the proof stronger because k_A would no longer be bound by the constrains of operating only a single layer.

3.5 Weight Space Transforms

The kind of function approximated by a neural network is different depending on the representation of the environment. Each form of abstraction configures the weight space in a distinct way. The applicable weight space transformations are thus entirely dependant on the form of problem abstraction. Two specific cases are considered here, namely multi-dimensional surface approximation and auto-correlated classification.

The following sections show how transformations on the weight space, for these abstractions, can result in affine transformations in the problem domain. Or alternatively, the effect that transformations in the problem domain have on the weight space. These weight space transformations need operate only on the first layer weight vectors as a direct result of theorem 3.1.

3.5.1 Surface Approximation and Interpolation

This type of functional approximation deals with situations in which the inputs to the network are orthogonal. The inputs can then be treated as independent orthogonal variables. The effect that an affine transform in the input x has on the first layer weights $w_{1..n}$, can be derived by equating the untransformed weight vector with an appropriately transformed copy.

For this type of encoding environment the weight vector can be manipulated to perform affine transformations on the input domain because the weights connected to one input are orthogonal to the weights connected to any another input. The decision regions formed may thus be analysed by simply using the rules of linear transformation. Examine the equation which occurs at a neuron j in the first layer (eq. 2.5),

$$N_j^1 = \sum_{i=1}^{n} w_{ji}^1 x_i^0 + \beta_j \tag{3.3}$$

expanding this,

$$N_j^1 = w_{j1}^1 x_1^0 + w_{j2}^1 x_2^0 + \ldots + w_{jn}^1 x_n^0 + \beta_j. \tag{3.4}$$

To further simplify the notation the layer numbers will be omitted and the equation vectorized over all the neurons in the first layer (each w, x and β is thus a vector of length j),

$$N = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + \beta \tag{3.5}$$

Assuming orthogonal input variables x_j implies that each set of w_j can also be treated as orthogonal. Using this knowledge it is now possible to calculate k_{affine} by substituting the change in the inputs x, into equation 3.5. By simplifying and equating coefficients, the required transformations on the weight space k_{affine} were derived (Table 3.2). Appendix C.2 contains the full derivations.

3.5.1.1 Bivariate Function Approximation

To demonstrate the effect of these transforms a problem is required where the output space of a network can be displayed in an easily interpretable form. One problem that fits this description well is determining the bi-variate function approximation to the surface described by a grey scale image. The position and intensity of each pixel in the image define a two dimensional surface to which the network must find an approximate solution. This problem has similarities with the two spirals benchmark [FL90] except that the input data is regularly spaced and the output target is continuous rather than binary.

Affine Transformation	Modification to first layer weights
TRANSLATION BY $(t_1, t_2, \ldots t_n)$	$\beta = \beta + \sum_{i=0}^{n} t_i w_i,$
SCALING BY $(s_1, s_2, \ldots s_n)$	$w_j = s_j w_j$
ROTATION BY (θ), (2D case)	$w_1 = w_1 \cos(\theta) + w_2 \sin(\theta)$ $w_2 = w_2 \cos(\theta) - w_1 \sin(\theta)$
SHEAR BY $(h_1, h_2, \ldots h_n)$	$w_j = w_j - h_j w_j + \sum_{i=1}^{n} h_i w_i$

Table 3.2 : Surface Approximation Affine Transformations

A number of papers demonstrating the use of this technique for compression, image processing and parallel image encoding [Dun94a, DA94, DG95] have been published. Whilst this technique has some advantages over other image representations, such as good edge reproduction at high compression ratios and scale invariance, the emphasis here is only on its use to demonstrate the effects of the previously derived k_{affine} transforms.

The neural network used to learn the image has two inputs for the pixel coordinates (normalised to lie in the range [0..1]) and an output that approximates the intensity value of the image at the point (also scaled between [0..1]). It is trained to learn the association between a particular location in the image and the gray level intensity. The training set z is thus defined by the coordinates for every pixel in the image and their associated intensities,

$$\vec{z} = \begin{cases} \text{input: } x_1 = [0..1], x_2 = [0..1] \\ \text{output: } y_1 = Intensity_{\text{pixel}(x_1, x_2)} \end{cases}$$

During learning the network converges towards a functional approximation to the images surface. How closely any image can be approximated is obviously dependent on both the internal structure of the neural network and the characteristics of the image. Any continuous surface can theoretically be reconstructed with only one such hidden layer [HSW89], for real images however at least two hidden layers are required to provide sufficient abstraction with a practical number of neurons.

Sigmoid activation functions were used in all neurons. Thus the output of each first layer hidden neuron was a two dimensional sigmoid surface,

$$A(w_1, w_2, \beta) = \frac{1}{1 - e^{-(x_1 w_1 + x_2 w_2 + B)}} \quad x_1, x_2 \in [0..1]$$

Affine transforms in the image effectively act to rotate, scale and translate an input window over the "generic" activation function surface [DA94]. The ratio of two weights w_1 and w_2, controls the rotation of the input window whilst the bias weight β represents the movement of the input window in the direction of the rotation. This flexibility allows these functions to represent various edge orientations and gradients. This first layer will be referred to as the *basis* layer, because all higher layers take their input from here. The surfaces generated from this layer are scaled and combined to form regions of various intensities at the second hidden layer. In essence this is combining the *basis* functions to form image "features". These features are then used to form an image by a weighted combination in the output layer. To demonstrate the different functioning of the layers an image of a face was encoded using a network with 16 hidden neurons in the first layer and 8 in the second. The surfaces calculated by the neurons in these layers are shown in Figure 3.6a. Each sub-square within a layer represents the output of a neuron over the entire image, and so is actually the same dimension as the output image.

Figures 3.6 (b,c) show the specific effect that some of the affine transformation equations given in table 3.2. have on the networks output. These transformations take place on the first layer weights only. The feature layer results appear similarly transformed, although the weights in this layer are unchanged, because the feature layer neurons use the results of the basis layer.

Given an image I and an affine transformed image $T_A(I)$, if the learnt representations are similar then the weights in layers higher than the first should also be similar, by theorem 3.1. One difficulty with this comparison is that network weights from different training runs cannot be directly related due to symmetries inherent in the neural networks topology. This problem will be examined further at the end of Chapter 4.

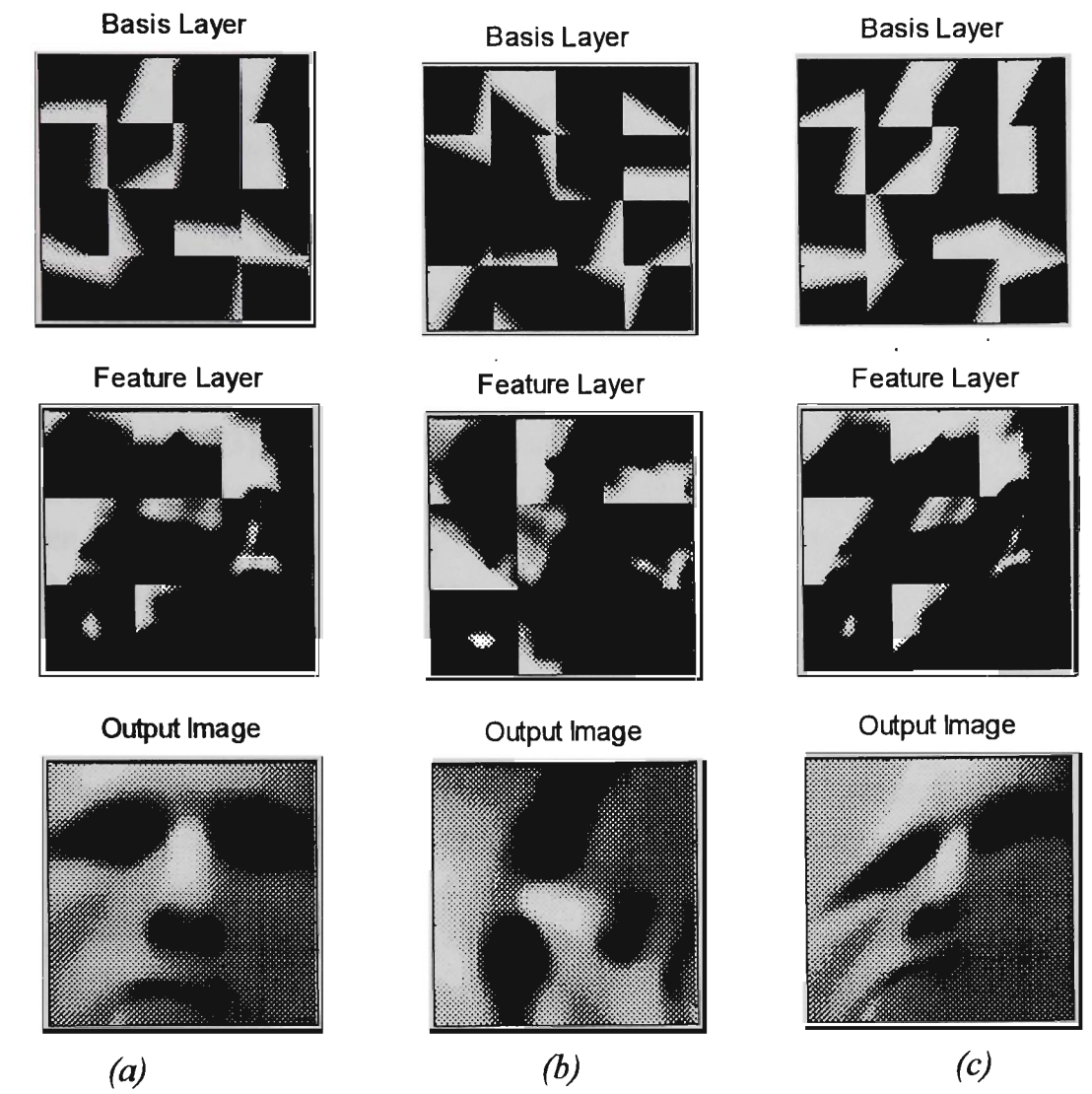


Figure 3.6 : Example neuron outputs from surface learning network

3.5.2 Auto-Correlated Classification

For a classification environment, the output space of the network is intended to indicate how strongly a particular input is associated with each output. This may be in the form of an estimation of the confidence (or probability) that an input has been correctly classified at an output, or it may simply be a binary decision giving no indication of the networks confidence in that decision. The input variables in these cases are often neither independent nor orthogonal. To take a specific case, consider the classification of an image where the input to the network is the pixels from the image and the output indicates an object type. As we are considering an image, the problem domain really only has two dimensions (or maybe slightly more if we are considering the fractal dimension [BBV93]), however the input dimension is the same size as the number of pixels in the image. Which will be much larger than two. This is known as the curse of dimensionality [DH73], where there is a very large input dimension, but only a small subset of that space contains valid representations. If the sampled pixels are close to each other then there is a high probability that the pixels will be strongly correlated with each other - this is known as auto-correlation.

Input samples must always be at the same points in the input space, otherwise the network has no way of knowing what it is classifying. If the sampling position for any of the pixels in an image can independently change, then the classifier cannot be expected to produce an appropriate classification as there is no way for this change to be reflected to the network (the network only has inputs that define the *intensity* of the pixels, not the *position*). Each input layer weight thus has a direct association with the point in the problem space from which the training data has been sampled. This forces the input weights to form masks or templates which go into each neuron. For a two dimensional image with input data sampled on a 4 by 4 grid, the mask for neuron j in the first hidden layer would be,

$$\boldsymbol{m}_j\boldsymbol{x} = \begin{bmatrix} w^1_{1j}x_{(1,1)} & w^1_{2j}x_{(2,1)} & w^1_{3j}x_{(3,1)} & w^1_{4j}x_{(4,1)} \\ w^1_{5j}x_{(1,2)} & w^1_{6j}x_{(2,2)} & w^1_{7j}x_{(3,2)} & w^1_{8j}x_{(4,2)} \\ w^1_{9j}x_{(1,3)} & w^1_{10j}x_{(2,3)} & w^1_{11j}x_{(3,3)} & w^1_{12j}x_{(4,3)} \\ w^1_{13j}x_{(1,4)} & w^1_{14j}x_{(2,4)} & w^1_{15j}x_{(3,4)} & w^1_{16j}x_{(4,4)} \end{bmatrix}$$

x-axis →

↓ y-axis

Usually the set of masks will end up encoding "stereotypical" composites of the training data. For example a network trained to classify lines in an image using a sliding window, will end up with weight masks that essentially show lines encoded at different orientations [Spe89]. These weight sets can be viewed as convolution filters. The output

of a first layer neuron indicates how strongly the input pattern matches the associated weight mask.

The appropriate affine transformation algorithms k_{affine} for this environment depend on both the dimension of the problem domain and the way the training data was sampled. It can be seen however that for regularly spaced grids, these masks may be manipulated using image processing style operators,

$$k_{\text{affine}} = f_{\text{operator}}(\boldsymbol{m}).$$

For instance a ninety degree rotation of the problem domain can be implemented by simply taking the transpose of the mask for each neuron,

$$k_{\text{rotate\_90}} = (\boldsymbol{m}_j)^{\mathrm{T}} \quad 1 \geq j > \mathrm{n}_l \,. \tag{3.6}$$

Many other affine transforms however will leave some of the weights without explicit values, eg. rotation by 45 degrees or any translation. Because the aim is to transfer training, it is appropriate to initialise unknown weight states in the same way a completely new network would have its weights initialised, such as with small random values. For the translation case,

$$k_{\text{translate}}(t_1\, t_2\, ..t_q) = \begin{cases} \boldsymbol{m}_{j(l_1,l_2,\ldots,l_q)} = \boldsymbol{m}_{j(t_1+l_1,t_2+l_2,\ldots,t_q+l_q)}, & l_i - t_i > 0 \\ \boldsymbol{m}_{j(t_1+l_1,t_2+l_2,\ldots,t_q+l_q)} = rand, & l_i - t_i \leq 0 \end{cases}, \; 1 \geq j > \mathrm{n}_l \tag{3.7}$$

3.5.2.1 Face Recognition

Face recognition techniques can be divided into two broad groups, *holistic* and *local*. *Local* techniques such as the Dynamic Link Architecture [LVV+93] use features on the image to find a best match between two face images and then return this lowest energy state as the recognition result. *Holistic* techniques, in contrast, have the a direct pixel to input mapping as described above. Much of this research has concentrated on auto-associative style neural network models [PT90,FC90]. Generally these holistic techniques have difficulty recognising faces under real world conditions. Any object that is plastically deformable, and which may be viewed under varying lighting conditions, will usually have an insufficient match between the stored masks and the input image, even given the appropriate training data. Despite this a holistic technique is considered here becuase it represents a good example of an auto-correlated classifier.

A training set was constructed using twelve normalised and centred facial images of resolution 32 by 20 pixels (Figure 3.7). The architecture used for this experiment had a

single hidden layer with four neurons, an input for each pixel in the image and a single output to classify each face in the training set (Figure 3.8). The network was trained to converge to a mean square error of 10^{-2}, at which time all faces were correctly recognised from the training set. Figure 3.9 shows the first layer weight masks ***m*** that were encoded after training. Although these masks are still noisy it can be seen that they encode head-like images. Any transformations to these weight masks, such as a translation or a rotation, will result in a network that responds with the same error level to input data which has been similarly transformed. This occurs because the output of the first layer neurons is simply a measure of the approximate correlation between these masks and the input pattern.

3.6 Seeding Weight Space with Transformed Solutions

Due to the difficulty of determining affine transformations in a general learning environment an alternate method to achieve some measure of transform invariance in weight space is now proposed. This technique increases the functional diversity of the prior knowledge by generating sets of new weight solutions using the affine transformation algorithms just presented. It has similarities to the hints technique described in Chapter 2, whereby extra training examples are generated from those available. Using the prior knowledge, the hypothesis space is seeded with transformed copies of the learned solution vectors. The operation of this "seeding" at a fundamental level is conceptually very similar to the transformation invariant weight space. It alleviates however the most difficult aspect, that of deriving an algorithm to detect such transforms in the training data. Figure 3.10 demonstrates the relationships between the two techniques. Each cluster of shapes represents a network solution, where the different shapes are sample classifications of a network in weight space, and **x** is the desired feature vector for the newly presented target problem. It can be seen that the desired feature vector **x** doesn't need to be moved in transformation seeding, so it is not necessary to detect the affine transformations in the problem domain.

The drawback to this technique is that there is a practical limit to the number of possible transformed parameters that can be tested, as each new transform parameter increases the effective size of the prior knowledge by another solution. Despite this, the method provides very real advantages for increasing the prior knowledge diversity (similar to that shown in standard training data with hints [Abu95]), especially in cases

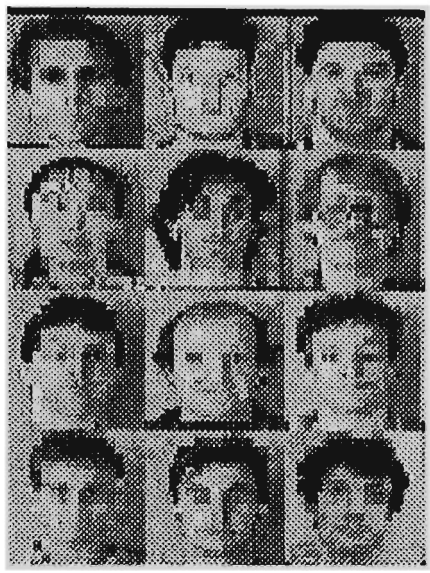


Figure 3.7 : Face images used to train network

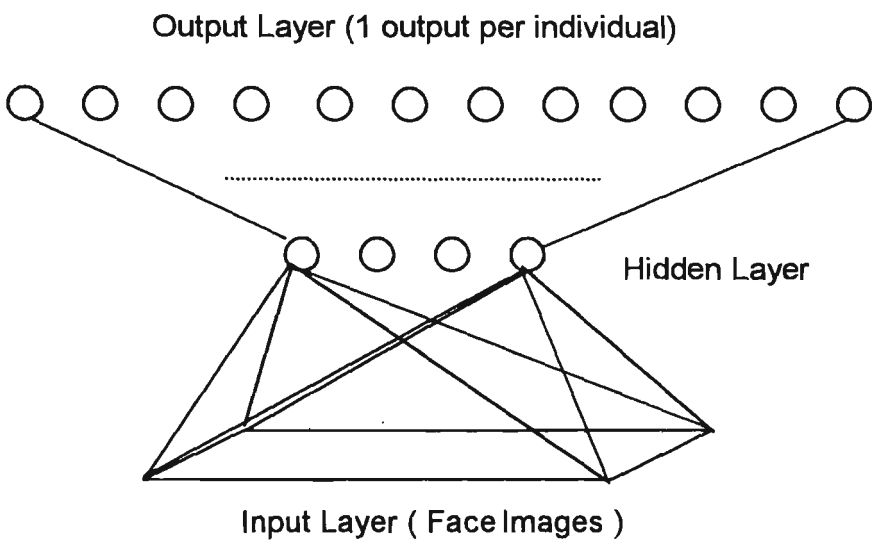


Figure 3.8 : Neural Network for simple image classification

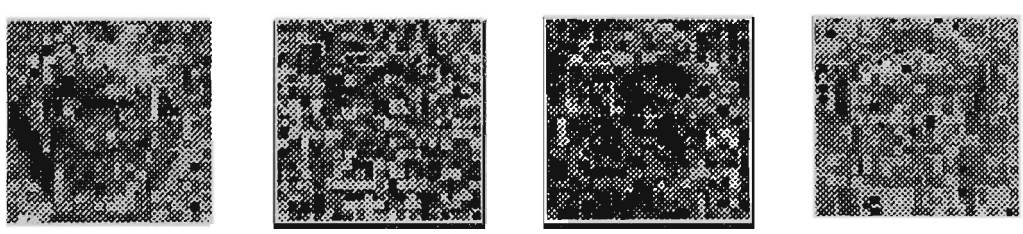


Figure 3.9 : First Layer weight masks learnt for correct classifiction of face images in Figure 3.3

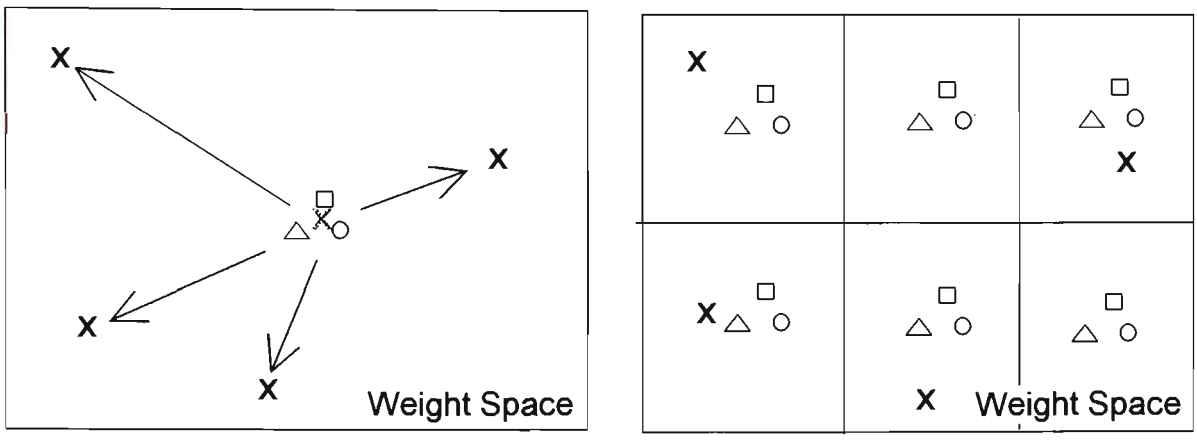


Figure 3.10 : Diagrammatic representation of the difference between normalised form and transformation seeding for providing transform invariance.

where the original number of prior knowledge solutions is small. Its advantages under practical conditions shall be shown in Section 6.3.

3.7 Summary

This Chapter has formulated a basis for representing task solutions in a manner that is independent of similarity preserving transformations. This allows the extension of a solution for one task to be made applicable across a range of other tasks that are related by a environmental similarity metric. It was then explained what consisteny in an environment means and why this is a desirable characteristic when transfer is to take place.

The determination of general techniques for detecting transformations on the solution space was identified as a difficult task. Some methods for this detection were then described for image training data.

A simple proof was given that affine transforms on the problem domain could only be produced by changes to the first layer weights representations. This ability to perform affine transformations in the problem domain was shown for two problem environments; surface approximation and auto-correlated classification. Specific task instances within these areas are shown to backup these claims.

The chapter concludes with a section on seeding the weight space. This alleviates the need to detect affine transformation on the training data at the expense of increasing the prior knowledge size.

Chapter 4

Weight Space Symmetry

As previously described, the process of obtaining and using prior knowledge in a machine learning system can be likened to mapping its hypothesis space. On this map it is desirable to have equivalent solutions located in the same position. As an example, consider a city with a mirror running through its middle. Much like Alice in 'Through the Looking-Glass'[1] it is possible to travel to the other side of this mirror, but unlike Lewis Carroll's tale everything appears exactly the same on the other side (your brain is reversed when passing through the mirror). If one wished to make a map of the city it would be unnecessary to visit both sides of the mirror, since a knowledge of one side completely defines the other. Now consider the situation where you are instantly transported to random locations in the city on either side of the mirror. Constructing a useful place map from just this information is much more difficult when the position of the mirror is unknown because only one side actually needs to be mapped. Two places that are physically close can appear to be far away if they are encountered on opposite sides of the mirror. The only way to avoid this problem is to know the exact location of the mirror. When this is known it is trivial to determine which side of the mirror you are on and to make the appropriate mathematical transformations to flip to the other side of the map if necessary.

This situation is analogous to the topology found in the solution spaces of most neural networks with hidden neurons. The difference being that the number of 'mirrors' present in a neural network of practical size is enormous, growing faster than the factorial of the number of neurons in each hidden layer. When a network is randomly initialised by assigning the weight values from some fixed probability distribution, as is common, the inital weight vector can end up in any one of these many different topologically identical or symmetric regions. Current learning algorithms take no account of these symmetry regions, since from a learning perspective they appear to have little effect, provided the symmetry boundaries are avoided as starting points. Perhaps because of this there has been little research done on the effect or decomposition of these

---

[1] *'Through the Looking Glass'* children's story written by Lewis Carroll (*1871*) and influenced by a mathematician, sequel to *'Alice's Adventures in Wonderland'* (*1865*)

symmetric regions despite their fundamental underpinning to the structure of weight space.

One immediate consequence of the symmetries is that there is no easy way of directly comparing neural network solutions. Even if two networks produce exactly the same response to all possible inputs, a situation that is called *equi-output* [CH93], the Euclidean distance between the actual weight vectors will usually be quite large. This has serious consequences for the use of prior information. When gathering solutions for placement on the map of weight space, in most cases there is no way (short of testing every possible input to the neural network) to determine if a solution is the same as one seen previously. It must therefore be recorded as a separate solution even though it may encode exactly the same function. For this reason the determination of weight bounds on the solution space is almost impossible in the presence of such symmetric regions.

Consider that we receive a number of different solution vectors to a problem, all of which lie in different symmetry regions. If they are then placed on the map of solution space it is not possible to infer any local bounds on regions useful to this problem. This is because the space between any two solutions in different symmetry regions is likely to contain a good proportion of the total unique solution space, which obviously has nothing to do with the required bounds for this problem. To return to the starting analogy, let us assume we have prior knowledge that finding two or more shops of a given type in close proximity indicates that there are likely to be other similar shops between the known locations. In our city however the location of two antique shops does not allow one to guess with any certainty that there may be an extended region containing more antique shops unless it is known that both shops are in the same symmetry region. If the shops are found on different sides of the mirror, guessing that there is a region between them that contains mainly antique dealers is completely unfounded and probably wrong.

4.1 Symmetry Classification

The types of symmetries described above belong to a class known as discrete symmetries, because each one folds the weight space into two exactly equal halves. Another form of symmetry, called continuous symmetry, also exists in weight space. As its name suggests, this latter form of symmetry occurs when some of the elements in the weight vector can be continuously modified, within bounds, with no change in the network's output response. The conditions under which this occurs tend to be very problem dependent, for example it could be that the continuous increase in one set of weight elements can be exactly offset by a continuous decrease in another set. This form of symmetry however has little effect on the use of prior knowledge. Given any two

solutions along this continuous symmetry region, the line between them contains equi-output solutions. An understanding of these continuous symmetry regions could be useful to a prior knowledge system by providing it with a better set of bounds on the solution space. However, unlike discrete symmetries, it is not critical to the use of such systems. Because of this, and the problem dependent nature of this type of symmetry, the emphasis in this chapter is on discrete symmetries only.

Graph symmetries are dealt with in some other practical fields as well, including parallel computing and telecommunication networks. However the emphasis in these other areas tends to be quite different from that discussed in this chapter. In general the network topologies considered are also quite different (usually much sparser) and the weights on connections between nodes are also often integer values. Most importantly however the problems being solved are frequently concerned with the determination of shortest paths [IZ93] or other weighted path type algorithms [Wer93]. These problems are only indirectly related to obtaining a weight set that exists in a minimal symmetry region. To further differentiate the approaches, these other areas have no concept of sign symmetry, continuous symmetry or equi-output solutions as they apply in the neural network field.

4.1.1 Alternate Symmetry Definitions

The term symmetry in neural network theory has a some other meanings in addition to that referred to by the title of this chapter. These are now briefly discussed.

The introduction of symmetries into the network weight vector, by constraining separate weights in weight space to always share the same value, is often called *weight sharing*. This technique is used to reduce the bias in networks by restricting the number of free parameters (weights), its main benefit is to allow certain transformations in the network's inputs to be rendered invariant at the output, for example graph isomorphisms. For some problems this technique has proved to be very effective in reducing training times and improving generalisation [Sha93]. These networks still suffer the same symmetry problems previously described, but the number of symmetry regions can be reduced. This effect will be used to advantage later in this chapter to help visualise the network symmetries.

Statistical mechanics is a technique for examining the statistical properties of large numbers of simple interacting elements. It is a general mathematical tool that has found application in areas such as thermodynamics and quantum mechanical systems. The application of this technique in the neural network field [Gar87] has been well formalised for some specialised network architectures. However for most architectures, including simple feed-forward style networks its application is still undergoing research with toy

networks [MO94] [2]. The notion of symmetry in statistical mechanics is usually different from that of symmetry in weight space. It involves a concept called replica symmetry which is found in the output space of the network (often referred to as the phase space). In simple terms, it is the symmetry that is found due to phase changes over the path that a network follows through weight space whilst being trained. As such it is tied to the particular learning algorithm used during training. Some research [BHS92] has related this problem to the symmetry found in weight space, however the current research only considers extremely restricted problem domains and learning situations. These situations commonly include such assumptions as very simple learning algorithms, binary neurons, binary weights and extremely simple network architectures. A practical technique for decomposing these symmetries for general multilayer networks has thus not been given. Future developments in this field are likely, however, to yield more results that are directly relevant to the problems considered in this chapter.

Although it would seem that the weight symmetry problem is well known, since the neural network literature on weight space topology often makes passing reference to it, there has been very little attempt to understand the nature of the symmetries or produce algorithms that exploit this symmetric nature. Hecht-Neilson, who was one of the first to publish references to this problem [Hec89], co-authored a paper [CLH93] in which the topology of weight space with these symmetries present was analysed with a view to the distance between symmetric solutions. It was shown that a minimal search cone existed in weight space, however a technique for moving weight vectors in any part of weight space into this minimal search cone was not given. A brief paper by Jordan et al. [JC92] used a basic technique to partially remove some of the network symmetries. In addition [Dun94b] also empirically demonstrates some other simple techniques for symmetric decomposition.

4.1.2 Chapter Overview

Given the importance of these symmetric regions in weight space to the use of prior knowledge transfer in neural networks, this chapter is devoted to analysing and proposing solutions to assist in decomposing network symmetries. The types of symmetries and the mathematical effects on the topology weight space are shown. Some proofs are derived to determine the equations of the symmetry boundaries and these are used to show the nature of the symmetric regions in solution space. The described

---

[2] The Hopfield model lends itself to this type of analysis because it can be easily considered as a mixture of random states and hence has a direct analogy to thermal systems. An architecture which is commonly used in the study of statistical mechanics in neural networks, precisely because it has no weight space symmetries, is called the committee machine and will be discussed later.

topology of the symmetric regions leads to a novel solution that uses simulated annealing and the angles between symmetric solutions to decompose the network symmetries. This technique is demonstrated, and its practical implications discussed using the results from simulations on different network sizes. An additional benefit of decomposing network symmetries is the increased ability to explain the internal workings of a network. The surface learning network introduced in Chapter 3 is used to illustrate this point.

4.2 Discrete symmetries in Neural Networks

Discrete symmetries are produced by modifications to the network connections that produce equi-output solutions. Each additional discrete symmetry can be visualised as a multi-dimensional mirror partitioning the solution space into ever more restricted regions that contain the minimal set of network solution vectors. The minimal solution set is exactly replicated across solution space, with the consequence that there are multiple global optima, one for each symmetry region. The symmetries can be divided into two types; *permutation symmetry*, which is concerned with the graph structure of networks, and *sign symmetry* that deals with the way an individual neuron processes its input signals to produce output signals.

Permutation symmetry is a direct consequence of the ability to view the network architecture as a unidirectional graph. Each possible reorganisation of this graph with no effect on the network output defines an isomorphism or symmetric reordering. Almost all networks that have hidden neurons will have such symmetry regions (Table 4.1). There are only a couple of exceptions to this, where there are no symmetries and yet there are hidden units. These architectures rely on the fact that each neuron has only one output connection or only one neuron per layer. One such network is known as a committee machine [BHS92] which is effectively a binary tree, with inputs at the deepest nodes and

Permutation Symmetries	No Permutation Symmetries
Multi-layered Perceptrons (MLP)	Hopfield
Learning Vector Quantisation (LVQ)	Self-Organising Feature Maps (SOM)
Radial basis function (RBF)	Cascade Correlation
Boltzman machine	Committee Machine
Counter Propagation Network (CPN)	
Elman (Recurrent) Networks	
Adaptive Resonance Networks (ART)	

Table 4.1 Permutation Symmetries in Neural Network Architectures.

an output at the top. Cascade correlation [FL90] is another example with no symmetries because each neuron effectively exists in its own layer.

In a strictly layered feed-forward network such a reordering is possible because pairs of neurons may be swapped within the same layer, provided that all the incoming and outgoing connections are also moved. Notice that this means that its effect on the weight space is limited to the weights in the layer above and those in the layer below. So within each layer the number of permutation symmetries is the factorial of the number of neurons.

Sign Symmetry occurs when some or all of the neurons in the network have an activation function where -G(-w) = G(w). This is called a bipolar activation function, of which tanh is a common example. Such symmetric functions are occasionally used in preference to non-symmetric functions, like the unmodified logistic function, because they have been reported to give faster convergence using current learning algorithms [Hin93, Hak94]. Like permutation symmetries the change in weight space is localised to the weight layer above and below, but involves a change to the weights of an individual neuron rather than a group of neurons. Thus the number of possible sign flips that can occur in a layer is equal to the number of possible binary combinations of the neurons.

Consider a network with L layers and n_k neurons in layer k. The number of permutation symmetries is,

$$N_{\mathrm{p}} = \prod_{k=1}^{L} n_k! \tag{4.1}$$

and the number of sign symmetries,

$$N_{\mathrm{s}} = \prod_{k=1}^{L} 2^{n_k} . \tag{4.2}$$

Appendix B.1 contains a simple inductive prove from [CLH93]. These symmetries are independent of each other and so the total number of symmetries in the network with bipolar activation functions is a combination of both,

$$N_{\mathrm{p}} \times N_{\mathrm{s}} = \prod_{k=1}^{L} 2^{n_k} n_k ! . \tag{4.3}$$

The number of symmetry regions thus increases exponentially with the number of neurons present. For a two layer network with one input, one output, 8 neurons in the first layer and 4 in the second, there are just over 3.96 billion symmetry regions within a 57 dimensional weight space.

4.3 Visualising Symmetries in Weight Space

Symmetry regions are hard to visualise in networks of a useful size, as the dimension of weight space is too large. By constructing small toy networks a simplified picture of this weight space can be shown, which still preserves some of the structure of the higher dimensional weight space. These toy networks use weight sharing and no bias to restrict the effective dimension of the weight vector to three, so that the symmetry boundaries can be pictorially represented.

Two such models are constructed to demonstrate both forms of symmetry. Consider the network depicted in Figure 4.1. This network has one input and one output neuron, three hidden neurons with no bias and a weight vector of dimension six. By forcing the fan-in and fan-out weights for each neuron to be equal, the effective size of the weight vector is reduced to three, w = [w1 w2 w3]. This allows direct visualisation of the network symmetry boundaries. Consider a weight vector w = [1 2 3]. Equi-output solutions are produced for each permutation of this vector (Figure 4.2). Note that there is no sign symmetry in the network regardless of the activation function because the weights into and out of a neuron are the same. All the equi-output solutions can be seen to lie on a sphere of radius $|w|$. The symmetry boundaries lie along the planes that are defined by the equality of the elements in the weight vectors. There are three such equalities in this network (w1=w2, w2=w3, w3=w1) and these carve the weight space into six (3!) separate pieces (Figure 4.3). This graphically demonstrates the meaning of equation 4.1.

All the planes intersect along the line defined by the equality of all the elements in the vector, w1=w2=w3, which lies at 45 degrees to all positive axes. Regions containing a complete set of non symmetric solutions are defined by each of the six symmetric slices. The best visualisation for a generalised network is an n dimensional sphere of radius r. The sphere is cut into $n!$ slices at 45 degrees off center to the positive axes. The volume of each slice contains all possible network solutions where the magnitude of the weight vector is less than r.

A different architecture will now be used to demonstrate the effect of sign symmetries on weight space. Figure 4.4 shows a network that contains two hidden neurons where the bottom two weights are independent and the top two are shared. This means there is only one possible sign symmetry because a change in the sign on one

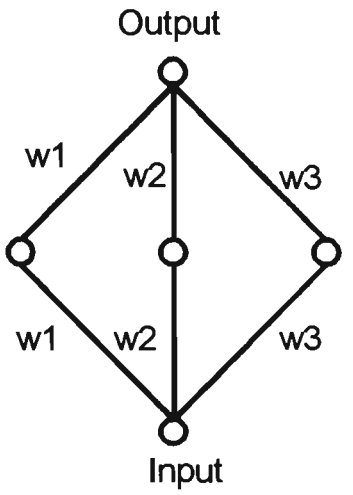


Figure 4.1 : Toy neural network for permutation symmetry visualisation, weight space w = [w1 w2 w3]

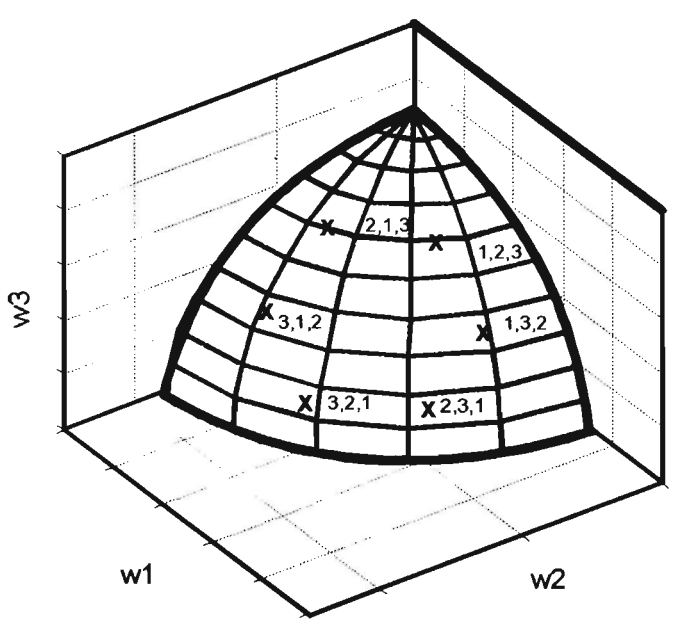


Figure 4.2 : Symmetric solution vectors for w = [1 2 3]

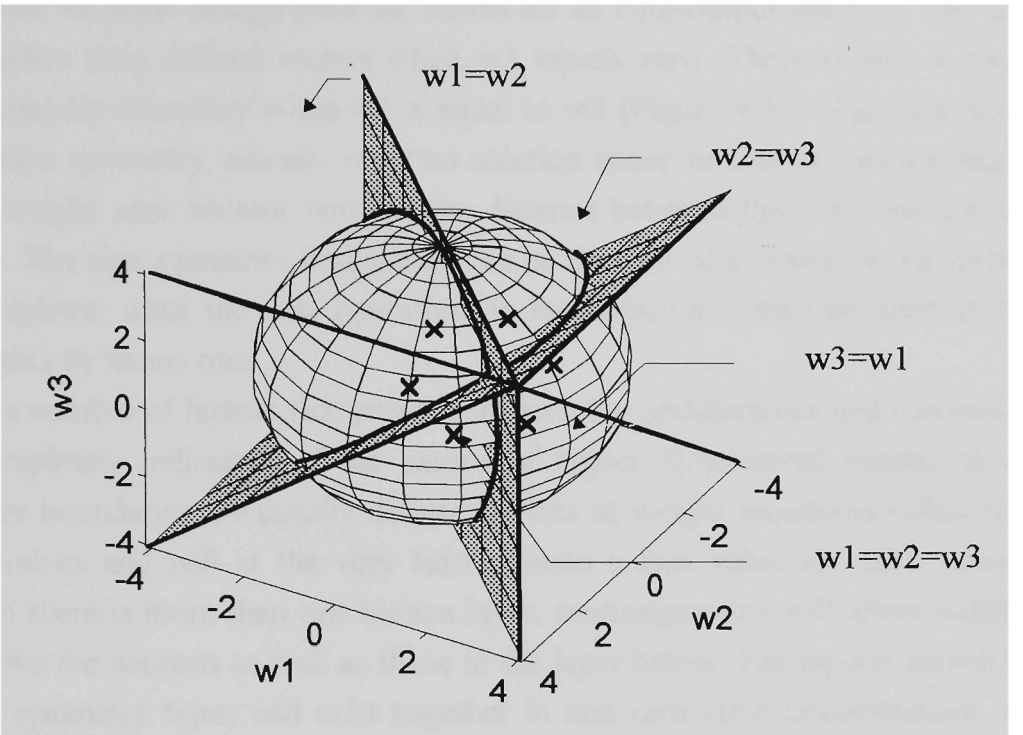


Figure 4.3 : Planes of permutation symmetry for architecture in Figure 4.1

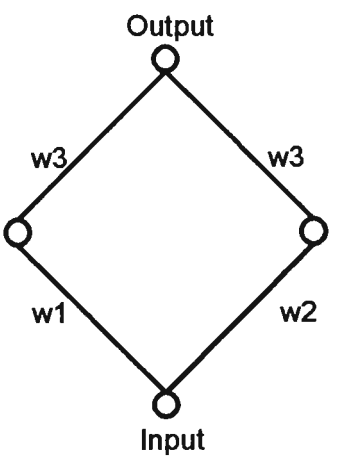


Figure 4.4 : Toy neural network for sign symmetry visualisation, weight space w = [w1 w2 w3]

weight means that we must change both the others for an equi-output solution. The sign symmetry boundary thus defined occurs when w3 equals zero. There is also a single permutation symmetry boundary when w1 is equal to w2 (Figure 4.5). Sign symmetry, unlike permutation symmetry, actually reverses solution space so that as solution angles relative to the weight axes become smaller, the distance between the solutions actually becomes larger. The sign symmetry boundaries can be thought of as lying on the surface of a unit hypersphere, since the only operation on each neuron is multiplication of the associated weights by minus one.

There are a number of factors that prevent the two toy architectures just considered from being completely indicative of the nature of higher dimensional weight space. Firstly symmetry boundaries are usually defined by sets of weight sequences rather than single weight values and will at the very least contain a bias value for each neuron. Secondly, when there is more than one hidden layer, rearrangements will affect weights in the layer above the neurons as well as those in the layer below. For bipolar activation networks both symmetry types will exist together in non restrictive circumstances. An attempt to give a more realistic visualisation of the weight space for larger dimensional networks is shown in Figure 4.6. In this figure the non-symmetric solution regions can be seen as slices of weight space radiating out from the origin.

4.4 Mathematics of Symmetry Boundaries

Now that the symmetries have been shown pictorially, a more general mathematical definition for the symmetry regions is derived. Hecht-Nielsen et al. [CLH93] proved the existence of minimal search cones for feed-forward networks using the distance between solutions on a unit sphere in weight space. The search cones correspond to areas of weight space in which no discrete symmetry exists and thus define a minimal sufficient set to find any possible network solution. The approach taken here is different because the emphasis is upon gaining an understanding of the equations governing the symmetry boundaries rather than on the demonstration of the existence and size of such regions. Hecht-Nielsen's proof is reproduced partially in Appendix B. Some alternate theorems about the nature of the symmetry boundaries that divide these regions are now presented.

DEFINITION 4.1 : *For a neuron i in layer l the neuron space ψ_{il} is defined as the vector of weights entering the neuron from layer $l-1$ and those leaving it into layer $l+1$. The set of all neuron spaces for a layer l will be denoted by Ψ_l.*

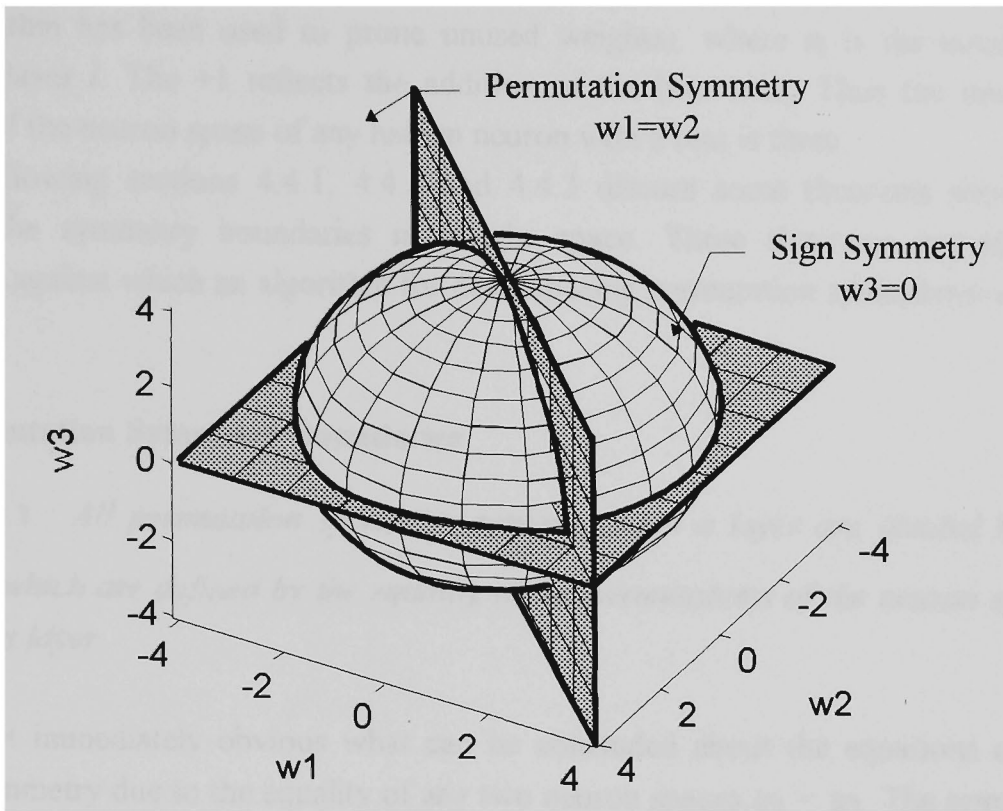


Figure 4.5 : Symmetry boundaries for architecture in Figure 4.4

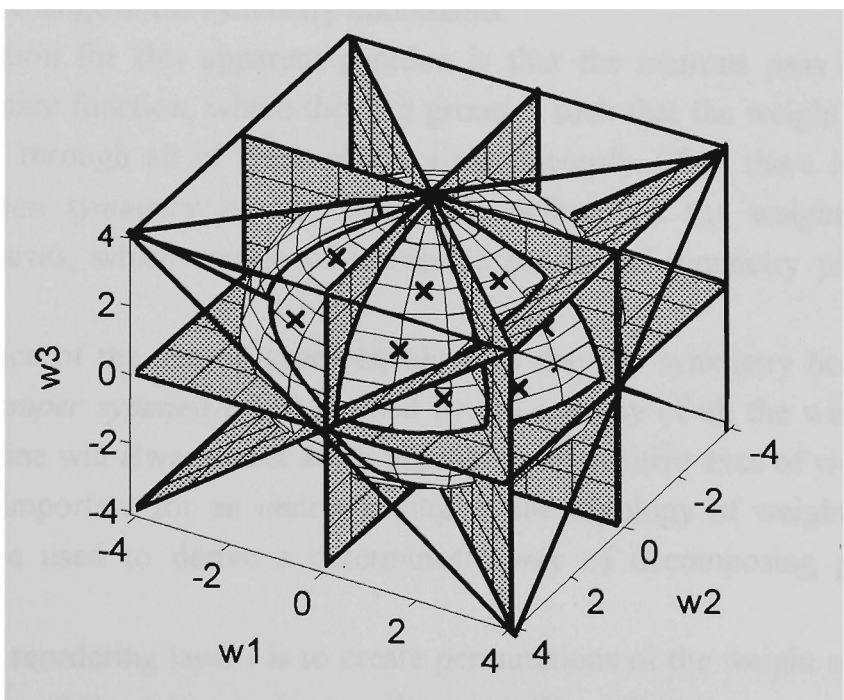


Figure 4.6. : Approximate visualisation for structure of planes of symmetry at higher dimension (note : clear boundaries will not be found in three dimensions as they would imply w1=—w2 etc. however this is a better visualisation of non-symmetric slices encountered in real networks of higher dimension.)

Generally ψ_{il} will have a dimension of $n_{l-1}+n_{l+1}+1$ in a feed-forward network (unless some algorithm has been used to prune unused weights), where n_l is the number of neurons in layer l. The +1 reflects the addition of the bias term. Thus the minimum dimension of the neuron space of any hidden neuron with a bias is three.

The following sections 4.4.1, 4.4.2 and 4.4.3 discuss some theorems about the nature of the symmetry boundaries in weight space. These theorems provide the background against which an algorithm for decomposing permutation symmetries will be constructed.

4.4.1 Permutation Symmetry Boundaries

THEOREM 4.1 : *All permutation symmetric regions within a layer are divided by* $\frac{n_l!}{2}$ *boundaries which are defined by the equality of the permutations of the neuron spaces* ψ *within that layer.*

It is not immediately obvious what can be concluded about the equations of the planes of symmetry due to the equality of any two neuron spaces $\psi_1 = \psi_2$. The symmetry boundaries are dependent only on the number of neurons, not on the number of weights in the layer. But any pair of corresponding weights in these vectors may be continuously modified with no change in the symmetry boundaries.

The explanation for this apparent paradox is that the neurons pass the weights through an activation function, where they are grouped such that the weight vector may only be reflected through all of these planes simultaneously. Thus there is a need to distinguish between *symmetry planes*, which are defined on the weight level, and *symmetry boundaries*, which consist of indivisible groups of symmetry planes at the neuronal level.

A consequence of the neuron space equalities is that the symmetry boundaries all intersect along a *super symmetric* line defined by the equality of all the weights in the network. Such a line will always exist at 45 degrees to all positive axes of weight space. This property is important for an understanding of the topology of weight space and could probably be used to derive a deterministic way of decomposing permutation symmetries.

The effect of reordering layer l is to create permutations of the weight sequences in the neuron spaces of the adjacent layers, Ψ_{l+1} and Ψ_{l-1}. This does not change the equations of the symmetry boundaries or planes because the individual neuron spaces all undergo exactly the same permutations.

4.4.2 Sign Symmetry Boundaries

THEOREM 4.2 : *In a network layer with n neurons with bipolar activation functions, there are* $2^{(n-1)}$ *sign symmetric boundaries which are defined by the neuron spaces being equal to the zero weight vector.*

The application of a sign change reflects all the weights in ψ through 180 degrees across symmetry boundaries. Because all weights in the network can undergo such reflections (with the exception of the bias(es) on the output layer) the planes of symmetry will lie on all the orthogonal axes of weight space. As with permutation symmetries these planes can only be crossed in neuron groups, not individually.

Unlike permutation symmetries however, sign symmetries in one layer can affect the sign symmetry boundaries in another. By changing the sign on the neuron space of one neuron the symmetry boundaries in adjacent layers may be altered, consequently there can be a flow on effect to all the other layers in the network.

4.4.3 Topology of Symmetric Weight Space

LEMMA 4.1 : *All discrete symmetry planes pass through the origin.*

It can be seen by inspection from theorems 4.1 and 4.2 that the zero weight vector is the only point in weight space which lies on all the planes of symmetry. This result is intuitively obvious because there is no offset on any of the planes of symmetry.

THEOREM 4.3 : *The symmetry boundaries in a bipolar activation function network define non symmetric regions in weight space which extend outward from the origin. The hyper-spherical surface areas described by weight vectors of a fixed magnitude will be elongated in weight space by,*

$$r_{p/s} = \prod_{k=1}^{L} \frac{n_k!}{2^{n_k}}.$$

The factor $r_{p/s}$ grows exponentially beyond a dimension of three (below three, $r_{p/s}$ is a fractional value) Thus the non symmetry regions of weight space are best described as extremely thin wedge-like regions extending from the origin.

4.4.4 Effect of Activation Function Polarity on Weight Space Topology.

Neural networks containing only bipolar activation functions have exactly the same representational ability as those with unipolar activation functions. The VC dimension (Appendix A.3) of a neural network architecture is independent of the polarity type of the activation function. The only reported difference from a learning perspective is that the training times for some problems are reduced in bipolar networks [Hak94 pg 160]. However the weight space in bipolar networks contains N_S times as many symmetric regions as the equivalent unipolar network. Thus the same number of solutions are contained in a more compact region of weight space. Compare figure 4.3, which is intended to show the interaction between sign and permutation symmetries, and figure 4.6, which shows the case for a unipolar activation function where only permutation symmetries are present. The unipolar activation function smears the solution space out along the permutation symmetric axis compared with the bipolar activation function network, which has its space further divided by the orthogonal sign axes.

Because the sign symmetric planes have no axis of super-symmetry, methods for analytically decomposing them will be different from that used to decompose permutation symmetries. They are also different because the effect of sign symmetry boundaries cascades from layer to layer. Many general networks use unipolar activation functions and achieve exactly the same end result (once normalised) but no network architecture can bypass permutation symmetries in the same way and make this claim[3]. Even in those networks which do have sign symmetries, for a practical number of neurons, these symmetry boundaries are greatly outnumbered by permutation boundaries (Theorem 4.3). In a layer with only 10 neurons the percentage of symmetry boundaries which are due to sign symmetry is less than 0.03 %. For the above reasons the following sections focus mainly on permutation symmetry. In terms of generality a restriction is simply being placed on the type of activation function used. As mentioned above this does not restrict the representational ability of the networks but simply makes the task of analysis easier.

4.5 Decomposing Permutation Symmetries in Weight Space.

As explained in the introduction to this chapter, symmetry regions in weight space interfere with the acquisition and use of prior knowledge from network ensembles. Thus the removal of symmetries is an important practical problem. The decomposition of discrete symmetries can be likened to the folding of weight space along the planes of

---

[3] Cascade correlation comes close, but tends to lack the encoding flexibility of a truly multi-layered structure, as each additional node takes as much of the remaining error as it can.

symmetry. For a vector in any part of weight space we wish to define some form of transformation D that will fold that vector onto a given non symmetric region. Obviously if the neuron spaces were scalars as they are in the toy examples, rather than vectors, then removing all permutation and sign symmetries would be easy. It would be simply necessary to order the spaces on these scalar values. With vectors however this ordering no longer makes sense unless it is possible to define a function that will give a unique and consistent encoding value for all the items in the vector.[4] A number of orderings [Dun94b, [JC92] can be imposed upon these vectors but none are able to decompose vectors into exactly the same symmetry region (remembering that the vectors contain real numbers) unless they have this property of uniqueness. All these other orderings achieve is to restrict the weight space, not find its minimal form.

4.5.1 Deterministic Symmetry Decomposition

The brute force approach to symmetry decomposition involves checking each possible symmetric copy of a weight vector against a reference weight vector until it can be determined that it is in the reference vector's symmetry region. This requires a number of iterations that is equal to the number of symmetry boundaries (or half the number of symmetry regions). For all but the smallest networks this will be an impractical number of operations.

The ideal deterministic algorithm for permutation symmetry decomposition would require $O(n\log(n))$ operations, where n is the total number of neurons. This is because the required algorithm is essentially one of folding along the neuron space boundaries, and would be analogous to a sorting routine in this respect. Such an algorithm could make use of the fact that permutation symmetries have a super-symmetric axis. By converting weight vectors to a set of spherical coordinates along this super-symmetric axis the permutation symmetries should be explicitly represented as angle bounds, which could be easily folded. Unfortunately complications arise when dealing with neuron spaces of a dimension greater than one, as these angle bounds are no longer as simple.

Another problem is that the difference in angles becomes vanishingly small as the number of neurons increases. The average angle for θ_n between two symmetric weight vectors in an adjacent permutation region in a layer which has twenty neurons, will be approximately $2\pi / 20!$ which is less than 2.5×10^{-18} radians. Thus the precision of the

---

[4] There is an algorithm defined by Gödel's theorem that achieves a unique encoding for a vector by raising each element to the power of sucessively larger prime numbers and then multiplying them together. This theorem is however of limited practical application because the numbers become too large for even quite small vectors.

mathematical operations and the resulting rounding errors are important even for relatively small networks.

In addition, this technique is specific to permutation symmetries and could not be generalised to work for sign symmetries. A good deterministic algorithm for symmetry decomposition is however an open problem and one worthy of further investigation.

4.5.2 Stochastic Symmetry Decomposition

The alternative approach for solving this type of problem is to use a stochastic algorithm. Stochastic techniques are well suited to this problem because of the discrete nature of the symmetries. They are also quite general and, although only tested on standard feed-forward networks here, the same algorithm could easily be adapted for any network topology such as those where connections can skip layers or even feed back into early layers (recurrent networks). The appropriate modification to decompose sign symmetries as well should not be particularly difficult. Stochastic techniques have the benefit that the sign symmetry problem needs to be solved globally rather than locally, which is something that these techniques do well. The negative side is that such algorithms are usually slow.

Unlike a deterministic algorithm, there is no way of checking whether or not a solution vector lies in the desired symmetry region, thus stochastic algorithms require a reference weight vector $\mathbf{w}_r$ that lies in the desired symmetry region to use as a goal. It can be easily seen that the radius values are equal for symmetric solutions, because the magnitude of weights remains the same and only their order is changed. Thus it is the weight vectors shperical angleϕ that contains the information relevant to decomposing symmetries. The general from of the algorithm D is given by,

$$D(\phi_{w1}, \phi_{w2}) \rightarrow \phi_{w1'} \quad \text{such that} \quad \varphi(\mathbf{X}:\mathbf{w1}) = \varphi(\mathbf{X}:\mathbf{w1'}). \tag{4.4}$$

The solution ϕ_{w1} is transformed to the same symmetry region as ϕ_{w2} (it is obviously important that neither weight vector lies on a symmetry boundary).

Perhaps the best studied stochastic technique is simulated annealing [KGV83]. It was inspired by the technique of annealing in metals where strength can be added, or different metals joined, by heating the metals and then cooling them rapidly. This causes the metallic bonds, which were originally randomly aligned, to form in a low energy state with high strength because the atoms have settled into a regular lattice. In simulated annealing an arbitrary temperature is assigned to the system and, in a similar way to thermal temperature, this controls the amount of randomness that is allowed to occur in the system state p. With the neural network symmetry decomposition this state represents the symmetry region of the weight vector. With a high temperature the weight

vector could exist in any symmetry region but as the system cools it is restricted to an ever decreasing area of the total weight space.

The aim of the annealing process is to optimise the set of parameters p against an error function E. When a random change is made to the set of parameters p it usually results in a change in the error ΔE calculated by the error function. This change will always be accepted if the error has been decreased, however it will also be accepted if the error is increased with a probability,

$$P(\Delta E) = \exp\left(-\frac{\Delta E}{T}\right). \tag{4.5}$$

Thus at high temperatures $P(\Delta E)$ will be close to one regardless of the value of ΔE, so almost all changes are accepted regardless of whether they improve the system state or not. As the temperature is decreased the system is forced into a state that almost always minimises the error. At any stage when the temperature is not zero, there is still a finite probability that the system will escape from any local minima in which it becomes trapped. This algorithm has had extensive research performed on its convergence properties and it has been shown that given a slow enough cooling rate the system will always find the global minima [GG84]. These cooling schedules however are far to slow for a practical implementation so it is necessary to set the annealing parameters depending on the type and size of the problem to be solved.

To adapt this technique for breaking network symmetries we must define a suitable error criteria. The best error criteria, for reasons mentioned earlier, is the angle between two vectors; a reference weight vector $\mathbf{w}_r$, which defines the minimal symmetry region into which all other solutions should be folded, and a target weight vector $\mathbf{w}_t$ which lies in some other symmetry region,

$$E(\mathbf{w}_r, \mathbf{w}_t) = \cos^{-1}\left(\frac{\mathbf{w}_t \bullet \mathbf{w}_r}{|\mathbf{w}_t||\mathbf{w}_r|}\right). \tag{4.6}$$

The smaller E is, the closer $\mathbf{w}_r$ is to $\mathbf{w}_t$ and the more likely that the solutions will lie in the same symmetry region (figure 4.7). The changes to the parameter space are made by randomly swapping the hidden units or randomly changing the signs on the neuron spaces, depending on which sort of symmetry is required to be broken. Appendix D.3 contains some pseudo code for implementing this algorithm. Each neuron regardless of layer has an equal probability of undergoing a symmetric modification at any given annealing cycle. The following theorem gives a justification for the use of this error criteria,

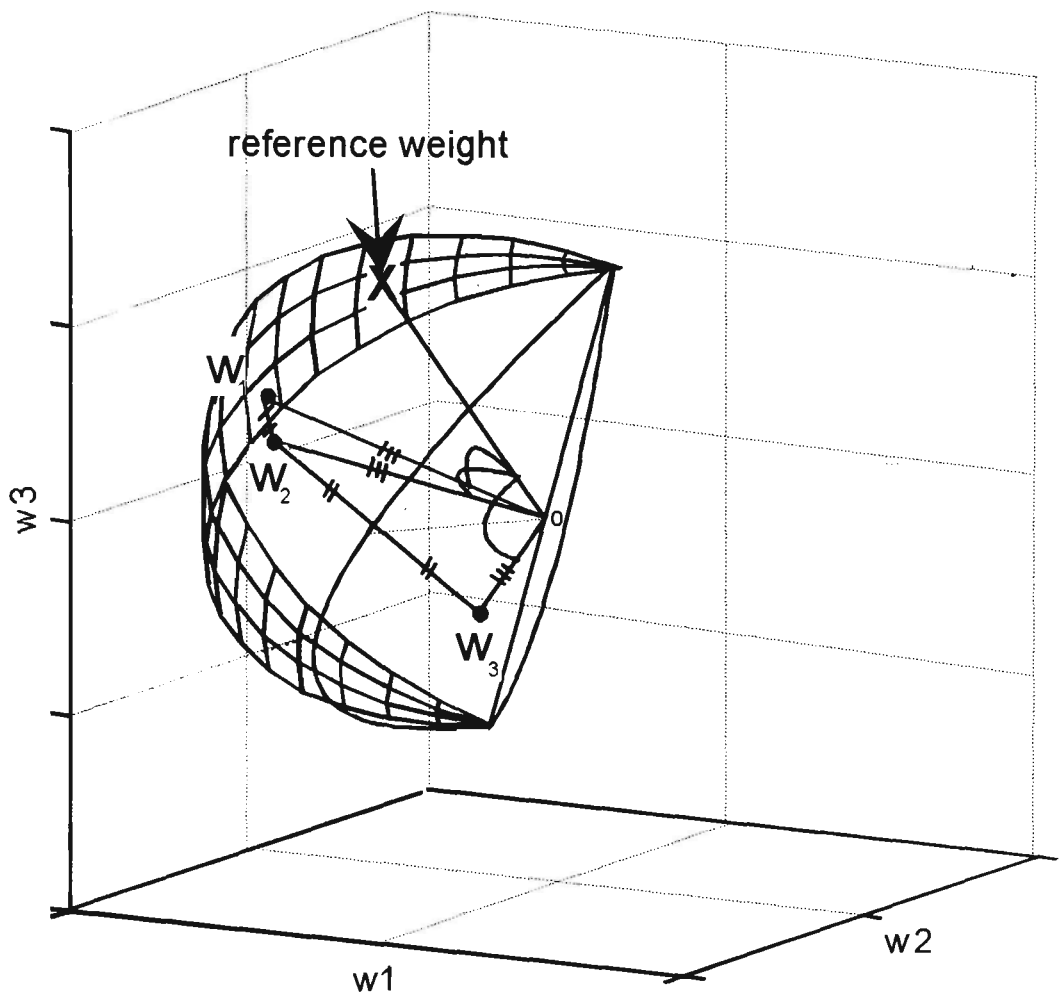


Figure 4.7 : Simplified view of permutation symmetries for example weight vectors **w**. The reference weight vector is denoted by **X** and the solution $\mathbf{w}_1$ lies in its symmetry region. Solutions $\mathbf{w}_2$ and $\mathbf{w}_3$ are symmetric copies of $\mathbf{w}_1$ in the next two symmetric regions. (Figure 4.3 may assist in understanding this diagram).

THEOREM 4.4 : *The minimal angle between a reference weight vector* $\mathbf{w}_r$ *and any other weight vector* $\mathbf{w}$ *occurs if and only if both weight vectors lie in the same symmetry region, assuming that neither vector lies on a symmetry boundary.*

This theorem shows that the global minimum sought by the annealing process, using the error mesured defined by eq. 4.6, will lie in a single symmetry region. When performing experiments, another property is also useful, namely that for any given symmetric weight vector, the angle found is unique to its symmetry region. This means that no two local minima will have exactly the same angle.

THEOREM 4.5 : *The set of angles* Φ *between all symmetric weight vectors of* w_t *and a given reference weight* w_r *contain no equal angles provided that* w_r *is neither equi-distant between any two adjacent symmetry regions nor lies on any symmetry boundary.*

The proofs for these theorems are given in Appendix C.3 and C.4.

4.5.3 Annealing Experiment

To demonstrate the performance of simulated annealing on real networks a number of architectures with two hidden layers were tested. Such networks were chosen because they represent the smallest non-trivial case that allows inter-layer effects to be seen. For the purpose of this experiment, layer one was taken to have twice as many neurons as layer two. Each hidden neuron had the exactly same probability of undergoing a symmetry swap, independent of layer.

The annealing schedule used here is a simple variation on Kirkpatrick et al. [KGV83]. The initial temperature was set to one thousand units and each temperature T is held until ten accepted transitions occur, T is then decayed by 0.9. When one hundred trials produced no accepted transitions then the system is assumed to have converged.

One hundred annealing schedules were run per random weight vector. At the start of each run the same weight vector was placed in different symmetry regions by randomly shuffling all the neurons in each layer. The probability that the simulated annealing process will correctly place the solution vector in the same symmetry region for that architecture can be estimated by determining the number of times the minimal angle is found. Because of the stochastic nature of the results this cycle was repeated ten times for each architecture, using both different reference and solution weight vectors, to obtain a good approximation to the true effectiveness of the annealing schedule. Thus a total of a thousand trials were conducted per architecture. The results over several

architectures can be seen in Figure 4.8. The average time per annealing trial versus the number of symmetries is also given in Figure 4.9.

The assumption made in this experiment is that the probability of obtaining at least one correct solution is very high given enough annealing runs with the same target and reference vectors, but starting the target vector from different random symmetry regions for each run. This assumption can be justified on two grounds. Firstly the annealing conditions are quite rigorous and so will be very likely to converge for reasonable sized architectures, given starting positions in different areas of weight space. Secondly, the probability that the network would become stuck at exactly the same non-local minima when started from completely random initial symmetry regions would be extremely low because of the vast number of possible symmetric solutions. Given that the minimal angle over these cycles is assumed to be the solution that lies in the same symmetry region as the reference weight, it can be inferred that there is only a very small chance of receiving exactly the same angle solution a large number of times. This is especially so if it is not the minimal angle, because the solutions are started from completely different symmetric regions in weight space.

4.5.4 Analysis

The experimental set-up described here was designed to demonstrate both the best and worst case performance for the given annealing schedule. They thus provide a confidence interval for the average probability of successfully decomposing symmetry in a given architecture.

When the annealing procedure is repeated with the target vectors as simply symmetric copies of the reference vector, the algorithm converges with a probability of one even for very large architectures.[5] This demonstrates that it is much easier to anneal to a similar solution vector, presumably because most local minima have been smoothed out or removed. Similarity in the solution vectors in this context means that the angle between two weight vectors when they lie in the same symmetry region is small. This angle can be seen to represent the effective non-symmetrical correlation between these solutions.[6] Thus the experimental set up described here demonstrates close to worst case performance when the vectors are un-correlated and obtains better performance depending on the extent of symmetric correlation between the vectors.

---

[5] This procedure is much faster than the brute force method of checking for equality of the symmetric weight vectors.

[6] Any pair of solutions in the same permutation symmetry region however could still be separated by a maximum of pi radians.

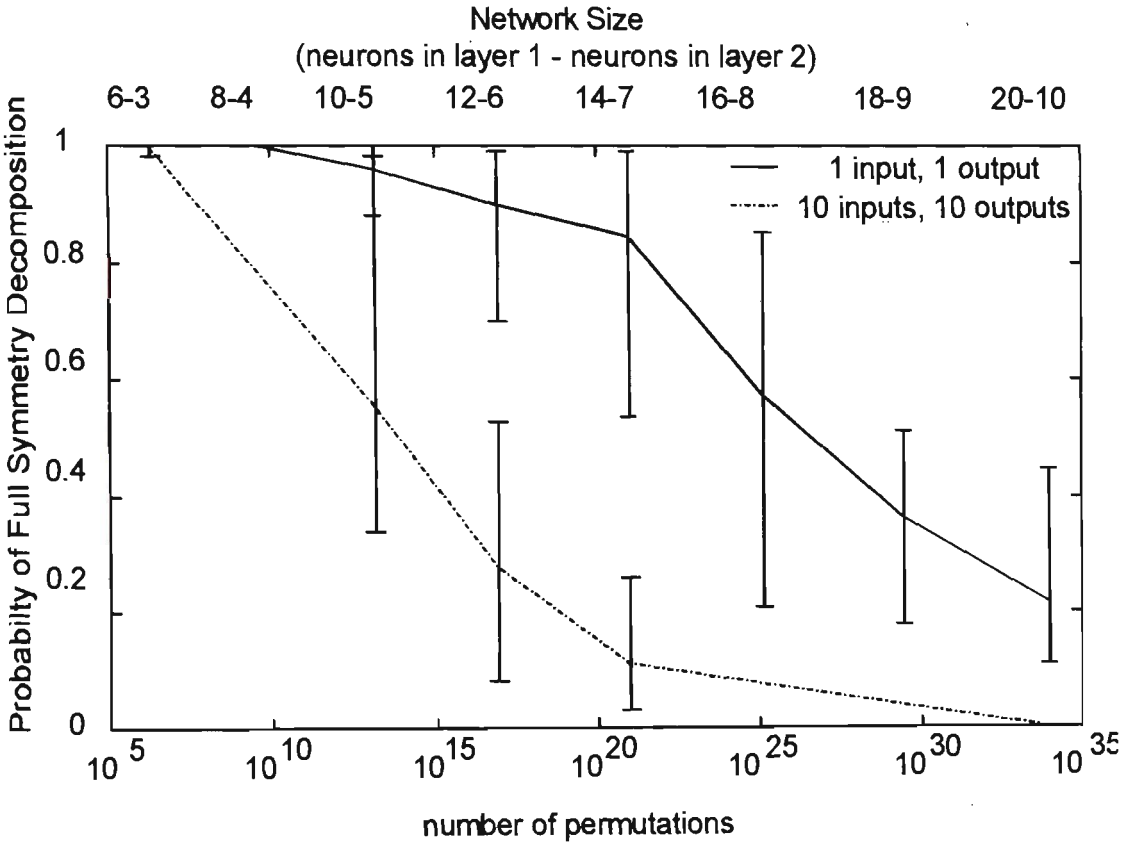


Figure 4.8 : Probabilty of correctly moving a random weight vector into a minimal symmetry region versus the number network symmetries.: To=1000 alpha=0.9 trials=10.

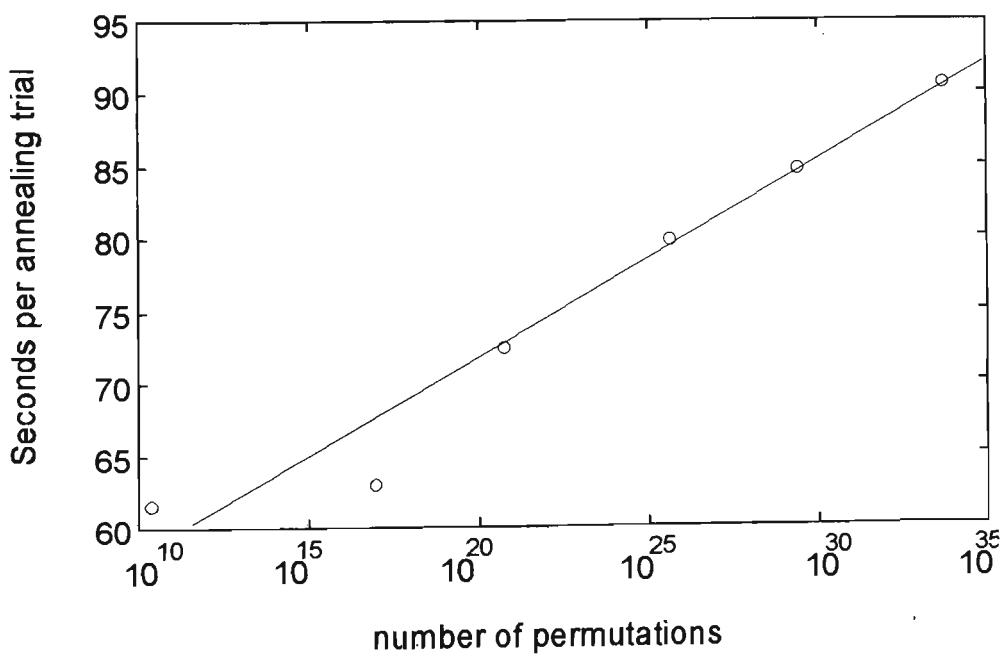


Figure 4.9 : Average time in seconds for a simulated annealing schedule to converge to a solution (using above parameters) verses the number of symmetries in the network.

The plots in figure 4.8 demonstrate the results obtained for two architectures with different numbers of input and output neurons. The shape of these graphs demonstrates that the initial assumptions made for this experiment were reasonable. For very small architectures all the weight symmetries are decomposed with a probability of one. As the networks become larger those solutions that are highly uncorrelated fail to converge on some occasions and so the worst case bound slowly falls while the best case bound remains high. Eventually the best case bound also starts to fall sharply as the algorithm finds it difficult to anneal even highly correlated weight sets. Finally, once the number of neurons becomes large enough, no matter how correlated the weight sets are in their own symmetry region, they are overwhelmed by the huge number of other possible symmetry regions. The larger the network the more rigorous the annealing conditions must be for a high probability of decomposing all the network symmetries. In many practical situations we will be annealing weight vectors for similar types of problems so the best case error bound is more likely than the average probability.

The difference in annealing success as the number of inputs or outputs is increased is due to the extra confusion in weight space caused by an increase in the number of symmetry planes rather than just symmetry boundaries. These planes make the system more susceptible to being trapped in local minima, and consequently mean that the annealing fails earlier. The effect of an increase in the input or output dimensions for a network with a fixed number of neurons is thus to increase the amount of effort required by the annealing processes to ensure all symmetry is broken.

4.6 Practical Examples

Apart from its use in the application of prior knowledge to neural networks, the decomposition of symmetry has another related advantage. It can aid in the interpretation of neural network representations. In chapter 2 the concept of a neural network as black box solution was discussed, this has been brought about by a perception that the internal solutions found by neural networks are very difficult, or perhaps impossible, to interpret. This has limited their application in some safety critical areas where high reliability [PS94] is required, such as in aircraft flight systems or implanted medical devices. In these aplications it is important to be able to rigorously show the function that has been learnt, and under what conditions it may fail. Visualisation techniques such as Hinton diagrams [RM86] and bond diagrams [WT91] show the internal weight states graphically, but it is often still difficult to explain the knowledge that the network has encoded. Even given that an explanation can be constructed using these techniques, it will be specific to both the particular problem and the particular training run. A new explanation must hence be created for each new network trained, even if they encode

similar problems. Symmetry decomposition can allow a much more detailed understanding of the functioning of a neural network by overcoming some of these problems. When the network weight vectors lie in the same symmetry region, neurons in the same position in each network will almost certainly encode similar functionality when trained on the same problem domain. As a result, if an explanation of the internal functioning of one network can be given, then the effect of the neurons in any other solution vector within the same problem domain can also often be determined.

4.6.1 Weight Space Topology and Non-Symmetric Solutions

Symmetry decomposition provides the ability to obtain a much better picture of the topology of weight space. Consider a group of networks all trained on exactly the same problem. When the symmetry in these solutions is decomposed continuous symmetry regions can be deduced from any solutions where the majority of weights are very similar, with a few individual weights that take on a distribution of values. Solutions where the weight vectors differ significantly but the network error is still low, have found an alternate way of encoding the same solution.

To demonstrate these effects, a simple one dimensional function was chosen from those given in Appendix E.1. Of the one hundred training trials, the ten solutions with the lowest mean square errors were selected. The output for these networks can be seen in Figure 4.11. None of the solutions found produced the same output, so the weight vectors reflect parts of solution space that encode similar but not identical functions as in Chapter 2.

Hinton diagrams are used to show the weight vectors for these ten solutions (Figure 4.12a). Each different solution is displayed in a separate column. The Hinton diagrams help to visualise weight space by using squares to represent the weights. The square's size indicates the magnitude of the weight, whilst the sign is indicated by a cross through the square when the weight is positive and no cross if the weight is negative.

The symmetry in these solutions was then decomposed using the annealing schedule presented in Section 4.5.2. This schedule is unnecessarily strict for these small networks with only 120 (5!) permutation symmetries but this ensured that the correct symmetry region was found with a high probability. The reference weight vector was taken from the network that had achieved the lowest error, so all solutions were folded into its symmetry region. The results of this annealing are given in Figure 4.12b. After the annealing, the weight vectors in both weight sets (a) and (b) were reordered on the symmetry broken angle between the reference weight vector and the other weight vectors. Remembering that each corresponding column in (a) and (b) represents exactly

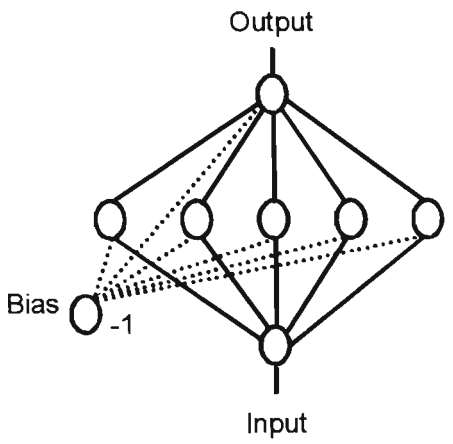


Figure 4.10 : Network used for simple one dimensional approximation

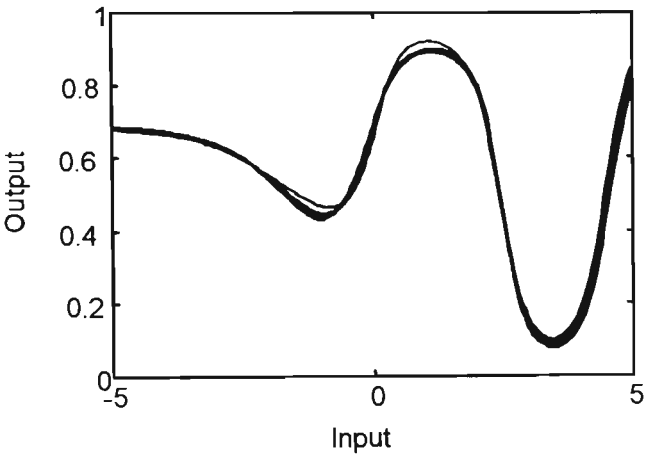
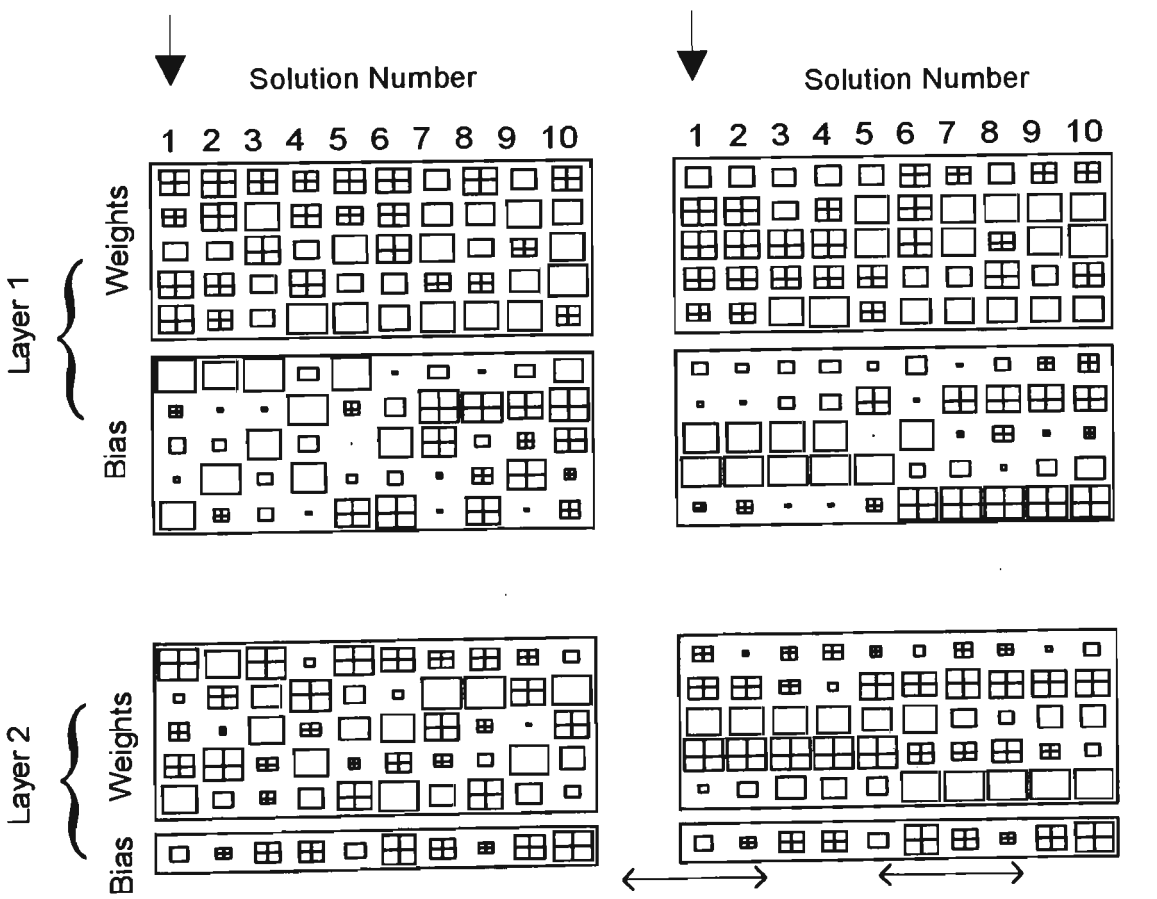


Figure 4.11 : Ten solution outputs found by the network in figure 4.10 to $0.7(1-\sin(5-x)e^{\frac{x-5}{5}})$



(a) Original Network Weights (b) Symmetry Decomposed Weights

Figure 4.12 : Hinton diagram for network weight vectors before and after having symmetry decomposed, ordered on the angle between the first solution and each other solution in the same symmetry region. (Each column represents one solution vector.)

the same solution but in different symmetry regions, it can be immediately seen that the symmetry decomposition has imposed some order upon these weights.

The symmetry decomposed weights seem to be broken into three approximate solution groups. Solutions 1 to 4 are all quite similar as are solutions 7 through 10, with solutions 5 and 6 appearing to be a mixture of these two groups. Weight arrangements like this, where some weight value vary over a range with most of the other weights in the being network fixed, define continuous symmetry regions.

4.6.2 Neuron Output Decomposition

Using the surface learning architecture described in section 3.5.1, the effect of symmetry decomposition on a more complicated network structure can be examined. In contrast to the the previous one dimensional approximation example, these networks can easily be used for the direct visualisation of solutions at a neuronal level, rather than the weight level, and it is at this level that the discrete symmetries are most explicitly represented. Because these networks contain multiple layers and thus a large number of symmetric regions they provide a much more complicated example than given in Section 4.6.1.

The surface learning networks were trained to learn approximations for three completely different digitised images. The network solutions were then annealed with the initial temperature of 2000 and the number of unsuccessful changes required to consider the network converged equal to 200. Each network then had its symmetry decomposed 20 times to ensure that the correct symmetry region was located. These conditions were considered necessary for a high probability of success, given the previously determined probabilities (Figure 4.8) for other networks. The weight vector that encodes the first image was used as the reference weight vector (Figure 4.13).

The first hidden layer shows a lot of correspondence between the different neuron functions despite quite different images being encoded. The line orientations between the different neurons are in general well matched, although there are some cases where it seems that unsuitable choices have been made. In these cases it must be remembered that the visualisation displayed in this form only reflects the fan-in weights from lower layers. As has been discussed, the symmetry on one layer must use the whole of each neuron space, which includes the fan-out of a neuron to the layer above. Thus although a particular combination may appear to be a bad visual match this does not imply that the neuron spaces themselves are badly matched. As would be expected this effect is more noticeable the larger the ratio between fan-out to fan-in weights on a given layer.

The results of hidden layer two are more interesting. Neuron spaces at this level are abstracting higher level "features" of the image itself and thus are much more image

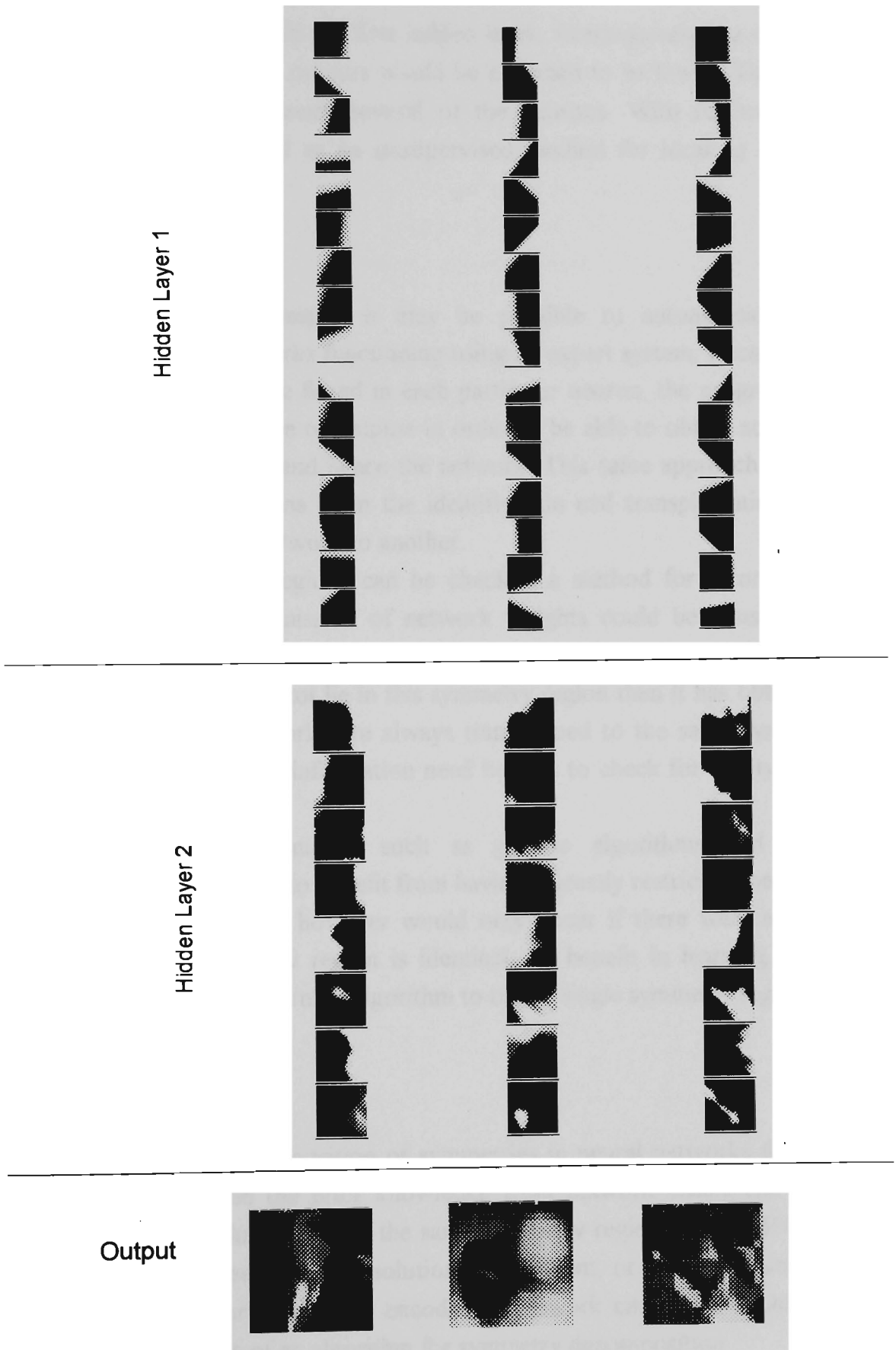


Figure 4.13 : Surface Learning Image Networks with Symmetry Decomposed.

(See Appendix E.3 for training data)

dependent than those found in the first hidden layer. Consequently the correspondence between the different neuron outputs would be expected to be lower. There is however still a good correlation between several of the neurons. With further research this application may prove useful as an unsupervised method for locating and comparing image features.

4.6.3 Other Applications

For some problem domains it may be possible to automatically generate a description of a neural networks functioning using an expert system. Because it is known what sort of solutions will be found in each particular neuron, the program would only need to match a limited range of outputs in order to be able to obtain some idea of the functionality of the neurons and hence the network. This same approach may aid in the location of redundant neurons or in the identification and transplantation of desirable neural functions from one network to another.

Given that symmetry regions can be checked a method for error detection and correction during the transmission of network weights could be constructed around knowing that the weight solution should lie in a given symmetry region. If a weight vector is received and it did not lie in this symmetry region then it has obtained an error during transmission. If networks are always transformed to the same symmetry region before transmission no extra information need be sent to check for this type of error at the other end.

Stochastic search techniques such as genetic algorithms and Monte-Carlo simulations would all appear to benefit from having a greatly restricted space in which to find a solution. This benefit however would only occur if there were a single global minimum. As each symmetry region is identical, no benefit in learning speed can be obtained by restricting the learning algorithm to only a single symmetry region [CLH93].

4.7 Summary

The analysis and decomposition of symmetries in neural networks forms a central part of the ability to use the prior knowledge from network ensembles. Without the ability to ensure that solutions are in the same symmetry region, there will be no weight space consistency between network solutions to different, or even the same, tasks. The optimisation of the prior knowledge encoded in network ensembles could thus not be performed in the absence of an algorithm for symmetry decomposition.

This chapter has introduced the concept of symmetries in weight space, and shown how they may be decomposed. As a prelude to providing a more theoretical grounding,

methods of visualising symmetries in weight space were demonstrated using weight sharing. These visualisations provided the basis for more formal analysis of the mathematical properties of the symmetry boundaries.

This understanding of the symmetries boundaries assisted with the construction of a stochastic algorithm for decomposing permutation symmetries. The effectiveness of this algorithm was shown empirically. It was then tested on two practical problems, one showing consistency in the weight space representations and the other in the neuronal functions.

Chapter 5

Ensemble Optimised Prior Knowledge Transfer

By using the properties of weight space established in the previous chapters, techniques are now developed to assist in transferring the prior knowledge that is contained in ensembles of pre-trained weight vectors to new problems. The simplest way of transferring prior knowledge from an ensemble of trained networks to a new task is to determine the network solution that most closely approximates the target task and then to use this network as the basis for further training. This is effectively just direct or literal transfer and will be referred to as Ensemble Literal Transfer (ELT). This approximation however can be significantly improved by optimisation algorithms based around the mixing of weight vectors in the same symmetry region. The goal of the optimisation is to find the best mixture of the available prior knowledge solutions, for a particular task, in a minimal number of iterations.

Two specific algorithms for optimising the transfer of prior knowledge from ensembles of network solutions are initially proposed, one based on mixing whole networks and the other on neuron substitution. They are tested over a range of parameters using the one dimensional approximation examples given in Appendix E.1. These two algorithms are then shown to be special cases of a third more general optimisation technique. This final algorithm results in significantly improved performance when compared to these specific cases, reducing both the number of iterations and the amount of prior knowledge required to reach a specified error level.

The next chapter provides three more complicated task domains to show the general utility of the optimisation algorithms in reducing average training times and increasing the transfer stability. To ensure a reliable metric for measuring the effectiveness of the transfer algorithm a brief analysis of the computational requirements for the optimisation is undertaken.

5.1 Prior Knowledge Weight Solutions

The volumes in weight space that generalise well on particular solutions were discussed in Chapter 2. In practice however there will only ever be a discrete ensemble of such trained weight vectors about which there is knowledge. Consider m network weight vectors that have been trained on problems such that, $z_l = \varphi(\mathbf{w}_l)$ where $l \in (1..m)$. Let S equal the set of weight vectors such that,

$$S = \bigcup_{i=0}^{m} \mathbf{w}_l . \tag{5.1}$$

The corresponding set of problems P for which this set of weight vectors has solutions is then,

$$P_{z_{(1..m)}} = \bigcup_{i=0}^{m} \varphi(\mathbf{w}_l) . \tag{5.2}$$

Given a new training set z, which may not be directly encoded in P, it is desired to find from S the expected weight vector that will encode the new training set z (such that it is consistent with the prior solutions). Theoretically the expected weight value for any problem f_i can be calculated using the joint conditional probability between the weight space and the problem space $p_{S|f}$,

$$E(S \mid f_i) = \sum_{\text{all j}} w_j p_{S|f_i}(w_j) \tag{5.3}$$

Determining $p_{S|f}$ would be extremely difficult in the general case because of the highly non-linear relationship between the weight vector and the function which it encodes. In addition it would also be difficult to construct a suitably general representation for the training set, as this would need to be of a fixed dimension[1].

5.2 Network Mixtures

Although it may not be possible to calculate the actual expected weight value for a given feature vector, equation 5.3 does suggest optimisation based on mixing weight

[1] One approach to this problem would be to extract the first n principal components of the training set.

vectors to form composite solutions to new problems. In order for ensembles of prior solutions to be mixed it is necessary to show that an approximate form of algebra can be defined on the weight vectors. Consider two problems A and B that generalise well on their respective problem domains, and to which there are corresponding weight vectors $\mathbf{w}_A$ and $\mathbf{w}_B$. If a problem C is now presented that is known to be a mixture of problems A and B, it is desired to find a method to calculate an approximation to the solution vector $\mathbf{w}_C$ for task C.

The direct mixing of weight vectors that are in different symmetry regions makes no sense theoretically as there will be no consistency in the positioning of the weight values, hence one is not mixing like with like. For a practical realisation of this algebra it must be therefore be ensured that, as much as is possible, the functions performed by individual neurons are the same. Chapter 4 demonstrated that by breaking the symmetry in weight space a specific ordering was imposed upon the neurons in a network. If $\mathbf{w}_A$ and $\mathbf{w}_B$ are in the same symmetry region then weight values in the same positions in the vector will be likely to encode similar functionality. This was empirically shown in Figures 4.12 and 4.13.

In this way a simple form of algebra can constructed by using the symmetry broken weights $s(\mathbf{w}_A)$ and $s(\mathbf{w}_B)$. Let problem C have a training set Z_C such that,

$$Z_C = a\, Z_A + (1-a)\, Z_B, \quad 0 \geq a \leq 1 \tag{5.4}$$

then the weights for network C can approximated by,

$$\mathbf{w}_C = a\, s(\mathbf{w}_A) + (1-a)\, s(\mathbf{w}_B). \tag{5.5}$$

Because of the non-linear nature of the activation functions, this linear combination gives only an approximate relationship between $\mathbf{w}_C$ and Z_C. On average the error from the desired Z_C is directly dependent on a (the mixture of the weights). When a is close to either zero or one the error will be smaller, because a weight value that is known to be correct for that solution is dominant. Conversely in the middle when $a = 0.5$, the proposed network solution is at most speculative, so the error may be expected to be a maximum.

$$\text{error}(\varphi(\mathbf{w}_C), Z_C) = \begin{cases} 0 & , a = 0 \\ \max & , a = 0.5 \\ 0 & , a = 1 \end{cases} \tag{5.6}$$

To state this situation in a more general form the following proposition is made (supported empirically by the results in section 5.5.1) ; Given a set of weights $\mathbf{w}_k$ (k=1..m) and a new training set Z_P, that can be determined to be a linear combination of $\varphi(\mathbf{w}_k)$ where,

$$Z_P = \sum_{k=1}^{m} a_k \varphi(\mathbf{w}_k), \tag{5.7}$$

then a reasonable approximation to $\mathbf{w}_P$ can be calculated as,

$$\mathbf{w}_P = \sum_{k=1}^{m} a_k s(\mathbf{w}_k). \tag{5.8}$$

5.2.1 Simple Mixture Example

To demonstrate the performance of this technique, and to support the theorem 5.1, an experiment was conducted using the one dimensional approximation training sets and associated solutions from Appendix E.1. The testing method involved randomly selecting two different problems and their associated weight vectors. The error for different values of a was determined by the mean square error between the desired solution Z_C (eq. 5.5) and the network response to the calculated weight value $\varphi(\mathbf{w}_C)$ (eq. 5.6).

For each pair of solutions two sets of results were calculated. One for weight vectors in the same symmetry region and another for sets of weight vectors that were randomly distributed amongst different symmetry regions. The highest error solution out of the five randomly selected symmetry regions was taken to give a worst case bound. This procedure was repeated one hundred times and the results averaged over all trials (Figure 5.1). Both error curves conformed to the general rule shown in eq. 5.7 and the symmetry broken solutions had significantly reduced error levels over those that were left in random symmetry regions. It is expected that the random symmetry region results would have even higher errors in networks with either more neurons, more layers or bipolar activation functions[2].

[2] With bipolar activation functions the average of two neurons that encode exactly the same function, but lie in different sign symmetric regions, would cancel out rather than being equal (as in a unipolar activation function network).

5.3 Prior Knowledge Transfer Optimisation

Prior knowledge diversity is the amount of variation in output that can be expected from mixing in some way the available weight space prior knowledge provided by the ensemble of network solutions. With a single prior knowledge solution, the prior knowledge diversity is fixed as it is not possible to mix a single solution with itself. The more solutions that are available the greater is the diversity, and so the closer any particular training set can be approximated using only this knowledge. This diversity is not only dependant upon the number of prior solutions stored, but on how well those solutions cover a given problem domain. A large number of solution vectors that approximate only a small subset of a particular problem domain may have less prior knowledge diversity than a few weight vectors that cover that space well. This diversity is clearly limited by the overall functional diversity of the specific neural architecture, as this represents the maximum diversity in representation that can be achieved for a given network independent of the problem.

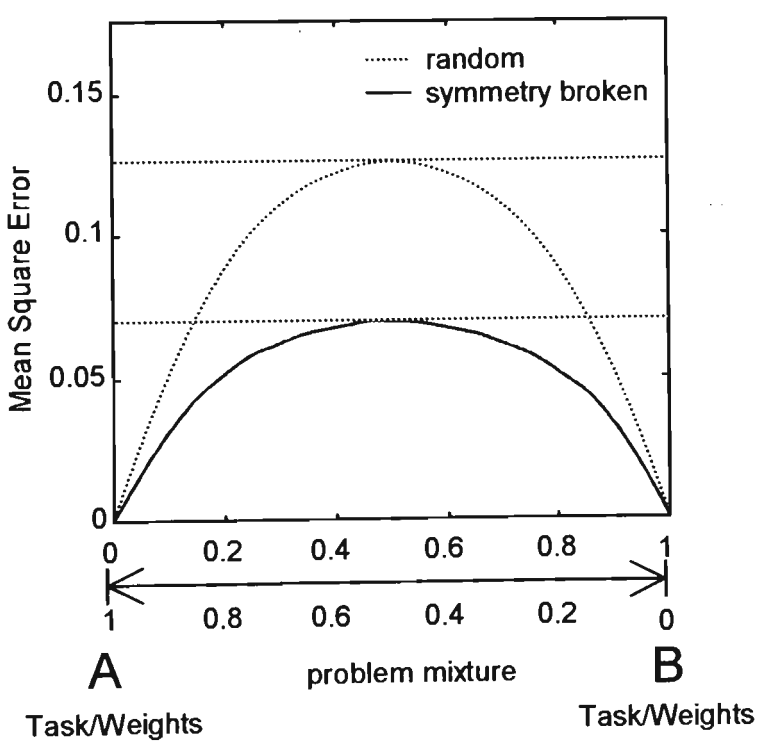


Figure 5.1 : Comparison of the performance of network algebra for approximating two mixed training vectors (A & B) in both symmetry broken and random weight space.

Three different algorithms for mixing these prior solutions are demonstrated by testing over a range of different parameters, again using the one dimensional approximation examples from Appendix E.1. The first two techniques are then shown to be specific cases of the third more general algorithm which, with appropriate parameters, provides significantly improved performance over the other optimisation algorithms.

A desirable characteristic of practical prior knowledge systems is that if the input is a problem which is exactly the same as one to which a solution has previously been encoded in the prior weight solutions, then the output weight vector should be exactly the same as that prior solution. In other words, for known solutions, the system should simply act as a look-up table. Under this provision the simplest algorithm involves direct transfer of the prior knowledge weight vector that has the least error on a given training set (ensemble literal transfer).

5.3.1 Experimental Technique

To examine the performance of the optimisation schemes we are interested in how closely we can approximate a training set (to which an exact solution does not already exist in the prior knowledge) for a fixed number of prior solutions and a fixed number of iterations. Given that one of the aims of the prior knowledge is to reduce training times, excessive computation time for the optimisation will be counter productive.

An approximation to the prior knowledge diversity was achieved by using the mean network response sampled from three hundred random permutations of the nine weight vectors. From these were drawn two disjoint sets; the prior knowledge set and a single problem to be approximated. The one dimensional approximation examples from Appendix E.1 are used again, and can be seen to provide a reasonable coverage of this domain (exponentially decaying sinusoids) given that both the prior knowledge and the solution to be approximated is taken only from this set.

5.3.2 Network Mixtures

To use network mixtures in the transfer of training, it is desired to find a combination of prior network solutions that provides an optimal error output. Starting from the literal transfer configuration, a Nelder-Mead simplex search [DW87] is used on the mixture vector a to attempt to find a good mixture of the prior knowledge in terms of the sum square error. This search technique takes k+1 points from the k-dimensional mixture vector and at each iteration a new point in or near the current volume described by these points is chosen. If the error on of the new network formed from this mixture vector on the given training set is decreased, then this point is accepted as a new vertex. This is repeated until either the diameter of the simplex falls below a threshold or the

maximum number of iterations is exceeded. Note that this algorithm is completely deterministic and will hence produce exactly the same results each time the algorithm is started with the same prior knowledge.

The results of applying the experimental technique outlined in 5.6.1 to this algorithm are shown in figure 5.2. As the functional diversity of the prior knowledge increases the network sum square error decreases roughly negatively exponentially. A significant reduction in error occurs initially as there are correspondingly more degrees of freedom, however the effect of extra prior solutions becomes very small once the solution space is already well covered. This effect occurs with a relatively small number of prior solutions. As more iterations of the optimisation algorithm are used the error also decreases with an negative expediential distribution. Note that with a functional diversity of one the solutions will all have the same error level (ignoring small random fluctuations) because no mixing of solutions can take place. With a single iteration we are essentially looking at a literal transfer situation.

5.3.3 Neuron Substitution

In a symmetry broken network, neurons that are in the same position are likely to encode similar functionality as was demonstrated in chapter 4. Thus it is possible to randomly substitute or transplant neurons, from the prior knowledge solutions, into the same position into the new network in order to find neural functions that provide a better approximation to the new training set. The number of possible combinations of neuron substitutions in a network is given by the total number of network neurons to the power of the number of prior solutions. This quantity increases rapidly with the number of prior solutions available. The maximum number of substitutions for this experiment was 5^8 (3.9×10^5), however the search space can be biased by using *a priori* knowledge about the solutions to determine an appropriate frequency distribution. This distribution is derived by the relative response strength of the individual networks to the new training set z. Thus a network which has low error on a particular training set will have a larger chance of having a neuron transplanted from it than one with a high error,

$$R = [error(\varphi(w_1), z)^{-1}, error(\varphi(w_2), z)^{-1}, \ldots . error(\varphi(w_s), z)^{-1}],$$

where s is the number of solutions in the prior knowledge.

(5.9)
$$freq_z = \frac{R}{\sum_{i=0}^{s} R_i}.$$

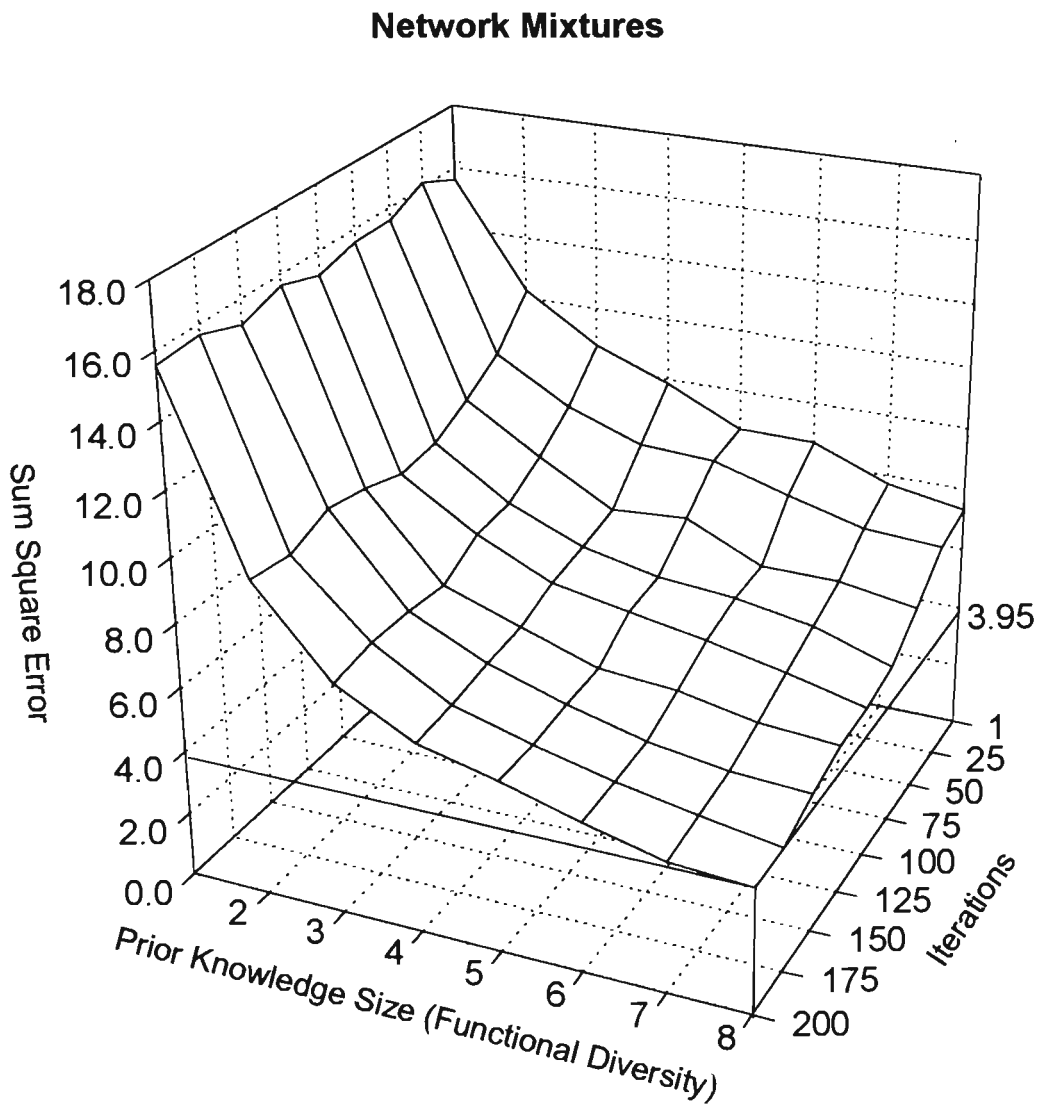


Figure 5.2 : Effect of Network Mixture optimisation on average sum square error as a function of the size of the prior solutions (Functional Diversity) and the number of iterations.

Because the optimisation is required to use a limited number of iterations the system is annealed with a zero temperature. It is pointless to use temperature cooling annealing given only a relatively small number of iterations. Thus, starting with the initial literal transfer network, a random neuron is chosen for replacement from those available. This neuron is then substituted with another neuron, in the same position in one of the other prior solutions. The prior weight vector to use is selected according to the frequency distribution given in eq 5.14. If this substitution results in a lower error then the change is kept. There is always a finite probability of any prior knowledge network contributing a neuron, as error values in R must be finite for well behaved activation functions.

The most noticeable feature of the results for neuron substitution (figure 5.3) in comparison with the network mixtures is that the error drops much more rapidly initially as the number of iterations is increased. The negative exponential nature of the decrease for increasing iterations is thus more apparent. Each iteration requires roughly the same amount of computation in both algorithms because most cycles per iteration are spent computing the new network error.

At two hundred iterations network algebra has a slightly better performance than neuron substitution in terms of reduced error for lower numbers of prior solutions. Neuron substitution is a stochastic technique so there is the potential for a random worst case solution to degrade performance in an individual case whereas the network algebra will always perform equally under equivalent prior knowledge conditions.

Because of the negative exponential nature relating the error to both the increasing prior knowledge diversity, and to increasing number of iterations, a situation is quickly reached where either a significant amount of extra knowledge or number of iterations are required in order to achieve a relatively small drop in overall error. This is important because it means that most of the power of such a prior knowledge system may be obtained relatively quickly and with relatively few prior examples (as long as those examples cover the problem domain well).

5.3.4 Network Fusion

It is now shown that both neuron substitution and network mixtures are specific cases of a more general prior knowledge utilisation algorithm. This algorithm requires arranging the prior knowledge in the form of a three dimensional tensor $\mathcal{T}$. A matrix is required per prior knowledge solution because the network's weight vector needs to be separated into the fan-in weights of the individual neurons, so that the networks may be mixed or 'fused' on a neuronal level, rather than just a network level. Neurons in different layers may have differing numbers of weights so it is necessary to zero pad these vectors such that all the neuron weight vectors are the same length. Thus each network solution

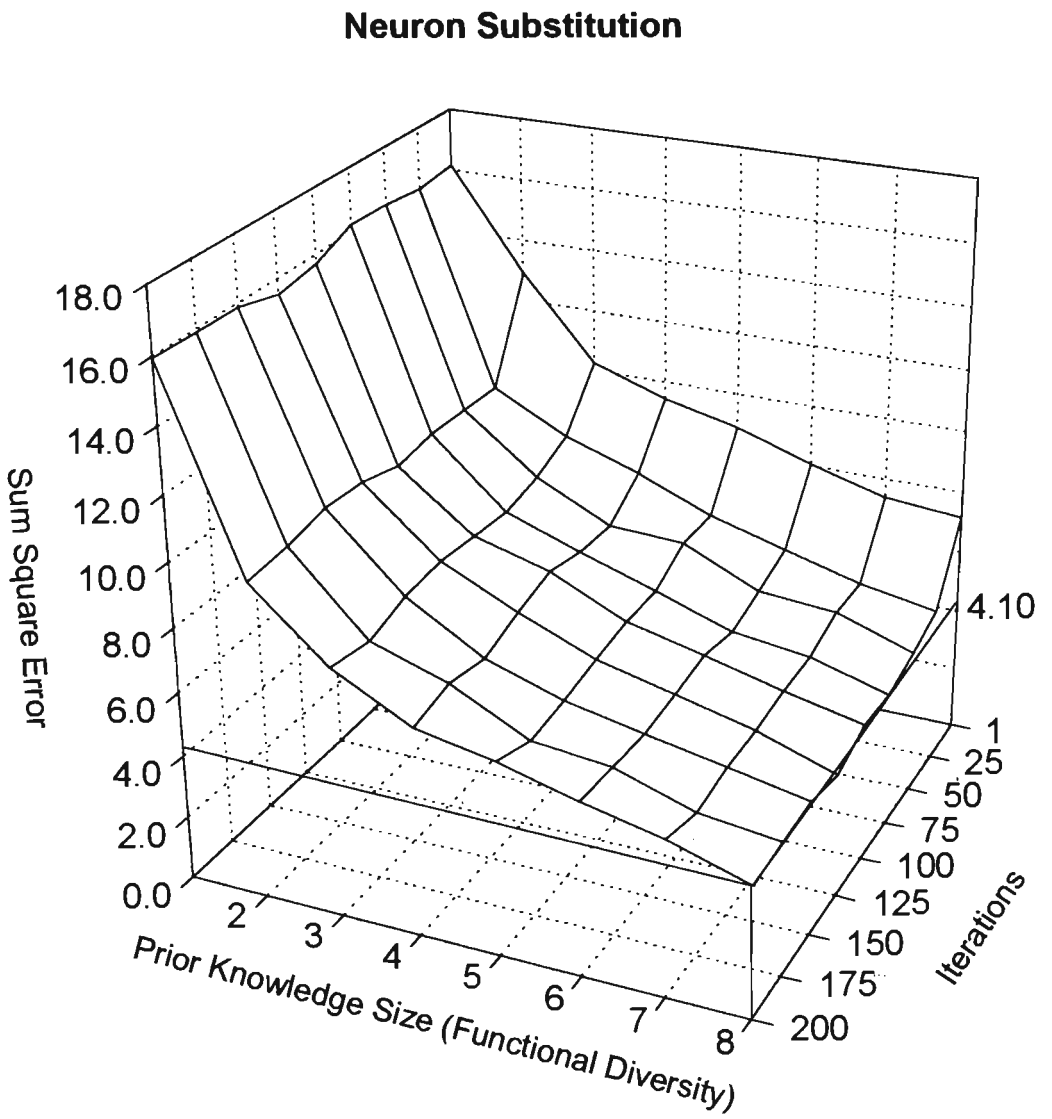


Figure 5.3 : Effect of Neuron Substitution optimisation on average sum square error as a function of the size of the prior solutions (Functional Diversity) and the number of iterations.

is represented by a matrix that has a size of the number of neurons by the maximum of number of fan-in weights to any neuron. The tensor $\mathcal{T}$ is formed by a concatenation of these matrices for each prior solution.

To allow the mixing of the prior knowledge on the neuron level a network fusion matrix is constructed. In this matrix a column exists for each prior knowledge solution p and a row for each neuron in the network n. The elements give the weighting factors for the fan-in weights of each neuron in each prior solution. This matrix thus has a size determined by the number of prior solutions and the total number of neurons. Each neuron column in the fusion matrix must sum to one so that the new solution vector is contained within the volume described by the prior knowledge solutions. The cross product between the prior knowledge tensor and the neuron fusion matrix provides the new solution vector $\mathcal{W}$ (Shown pictorially in figure 5.4).

(5.10) $$\mathcal{W}_{(wn)} = \mathcal{T}_{(wnp)} \times \mathcal{F}_{(pn)}$$

Equation 5.11 can also be written as a summation of matrix cross products, with the fusion vector for an individual network given as a diagonalised matrix.

(5.11) $$\mathcal{W}_{(wn)} = \sum_{p=1}^{\max p} \mathcal{T}_{(wn)}^{(p)} \times \mathrm{diag}(\mathcal{F}_{(n)}^{(p)})$$

It is thus the fusion matrix that requires optimisation. With a large number of prior solutions or neurons this could be almost as difficult as attempting to train the network from scratch. Thus a two stage process is used. In the first stage the fusion matrix is set up so that it mimics the behaviour of the network mixtures (section 5.3.2), and is optimised using a Nelder-Mead simplex search. Stage two uses zero temperature annealing in a similar way to the neuron substitution (section 5.3.3). A neuron is chosen at random and then a prior knowledge solution is selected according to the frequency distribution given by equation 5.10. The selected entry in the fusion table is then increased by some increment i. Finally the values along that neuron column in the fusion table are renormalised so that their total is still one. The combined effect of this is to increase the relevance of a neuron for one prior knowledge solution whilst decreasing the relevance of the that same neuron in other prior knowledge solutions. Appendix D provides a pseudo code implementation of this algorithm.

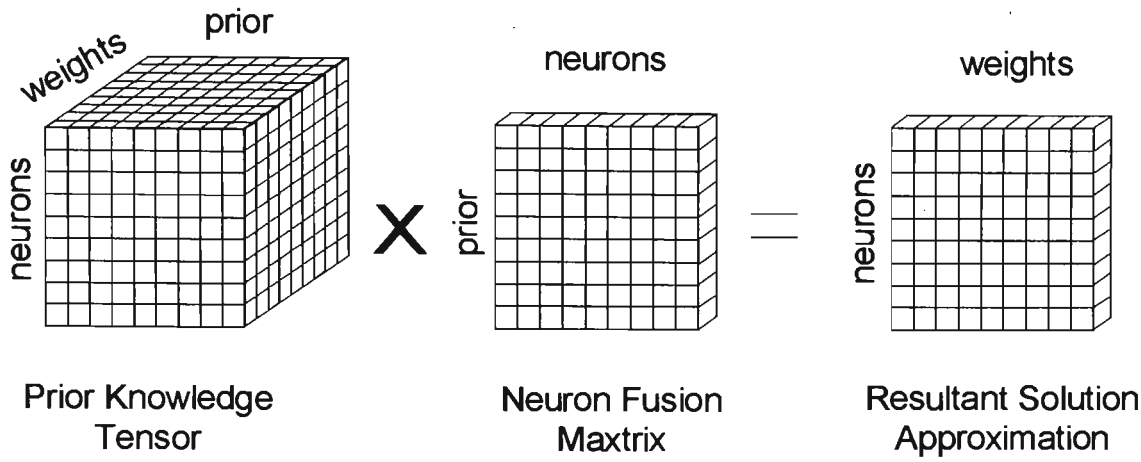


Figure 5.4 : Pictorial view of tensor cross product for Network Fusion Optimisation.

The two earlier algorithms can be seen to be specific cases of the fusion matrix. Network mixtures would have values in the fusion matrix that are the same for each neuron within a prior knowledge solution, because only the whole network can be mixed. Neuron substitution however does not allow for fractional mixtures of neurons, so each neuron can only be a direct copy from one of the prior solutions. Thus the fusion matrix for neuron substitution would only contain binary values.

To test the fusion algorithm the number of iterations were split equally between the two stages and the increment value was set to 0.2. The results demonstrate the error levels decreasing rapidly as the number of prior knowledge solutions is increased (Figure 5.8). More significantly, for iterations greater than one hundred and with a prior knowledge diversity of more than five, the sum squared error is relatively flat (compare with figures 5.5 and 5.6). This shows that the algorithm has been able to find a good solution with these parameters, using only part of the prior knowledge and in relatively few iterations. So it can be seen that, with good prior knowledge diversity, such an algorithm could perform close to the optimum with a limited number of training cycles and stored solutions.

5.3.5 Comparative Analysis

Figures 5.6 and 5.7 provide a direct comparison between the different techniques averaged over 25, 50 and 75 iterations and 150, 175 and 200 iterations, respectively. Neuron substitution performs better than network algebra for a high functional diversity and low number of iterations; however it is slightly worse for a larger number of

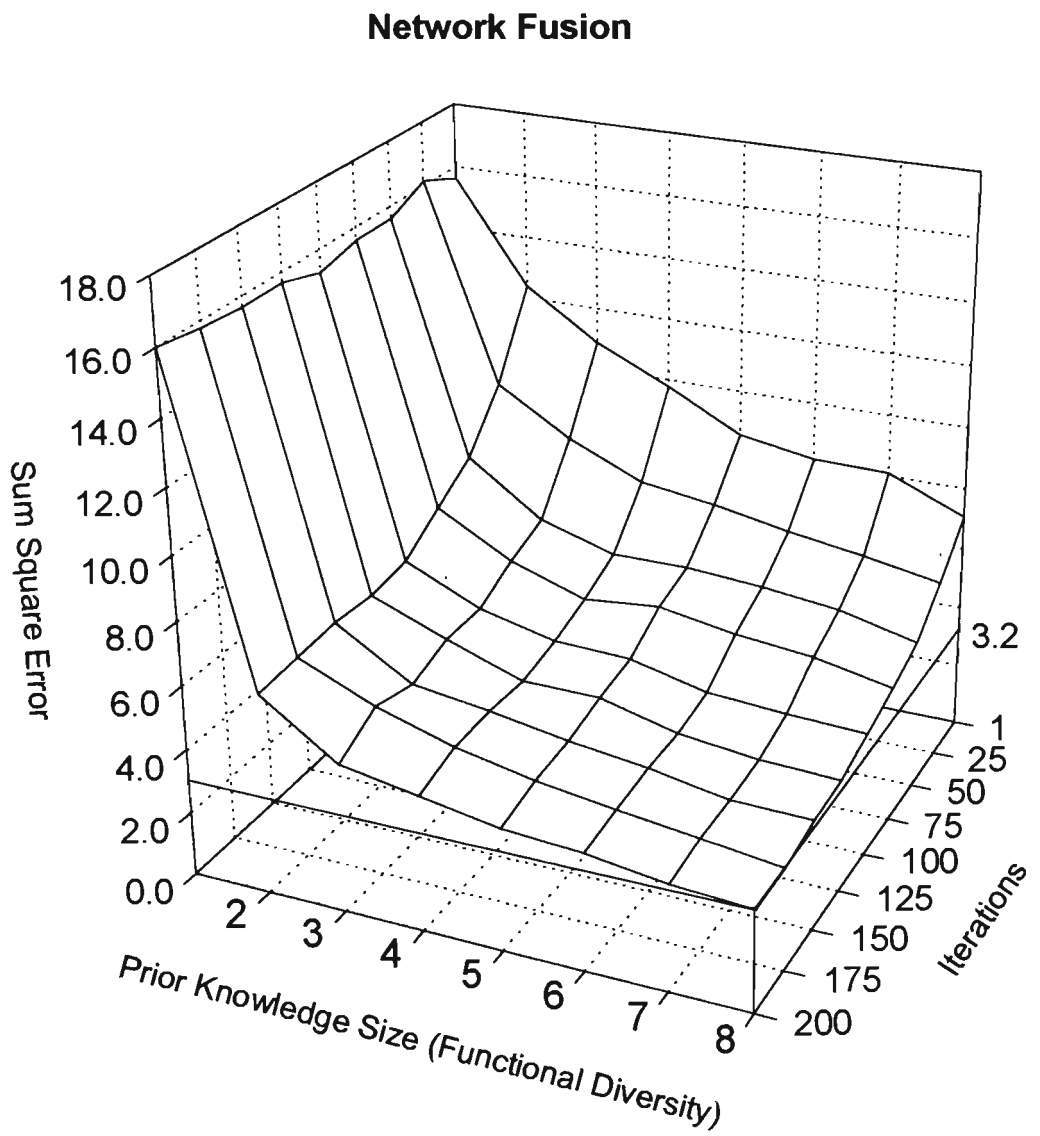


Figure 5.5 : Effect of Network Fusion optimisation on average sum square error with increment set of 0.2 and iterations equally divided between stage one and stage two of the optimisation algorithm. (For odd numbers of iterations training cycles were rounded up)

iterations. Network fusion is substantially better than the other algorithms in almost all cases. The one exception is for a high functional diversity and low number of iterations where neuron substitution is fractionally better. This difference could because with a relatively large number of prior knowledge solutions, and a very limited number of iterations, it is slightly more efficient on average to be transplanting neurons than making incremental adjustments.

For larger networks and more complicated training sets the number of iterations and the amount of functional diversity required to reach an optimal error level will increase, as each extra neuron increases the size of the fusion matrix. The actual size increase is dependant upon the number of prior knowledge solutions in the fusion matrix.

An optimal fit to the prior knowledge however is not usually required. Rather one that provides enough initial assistance to the learning algorithm in order that it can rapidly find a good approximation for the training set. In addition the more complicated the problem domain the more training cycles that will usually be required for the final learning process, and these will usually overwhelm any additional computation that is required for the initial optimisation.

5.4 Transformation Invariant Weight Space

The incorporation of the transformation invariant form into the network fusion algorithm involves ensuring that whenever prior solutions are transferred to or from the prior knowledge, they have the appropriate affine transformations applied to them. Thus, to add a new solution to the prior knowledge data, any detectable transforms in the training data must be corrected for in the weight solutions, such that the stored weight vector is in a normalised form (see Figures 3.2 & 3.3).

When calculating the prior solution to use as the seed for further optimisation in the literal transfer phase, each weight vector must be transformed so that it is consistent with any transforms that are detected in the new training set. This must obviously occur before determining the suitability of a prior knowledge solution for transfer. The optimisation algorithm can of course still be used without the transformation invariant weight space in environments where it is either too difficult, or not appropriate, to attempt the correction for such transformations.

Ensemble Optimised Transfer (EOT) refers the general class of optimisation algorithms that deal with the transfer of prior knowledge. However for the remainder of this thesis it will refer specifically to the network fusion algorithm (Section 5.3.4) plus, where appropriate, the transformation invariant weight space.

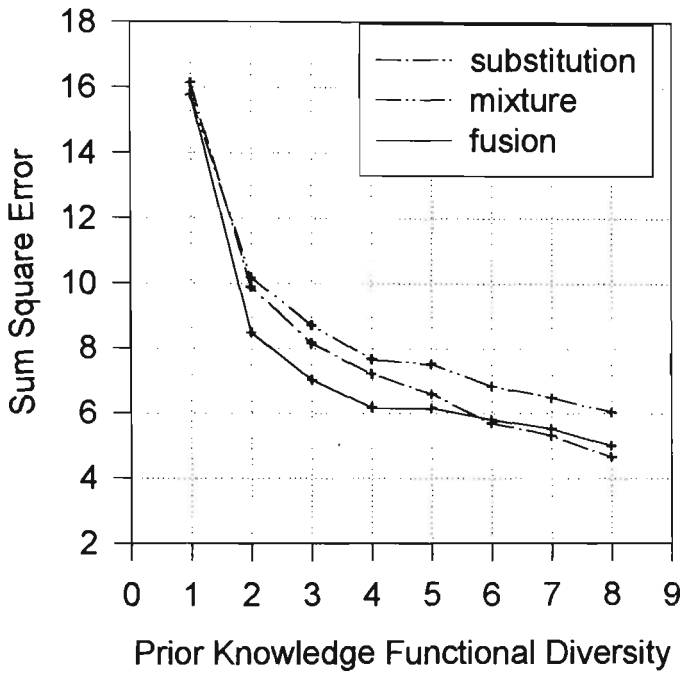


Figure 5.6 : Optimisation Comparision for low iterations (average of 25,50 and 75).

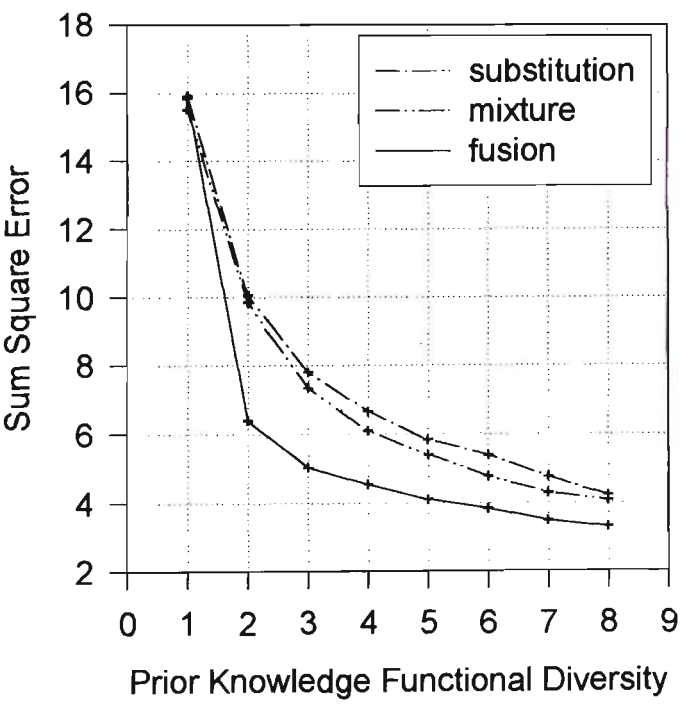


Figure 5.7 : Optimisation Comparision for higher iterations (average of 150,175 and 200).

5.5 Measuring Optimisation Performance

Training times on randomly initialised networks for the same problem vary considerably. Training can become stuck in local minima, find attractors at infinity or crawl along long flat ravines in weight space. Determining statistically significant average training times in non trivial problem domains for randomly initialised networks is difficult because of the size of the weight space. Training times are also hard to measure because of the difficulty of defining when training is complete in situations where it is not possible to obtain perfect classification. For instance one learning algorithm or transfer technique may on average reach a 90 percent error threshold in far fewer iterations than another algorithm but take the same number of iterations, or longer, to reach 95 percent (see [Pra93,Fah88] for a more extensive discussion). For both of the classification applications presented in this thesis perfect recognition on the training set is used as the target goal for the learning algorithm.

Networks initialised with literal transfer also exhibit such variable response, usually appearing to significantly decrease average training speeds over randomly initialised networks but also sometimes increasing training times. Such claims rely on the assumption that a statistically significant sample of the random weight space has been taken when calculating average training times. To avoid some of these problems, no attempt has been made in this thesis to compare the performance of EOT to randomly initialised networks. Improvement factors are determined by a direct comparison to only the literal transfer result. Because the EOT algorithm starts with this same literal transfer situation the performance figures truly reflect the enhancement gained over ensemble literal transfer. Nevertheless it is necessary to average over a large number of solutions because of this sensitivity to initial solutions in the prior knowledge.

5.5.1 Learning Algorithm

Training times are naturally dependent on the particular learning algorithm used. However a heuristic rule of thumb based on empirical experience is that the faster the learning algorithm, the more likely it is to fail to converge. This has been found especially in the more advanced learning algorithms such as Levenberg-Marquardt and Conjugate Gradient. For these techniques many of the training runs are quite short however there are small proportion of solutions that do not seem to converge within any practical number of iterations. These algorithms also do much more work per iteration than simple back-propagation variants and sometimes require a prohibitively high memory overhead for large training sets.

The learning technique used to obtain the results in the next section was an enhanced back-propagation variant with an adaptive learning rate and momentum. The learning rate is varied dependant on the network error, to keep the learning both fast and stable [VMR+88]. Learning was halted when either 100 percent recognition had been achieved or the maximum number of iterations exceeded. The learning algorithm was always used in batch mode, so a single iteration indicates an entire pass through the whole available training set (some literature would refer to this as a *epoch*). To a certain degree the measured performance increases are independent of the learning algorithm, because a faster learning algorithm will produce a speed up in both the literal transfer and in the optimised transfer situations.

5.5.2 Computational Complexity

For each additional network solution that is in the prior knowledge, an extra literal transfer iteration (a forward pass through the training data) is required to determine which of the prior knowledge solutions has the best fit for a presented training set. This occurs for both the optimised and the literal transfer. For small numbers of solutions this is a small overhead when compared with the cost of the full training. But as the number of solutions becomes large this method of selection becomes a brute force search of weight space which will perform much worse than even the slowest gradient decent style learning algorithm. Consequently there is a break even point for any prior knowledge system, where the average cost of finding the best prior knowledge solution is equal to the average cost of using a specified training algorithm from a randomly selected point in weight space. When the number of prior knowledge solutions is less than this threshold, this system gains an advantage over the learning algorithm.

Similarly, when comparing literal transfer to the EOT techniques, the cost of performing the ensemble optimisation in terms of lost iterations must be traded off against the performance gained through using ensemble literal transfer and the chosen learning algorithm for this same number of iterations. An approximation for this is given by assuming that one optimisation cycle requires the same amount of work as one learning cycle. Empirical tests verified that this assumption was reasonable

It has been suggested that a good way of calculating the performance of a transfer algorithm is to use the following metric [JS95],

$$Transfer\ Effect = \frac{T_{inital} - T_{enhancened}}{T_{inital} + T_{enhancened}}. \tag{5.12}$$

The advantage of this equation over calculating a straight speed increase is that it normalises the performance such that the result always lies between minus one (worst possible negative transfer) and one (best possible positive transfer). A value of zero means that the enhancement provided no overall increase in training speed. This equation is necessary because it is possible for the enhanced transfer algorithm to lead to increased training times. The mean speed increase in this case will give a deceptive performance improvement because the negative transfer "speed-up" can only ever be a fractional value whereas the values for strong positive transfers are unbounded. The transfer effect equation weights both positive and negative transfers equally and so is a much more reliable indicator of the performance of a transfer technique. A visualisation of the transfer effect is given in Figure 5.8 where example training iterations for ensemble optimised and literal transfer are given.

This equation has commonly been used to determine the transfer effect where the initial method is random initialisation and the enhanced method is literal transfer [JS95] [SS93]. The initial method in this thesis however is ensemble literal transfer and the enhanced method is EOT transfer. The approximate transfer effect is thus given by,

$$\textit{Ensemble Tranfer Effect} = \frac{I_L - (I_O + I'_L)}{I_L + (I_O + I'_L)}. \tag{5.13}$$

where,

I_L = number of iterations of learning algorithm (with ELT)
I_O = number of iterations for ensemble optimisation (EOT)
I'_L = number of iterations of learning algorithm after optimisation with EOT.

5.5.2.1 Cost Of Adding Solutions To The Prior Knowledge

Adding new solutions to the prior knowledge occurs off-line and so activities such as symmetry breaking don't contribute to the on-line running cost of the optimisation algorithm. On this point it is interesting to note that, if the solution being added was obtained as a result of using the ensemble optimised transfer (containing other symmetry broken prior knowledge solutions), it would be expected to be already close to the correct symmetry region.

Another process that can occur when obtaining new solutions for the prior knowledge is the normalisation of any transformations. If transformation invariant form is being used, the transforms must be detected in the training data and corrected for in the weight space before the solution is stored. It turns out however, for an actual

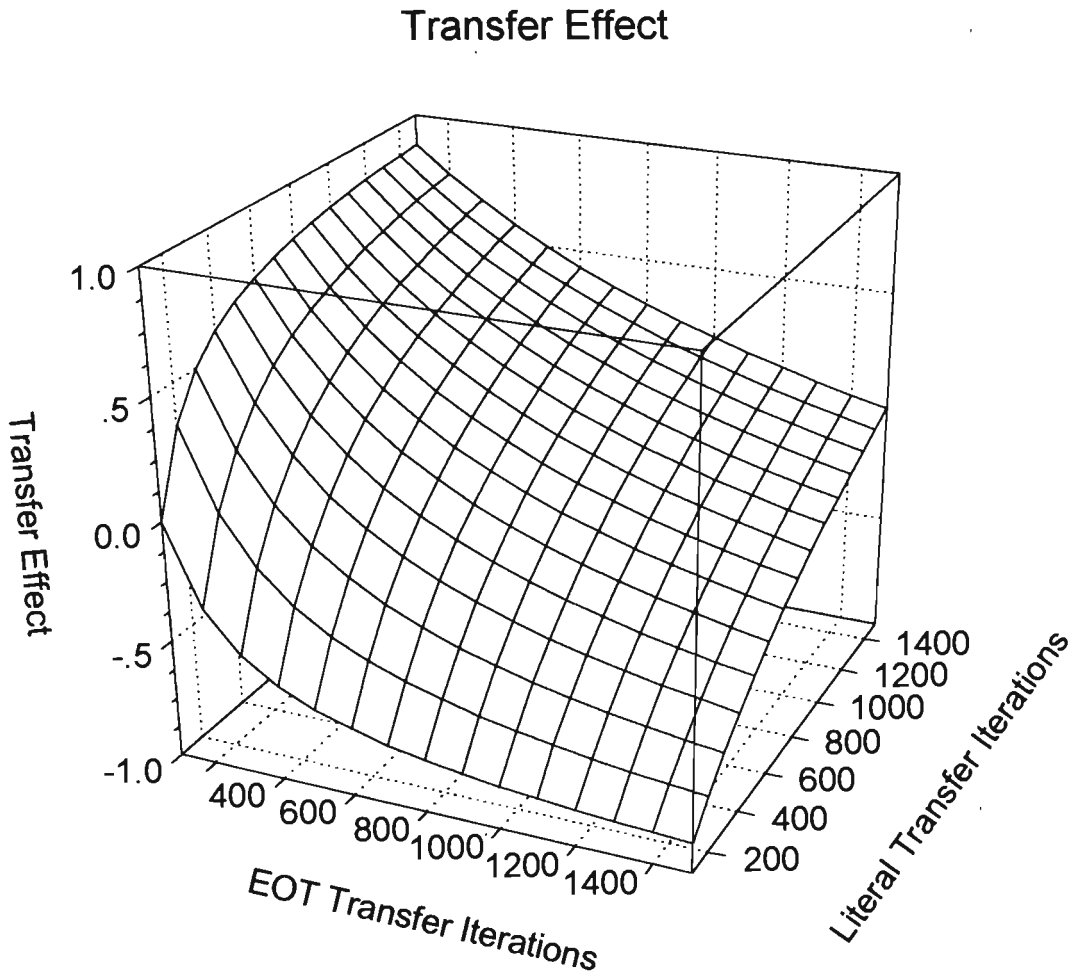


Figure 5.8 : The transfer effect function ploted against sample iterations for the Ensemble Optimised Transfer versus Ensemble Literal Transfer

implementation, that it is both more efficient and accurate to store the solution in its original form along with a tag field that identifies the transforms on its problem domain. When the 'closeness' of a particular prior knowledge solution is being determined, during the literal transfer stage of the EOT algorithm, it can be transformed directly to the correct position, rather that through the normalised form (Figure 3.1 rather than Figure 3.2). This saves on computation at the prior knowledge storage stage and, more importantly, avoids the inaccuracies that can be introduced by transforming the data twice, especially with classification data.

5.6 Connectionist Implementation

It is possible to implement the fusion optimisation algorithm using a connectionist architecture (Figure 5.9). This style of implementation demonstrates a different way of viewing the fusion table. It consists of neuron fusion groups that take the outputs from the weights in each of the prior knowledge neurons and mixes them to form the new weights in the target network. All the weights within the fan-in of a neuron share the same mixture weight (Figure 5.10). The error derived from the network's performance on the new training set is fed back to optimise mixture weights.

This type of architecture could be viewed as a meta-learning network because it uses the knowledge encoded by other networks on new learning tasks. As a speculative idea it is suggested that it would be possible to have groups of networks being trained in parallel with each member of the group connected to the weights of the other members. Learning would progress in bursts of normal training followed by a global sharing of that learnt knowledge amongst the participating networks.

The functioning of the network architecture is directly analogous to that carried out using the tensor multiplication shown in Figure 5.4. Thus the EOT algorithm can be thought of as a constrained neural network learning algorithm. This analogy suggests some possible modifications to the EOT algorithm. For instance the back-propagation of error through the neurons on the new network could be used to determine which of the neurons should have their mixture weights increased.

Similarly techniques used by the more advanced neural network training algorithms such as conjugate gradient [Mol93] may provide some benefits to the optimisation algorithm. A Newtons method type algorithm would require the follow steps. First the direction of the maximum partial derivative of the fusion matrix with respect to the error on the training set (the Hessian) would need to be calculated, although an approximation to this direction would probably be sufficient. A line search would then be conducted in this direction to find the minimum error. These steps are repeated until it is either no longer possible to decrease the error, or the desired error goal is reached. An efficient

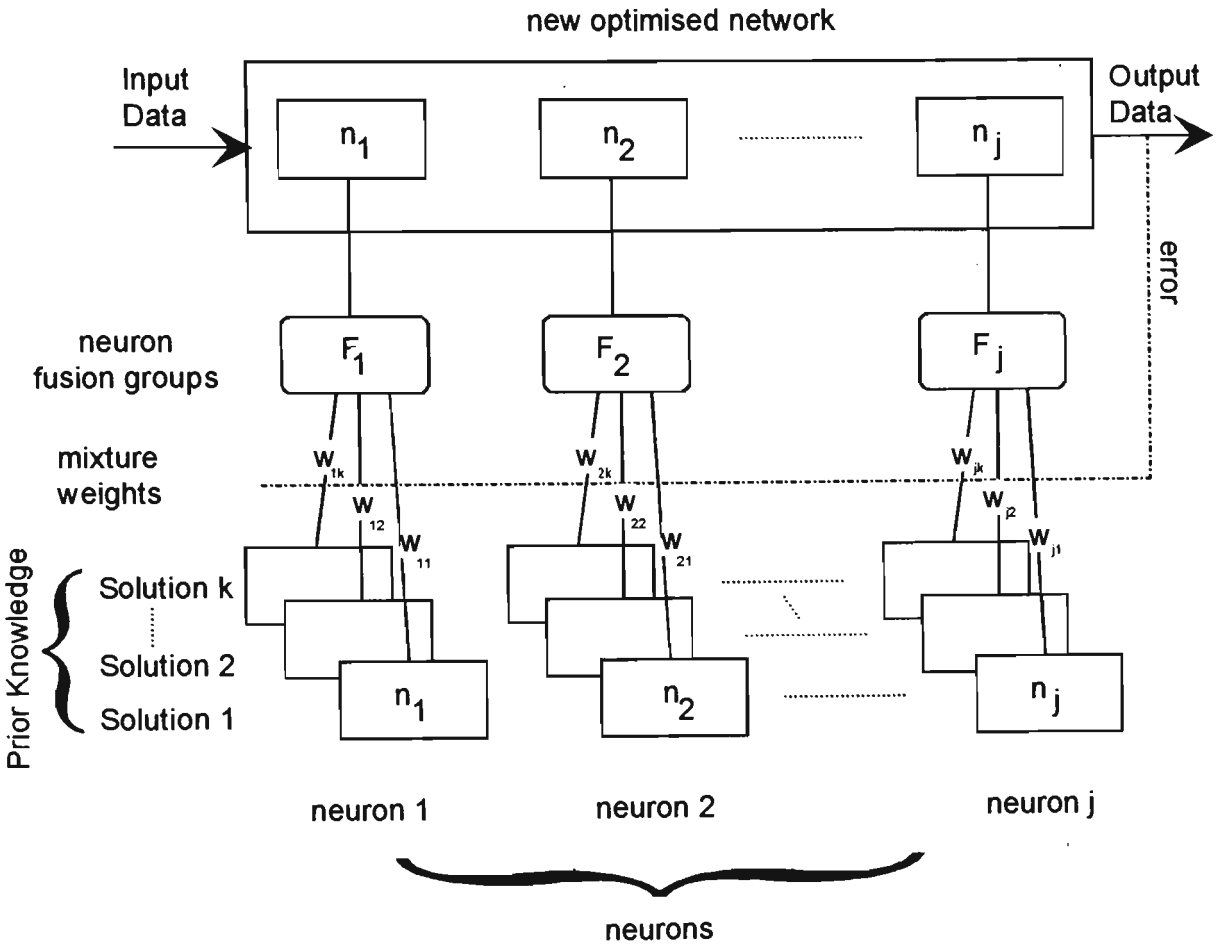


Figure 5.9 : Connectionist Implementation of the Prior Knowledge Fusion Algorithm

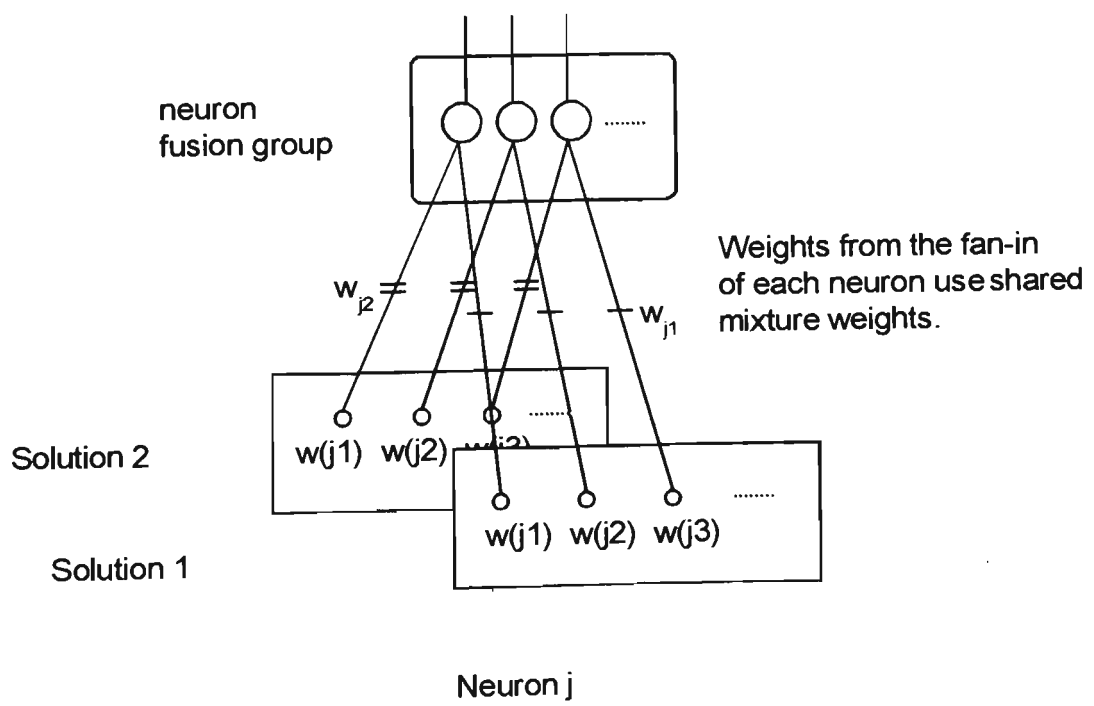


Figure 5.10 : Neuron fusion group internals.

method for computing the partial derivative of the Fusion matrix without perturbing all of the values would need to be derived in order for such an algorithm to be practically useful. With any such scheme however the computational cost of the prior knowledge optimisation must be weighed against the benefits provided over ensemble literal transfer and the chosen learning algorithm.

5.7 Training Data Optimisation

It would appear from the empirical results obtained in this thesis that a good functional diversity can be achieved with a minimal number of solutions. Indeed too many solutions degrade optimisation performance by increasing the size of the optimisation table, thus requiring more processing per iteration. It is suggested that in a more general learning situations, solutions may be pruned when it is determined that the prior knowledge contains sufficient functional diversity. After this size, each time a new solution is added to the prior knowledge, it is checked against the stored solutions. If the existing performance of one or more of the prior solutions is sufficiently similar on the new training data then the new solution should be considered as a replacement. Of course, as emphasised in Appendix A, an empirical estimate may not be a very accurate measure for determining the true similarity of the prior solutions on the new task domain. Ideally we would want to test the new network on all previously seen training sets as well (to determine a more accurate estimate of the true similarity between networks). Given that this would not usually be computationally practical, the suggested alternative is to link the replacement probability to some function that is dependant on the empirical error obtained on the new training data and the number of training samples available.

5.7.1 Validation Sets

It is common practice when training networks to utilise bootstrap type techniques by holding some of the available training data away from the learning procedure as a validation set. It is suggested here that such a strategy is counter productive when using prior knowledge for initialisation as the functional diversity of such systems is clearly already restricted. Although this technique may be useful when determining stopping conditions during learning, the validation set information may play a vital part in selecting the best solution from the prior knowledge.

A validation set is attempting to offset any bias inherent in the representational power of the learner, by determining appropriate stopping conditions for good generalisation. The use of prior knowledge is also attempting to achieve this biasing, although through different means. In addition because the EOT algorithm is used here

for only the initial optimisation there is little chance of it over-fitting the training data. The bootstrap techniques are of course still valuable during the normal network learning phase.

5.8 Summary

This chapter has examined methods of combining the prior knowledge stored in ensembles of networks, to approximate new tasks. Three different techniques were investigated and their approximation ability compared. The general neuron fusion algorithm was shown to give superior performance to the two other algorithms. All of the optimisation algorithms were shown to have a negative exponential relationship with both the size of the prior knowledge, and increasing optimisation iterations. This means that with a good prior knowledge diversity only a relatively small number of prior solutions are required to get close to the limiting behaviour of the algorithms.

Issues to do with measuring the performance of the algorithm were then examined. A metric known as the transfer effect was introduced that measures both positive and negative transfers equally. Finally some extensions to the algorithm were discussed including a connectionist implementation and the possibility of optimising the available prior knowledge based on its functional diversity.

Chapter 6

Experimental Results

The static advantages that can be gained using the ensemble optimisation transfer technique were demonstrated in section 5.3. These results allow confidence in the performance of the optimisation algorithm, however they provide no indication of the learning improvements to be gained using prior knowledge. This is now demonstrated with three different problem environments; image (character recognition - binary inputs / binary outputs), speech (spoken digit identification - continuous input / binary outputs) and surface approximation (image encoding - continuous inputs / continuous outputs). These tasks have been chosen because they represent different types of real world data for which solutions are obtainable using relatively small networks. This allows a large number of training runs to be performed to rigorously test learning improvements. The techniques presented are however not intended to indicate optimal general methods for solving these problems. For general character and speech recognition more sophisticated methods are available that would provide better classification performance under adverse conditions.

6.1 Character Recognition

The character recognition task provides a good set of training data to test the EOT algorithm because the task is non trivial but can be solved using a small number of neurons. This means that the weight symmetry can be easily broken. In addition both the input and output training data is binary, and there are easily understandable relationships between the different training sets.

The 26 upper-case letters from 20 different fonts were chosen (see Appendix E.2). The training data for each character was obtained as 12 point font (but still varied slightly in pixel size) and roughly centred in a 22 by 22 pixel grid. The recognition network had 26 outputs indicating which of the alphabetic characters was present at the input. The training outputs were set at levels 0.1 and 0.9 to avoid the saturation of the sigmoid function that can occur when the output levels are at the sigmoid's limits. The

input weights were then randomly initialised using a normal probability distribution with a mean of zero and a variance of one.

It was found that perfect recognition could be attained for all fonts using only six hidden layer neurons. To obtain a well optimised set of correct solutions, ten randomly initialised training runs were conducted for each of the 20 fonts until convergence was reached, and the ones with the minimum mean square error at this point were saved. Because there were only six hidden layer neurons, a brute force method of symmetry decomposition was used in order to guarantee the networks were in the correct symmetry region[1]. Thus the data used in the experiments below consisted of a training set and an associated symmetry broken solution vector for each of the twenty fonts.

6.1.1 Experiments

The prior knowledge algorithm is competing against the learning algorithm for the number of optimisation cycles it uses. There is thus a trade off between getting a close fit using the available prior knowledge and spending that same computation time performing learning. The optimisation performance was measured against an adaptive back-propagation learning rule initialised using ensemble literal transfer.

The detection of transformations on the new presented training data was performed by placing a bounding box around the mean of the character data using the technique discussed in Section 3.2 (Figure 3.4). For this experiment the transformation invariance algorithm attempted to correct for any shear or translation in the character sets. As can be seen from the training data in Appendix E.2 the character sets are relatively well normalised with respect to these attributes, thus the effect of the transformation invariance in this example would be expected to be small.

For each trial, two random disjoint groups were taken from the examples; one from the stored solutions (for use as the prior knowledge) and the other from the training sets (as the target problem to be approximated). Hence the task to be learned will never be present in the prior knowledge. The ensemble optimisation algorithm was run for fifty iterations followed by normal learning until the network reached perfect classification. The number of iterations for convergence was calculated by adding the number of iterations for the optimisation algorithm to the number of cycles required for the training. If the total number of iterations exceeded 1500 the network was taken not to have converged. Unconverged solutions were accounted for by doubling the training times for any such solutions.

---

[1] 6!=720 symmetry regions

One hundred full training cycles were undertaken for both the literal and optimised transfer over a range of different prior knowledge sizes. Note that for each trial the same prior knowledge set, and the same target task, were used in both the literal and optimised transfer. Hence the transfer effect and speed-up calculated on these individual trials are somewhat independent of the variability in mean training times. There were a total of 2400 complete training runs for this example, as eleven different prior knowledge sizes between two and seventeen were tested. Testing the algorithm on this problem took over a week of dedicated processing time on a Sun Sparc.

6.1.2 Results

Initial testing was conducted with a "full" fusion table that contained an entry for every prior knowledge solution, in the way described in chapter 5. The results obtained using this method however were unexpected. Average training times whilst initially decreasing until a prior knowledge size of three, then actually started to slowly rise with the increasing prior knowledge size until at a size of sixteen where average training times were roughly equal to ELT initialisation. An analysis of the EOT algorithm revealed that as the prior knowledge size increased, the fusion table was becoming too large for the optimisation to obtain a useful solution given only 50 iterations. There are two options to minimise this problem, either increase the optimisation time giving comparatively less time for final training, or decrease the number of prior solutions. The later was chosen as it appeared that an optimum was reached for a prior knowledge size of three with fifty iterations. To achieve this goal, when more than three prior knowledge solutions were available only the best three solutions on the new training set were used for further optimisation (Table 6.1). The algorithm is then parameterized as n–EOT where n is the size of the *minimum error subset* of the solutions selected for optimisation, so ∞–EOT means there is no limit on the fusion table size.

6.1.2.1 Mean training times

Due to the size of the weight space and the complexity of the training data even after one hundred training runs there is still a degree of variability in the mean training times for various numbers of prior solutions (Figure 6.1) however there is enough consistency with neighbouring prior knowledge sizes to be confident about the accuracy of the sampled data. Both transfer techniques appears to have a negative exponential relationships to the prior knowledge size, in a similar way to that shown for the approximation error in section 5.3.1. It would seem reasonable to speculate that there is a relationship between the initial approximation error using the optimisation and the

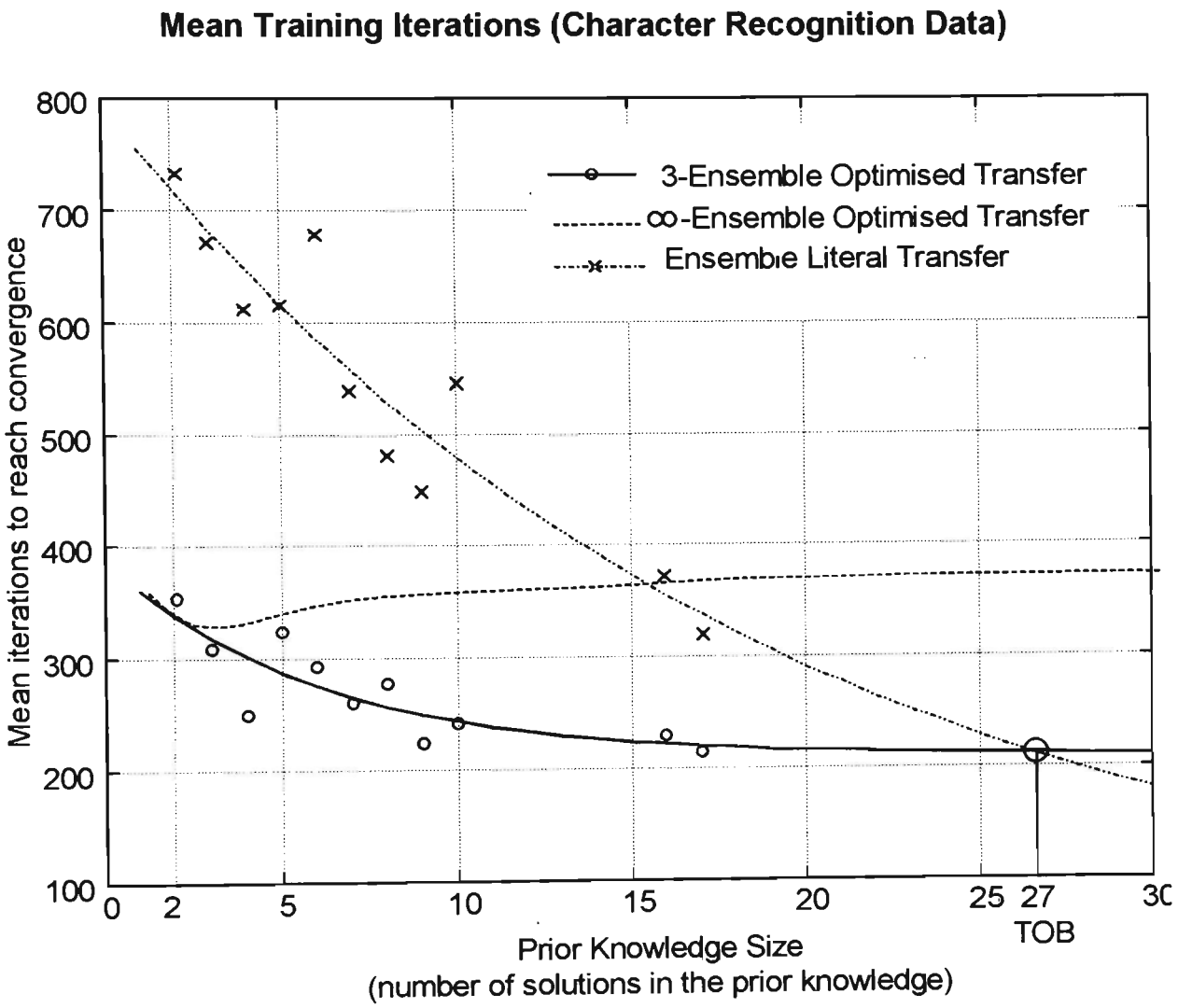


Figure 6.1 : Number of iterations to required reach convergence versus prior knowledge size averaged over one hundred random trials.

	∞–EOT		3-EOT	
prior knowledge size	speed up over Literal-T	transfer effect	speed up over Literal-T	transfer effect
2	2.1007	**0.1817**	2.1007	0.1817
3	**2.1511**	0.1716	2.1511	0.1716
4	1.6384	0.1300	2.4537	**0.1986**
5	1.2581	0.1045	2.5380	0.1594
6	1.0807	0.0311	**2.6694**	0.1560
7	1.1617	0.0651	2.3687	0.1229
8	1.0617	0.0195	2.1626	0.1334
9	1.1498	0.0581	2.1328	0.1284
10	1.0537	0.0123	2.2597	0.1261
16	1.0908	0.0200	1.7375	0.0518
17	0.9920	-0.0255	1.4588	0.0489

Table 6.1 : Table comparing two versions of the neural ensemble optimisation to literal transfer for both the speed-up and the transfer effect. ∞–EOT uses a full fusion table whilst 3-EOT is restricted to optimising the best 3 prior knowledge solutions.

resultant training times, however it will be shown in section 6.3 that this correlation does not occur in all problems domains.

To show the negative exponential relation between the iterations to convergence I, and the size of the of prior knowledge x, the following equation was fitted to the experimental data,

$$I(x) = \alpha e^{-x\beta} + \kappa \tag{6.1}$$

where β is the rate of decay, α is the scaling factor and κ is the minimum iteration limit. The best mean square error fits for both ELT and EOT are superimposed on Figure 6.1. This equation is only valid for x greater than two and much less than infinity. If the prior knowledge size is two or less then the EOT algorithm has no prior knowledge to optimise and so results will be equivalent to literal transfer. As the size of the prior knowledge tends to infinity the number of learning iterations required should go to zero and hence κ must also be zero, but for a practical prior knowledge size the limiting behaviour will require κ be much larger than this.

6.1.2.2 Transfer Optimisation Bound (TOB)

The extrapolations of these graphs clearly show an intersection between the mean training times of the ELT and the EOT algorithms at a prior knowledge size of about twenty seven. As the number of prior solutions increases the ability of the optimisation algorithm to compete against the learning algorithm and literal transfer decreases. The learning algorithm is acquiring better approximations simply through the increased functional diversity of the prior solutions without any optimisation required. The point at which the performance of the optimised transfer is equal to or worse than ensemble literal transfer is called the transfer optimisation bound (TOB). All ensemble prior knowledge optimisation algorithms will have a TOB because, as the functional diversity in the knowledge increases, it becomes increasingly difficult to compete with the learning algorithm. As shall be seen in the next section, the TOB varies depending on the metric which is being used to quantify the optimisation performance. It will also depend on the training data and the efficiency of the learning algorithm. This metric is important because it helps to define the limits of the usefulness of an ensemble optimisation algorithm. The TOB must be an integer value, as it is not possible to have a fractional prior knowledge size, thus the ceiling of the intersection size is always taken.

6.1.2.3 Transfer Effect

As mentioned earlier the transfer effect and speed-up ratios are calculated based on individual training runs with the same solution sets rather than averaged over all the trials, so these results are independent of the actual variability in training speeds shown in Figure 5.14. Indeed it can be seen that using the mean of the training set rather than the individual training data would give an optimistic positive transfer effect, because large variations in training times are averaged out.

Figure 6.2 shows the transfer effect of the ensemble optimisation technique over ensemble literal transfer. As the amount of prior knowledge increases there is a slow dissipation of the optimisations advantage, although the mean training times are still falling (Figure 6.1). One immediate observation about this data is that it appears as if there is a roughly linear relationship between increasing prior knowledge size and decreasing transfer effect. Why this should be so is not obvious until one substitutes the negative exponential equation (eq. 6.1) into the equation for the transfer effect (eq. 5.15). Ignoring the effects of the constant offsets for simplicity the transfer effect equation becomes (subscripts are L for ELT and O for EOT),

$$T_{effect}(x) = \frac{\alpha_L e^{-x\beta_L} - \alpha_O e^{-x\beta_O}}{\alpha_L e^{-x\beta_L} + \alpha_O e^{-x\beta_O}}$$

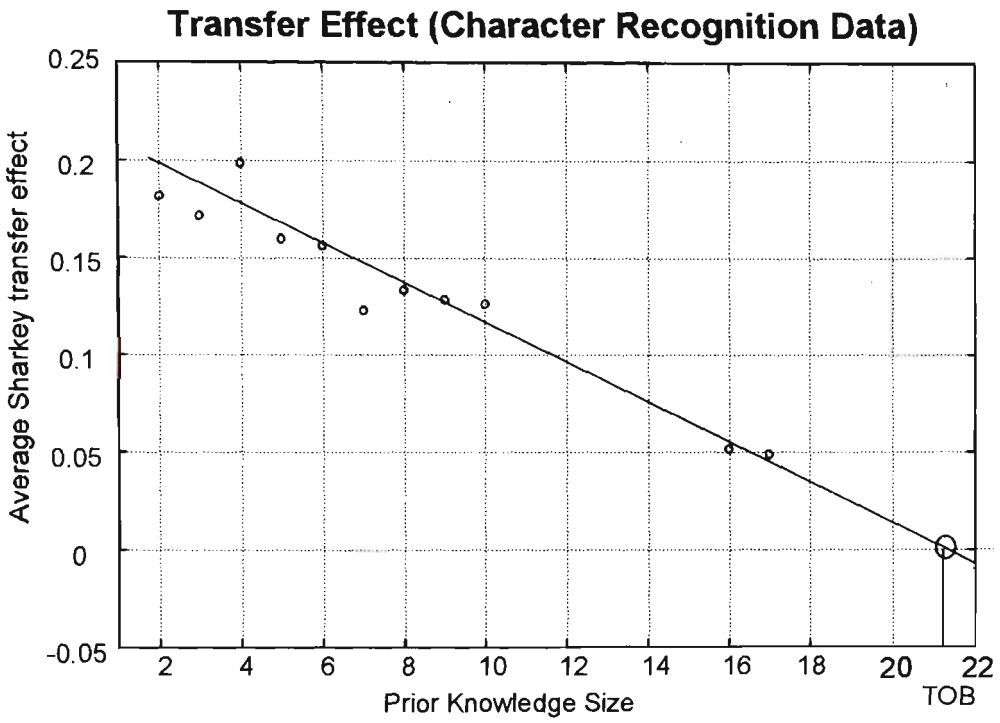


Figure 6.2 : The transfer effect of 3-EOT compared to ensemble literal transfer. Linear extension shows a predicted TOB of 22.

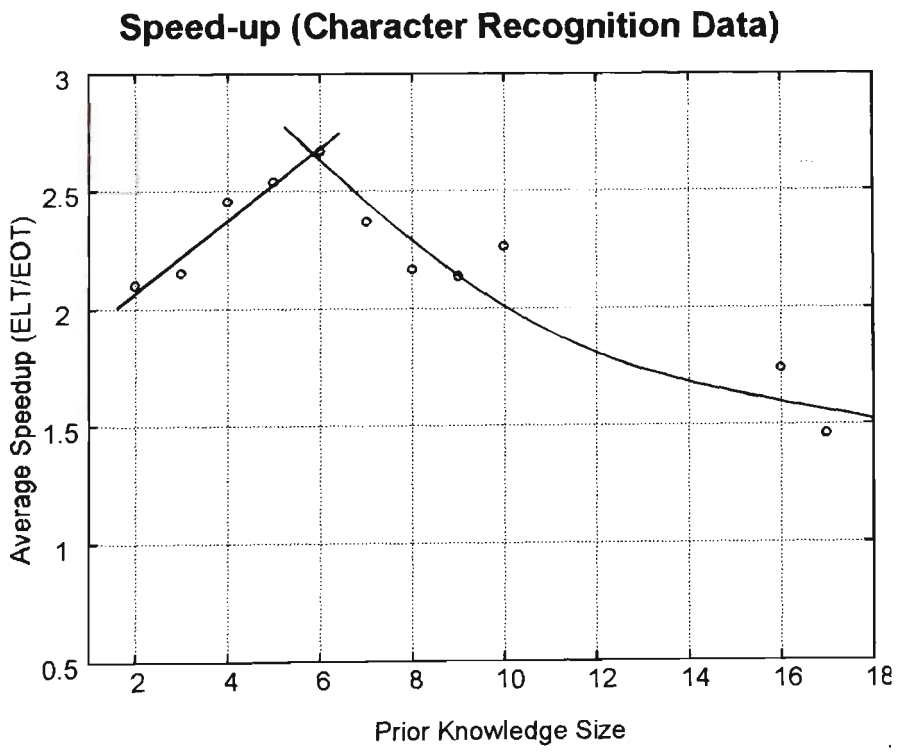


Figure 6.3 : Speed-up of 3-EOT transfer versus literal transfer. The predicted TOB will be much greater than 20.

dividing through by $\alpha_L e^{-x\beta_I}$ gives,

$$= \frac{1 - \frac{\alpha_O}{\alpha_L} e^{-x(\beta_O - \beta_I)}}{1 + \frac{\alpha_O}{\alpha_L} e^{-x(\beta_O - \beta_I)}}$$

$$= \frac{1}{1 + \frac{\alpha_O}{\alpha_L} e^{-x(\beta_O - \beta_I)}} - \frac{\frac{\alpha_O}{\alpha_L} e^{-x(\beta_O - \beta_I)}}{1 + \frac{\alpha_O}{\alpha_L} e^{-x(\beta_O - \beta_I)}}$$

then simplifying the second term,

$$= \frac{1}{1 + \frac{\alpha_O}{\alpha_L} e^{-x(\beta_O - \beta_I)}} - \frac{1}{1 + \frac{\alpha_L}{\alpha_O} e^{x(\beta_O - \beta_I)}} \tag{6.2}$$

When B_O is less than B_L then $(B_O - B_L) < 0$. Thus as $x \to \infty$ the first term tends to zero and the second to one. Thus the slope will be negative. While $x > 1$ and the magnitude of $x(B_O - B_L)$ is small then the magnitude of both terms grows at roughly the same rate thus,

$$T_{effect}(x) \propto -x \tag{6.3}$$

Using this relation to fit the observed data, the TOB is roughly twenty one, as compared with twenty seven calculated using the mean training times.

6.1.2.4 Speed up

By way of comparison Figure 6.3 shows the average network speed up calculated as simply,

$$Speed\ up = \frac{T_{literal}}{T_{optimised}} \approx \frac{I_l}{I_o + I_l}.$$

These results show that a maximum average speed-up of 2.7 was gained when there is a prior knowledge size of six. If we are to use the same criteria as for the transfer effect to evaluate the speed-up as a function of x, a simple substitution from equation 6.1 would yield a negative exponential decrease. This does not however match the observed data

until a prior knowledge size of six, which shows a roughly linearly speed increase. This effect may be due to the inadequacy of the speed-up metric for measuring empirical data. As described in Section 5.5.2 the scaling which is done by the transfer measure provides a more reliable estimate than the straight speed-up measure. The peak positive transfer speed-up that occurred was during training was 25.9 times however some literal transfer training times were also shorter than EOT-3 leading to a peak negative transfer of about 15 times. The TOB for the speed up (when the curve crosses one) would obviously be very large.

6.1.2.5 Cumulative Convergence

Another view of this data is provided by examining the fraction of solutions that converged in less than a specified number of iterations (Figure 6.4). It can be seen from this graph that all networks took more than 80 iterations to converge for the literal transfer and 110 iterations tor 3-EOT. However the convergence of EOT is very unlikely to occur in less than fifty iterations because this is the time that is spent optimising the prior solutions. The fraction of solutions decreases very quickly past this point and breaks even with literal transfer at 150 iterations. From here on the optimised transfer has a lower fraction of the total solutions, and by 500 iterations it is roughly a fifth of the literal transfer results. The fraction of networks still unconverged at 1500 iterations indicates the total number of unconverged solutions. Literal transfer had about 7% of the total solutions un-converged, whilst 3-EOT had just less than 1%.

6.1.2.6 Stability

One result that is quite clear from Figure 6.4 is that the optimised transfer is significantly more stable than the literal transfer. The literal transfer graph is quite rough and has many irregular drops at various numbers of iterations. These drops correspond to situations where most networks have had few improvements in the learning speed for the intervening number of iterations. This indicates that as the iterations are increasing the solutions become limited by the local topology of weight space, for instance becoming stuck in local minima. The unstable nature of literal transfer has been noted before [Pra93] and these results provide an emphatic confirmation of this nature. The ensemble optimised transfer by contrast has a relatively smooth cumulative convergence curve.

6.2 Digit Recognition Example

To provide another application from a different task domain, the transfer of training from networks trained to classify speech was chosen. Nine speakers taken from the

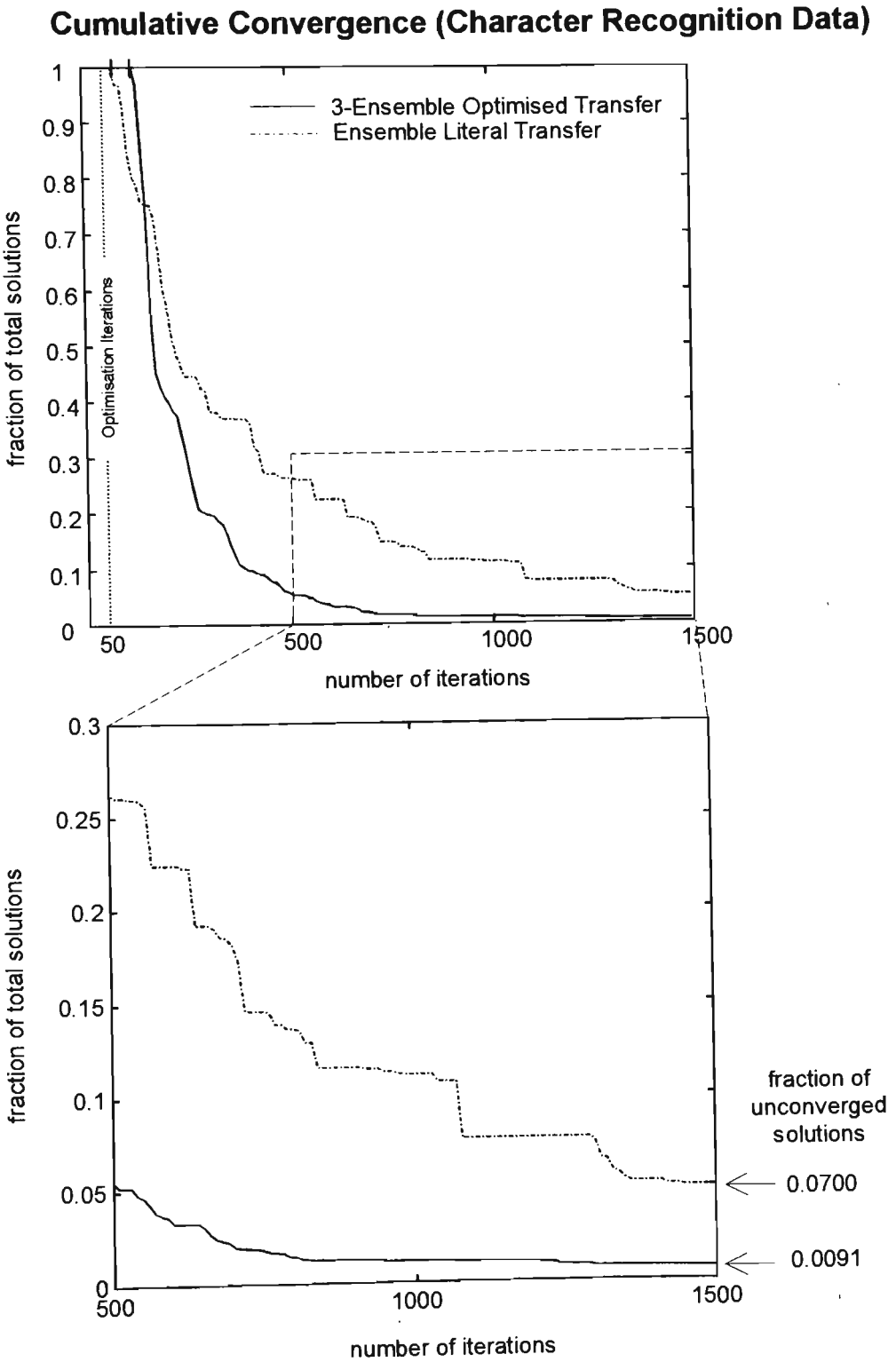


Figure 6.4 : The cumulative fraction of networks that had converged to one hundred percent recognition in less than a given number of iterations. Training was stopped at 1500 iterations so any networks that reached this point represent unconverged networks.

DARPA TIMIT isolated digit database were used to train networks to recognise the spoken numbers from "zero" to "nine". The speakers selected from this database were all North American males but each sounded distinguishably different from the others.

The input to the networks was the spectragram of the speech data. The one dimensional speech data was converted to a two dimensional signal in which each column contained an estimate of the short-term, time-localised frequency content. The log of the spectrogram was taken and then normalised to lie between zero and one. The inputs were thus continuous and there were with ten binary outputs (one for each of the spoken digits). A crude version of dynamic time warping was done by detecting the start and end of the word using a simple energy thresholding technique and then re-scaling the resultant vector so that the input data contained only 40 time and 40 frequency intervals. This data was fed into a back-propagation network with eight hidden units and ten outputs.

6.2.1 Experiments

Training was conducted as for the character recognition data except that the symmetry was broken using the simulated annealing technique. All the final solutions classified their respective training data perfectly, however only one sample word per digit was used, so the generalisation ability was not tested. This technique is only intended as a test of the transfer of training, and does not necessarily represent a good general method for recognising spoken words, although it is the basis upon which many more sophisticated algorithms are built.

The experiments for this data were conducted for 50 random trials in the same manner as the character recognition trials. No attempt however was made to correct for affine transforms[2] in the data, so the results here are derived solely from the fusion table optimisation. Once again an upper limit on the size of the fusion table was placed at three.

6.2.2 Results

Figure 6.5 shows a negative exponential relation between the prior knowledge size and the number of iterations to reach convergence. In comparison to the character recognition data however the mean number of iterations to reach convergence was much higher for all prior knowledge sizes. This indicates that finding a perfect solution to the

[2] [PA92] contains some research on detecting and correcting affine transforms in speech data.

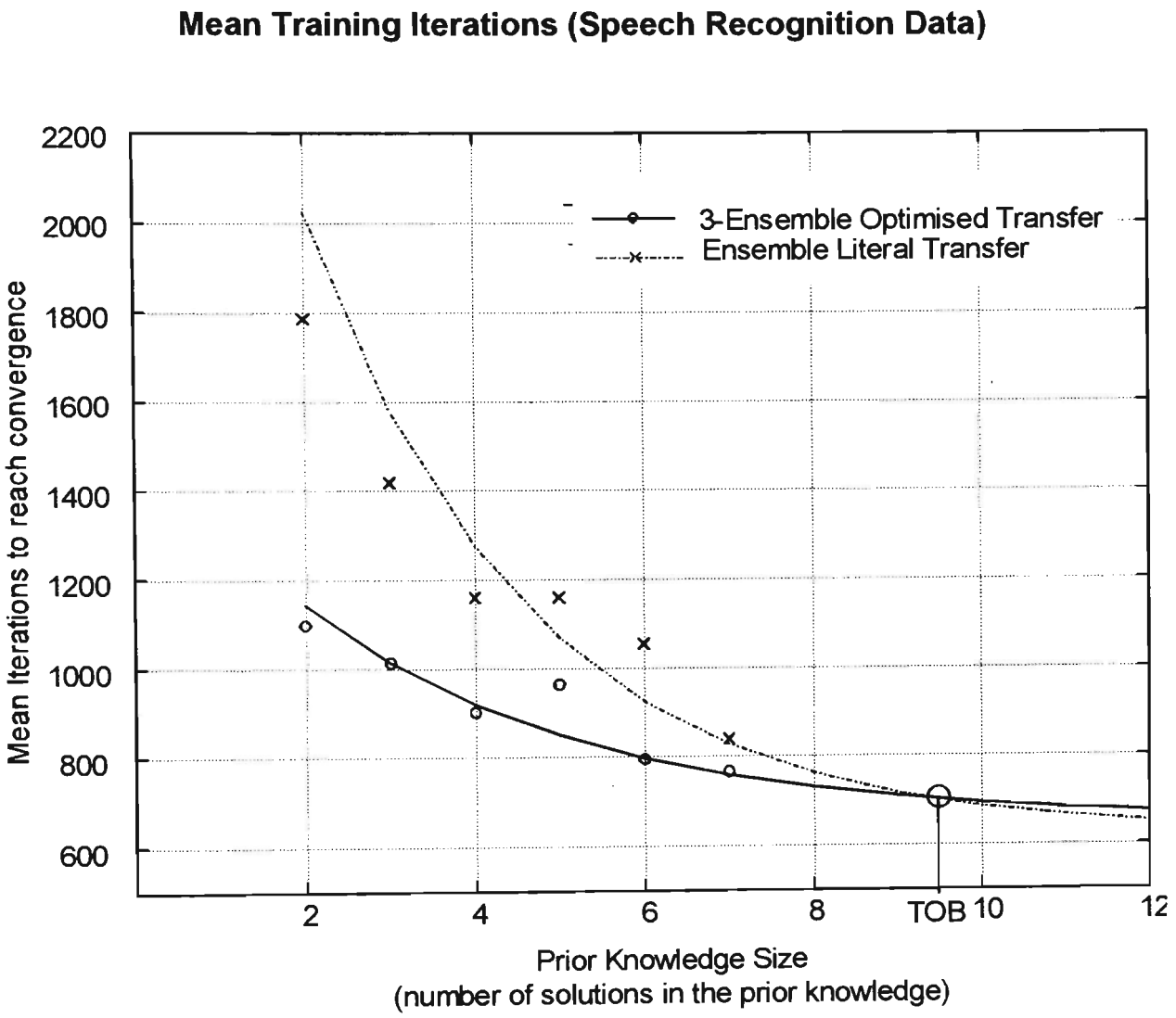


Figure 6.5 : Mean number of iterations to required to reach convergence versus prior knowledge size for the speech recognition data.

	3-EOT	
prior knowledge size	speed up over Literal-T	Sharkey transfer effect
2	**2.7958**	**0.1785**
3	2.6056	0.0670
4	1.6288	-0.0303
5	1.9312	-0.0305
6	1.5019	-0.0516
7	0.9177	-0.1270

Table 6.2 : Table comparing two versions of the neural ensemble optimisation to literal transfer for both the speed-up and the Sharkey transfer effect

speaker recognition problems was more difficult. One reason for this is that there is less regularity (Section 3.1.1) in the speech environment than the binary character environment, which of course leads to an decreased ability to transfer. In addition, because the entire set of prior solutions to draw on was smaller the estimation of general transfer ability is probably less accurate. The EOT solution at a prior knowledge size of five looks like an irregular outlier, and was not used in the data fitting for the negative exponential. The estimated TOB from this data was ten.

6.2.2.1 Transfer Effect

The transfer effect on this data set is not as strong over extended prior knowledge sizes as for the character recognition data (Figure 6.6 & Table 6.2). Note that the maximum speed-up and transfer effects are however quite similar in both environments (compare tables 6.1 and 6.2). This data also illustrates the discrepancy between the calculations based on the mean training times, versus the average calculation over individual training runs. If the values from figure 6.5 had been used directly, all the transfer effects would have been positive. As it is, the transfer effect has a calculated TOB of only four. This is likely to be a more accurate indication of an ideal TOB for this problem than that of nine indicated by the mean training times. It can be seen that the linear fit indicated by relation 6.3 still approximates this transfer data quite closely.

6.2.2.2 Speed-up

The speed-up data more closely approximates the predicted negative exponential distribution (Figure 6.7) than in the character recognition problem. The results here also graphically show that the speed-up results can be deceptive. Well after the transfer

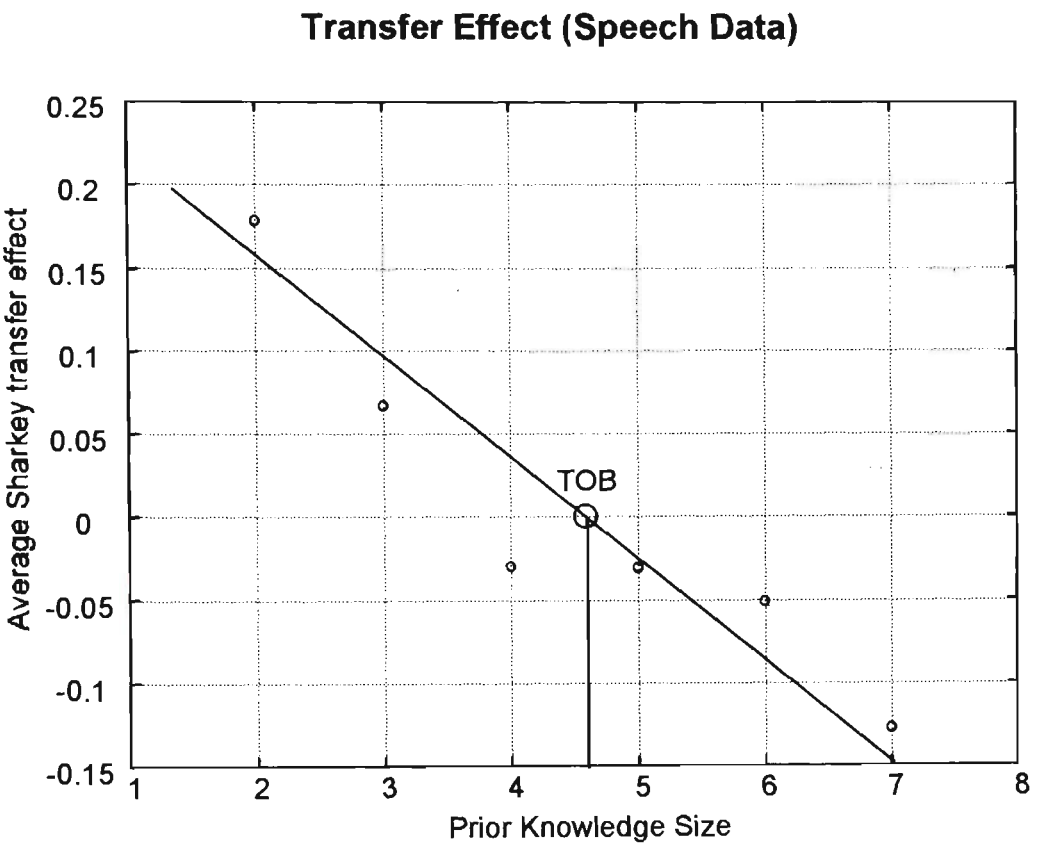


Figure 6.6 Transfer effect of 3-EOT versus ensemble literal transfer

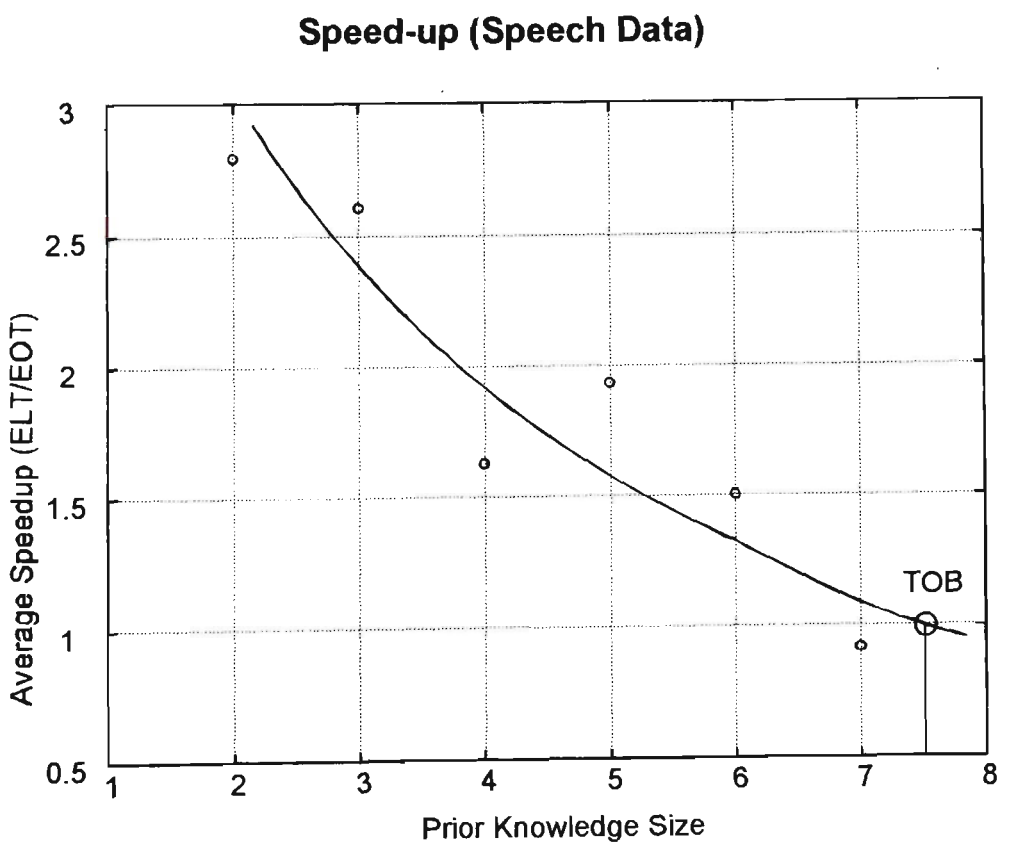


Figure 6.7 Speed-up of 3-EOT versus ensemble literal transfer

results are negative the speed-up is still above one. An observed TOB of about seven is noted, which interestingly lies almost exactly between the other two estimated TOB values.

6.2.2.3 Cumulative Convergence

An indication of the increased complexity of the training data is given by the fact that a much higher percentage of the solutions were un-converged (Figure 6.8). For Ensemble Literal Transfer just over 30 percent of solutions did not converge where as it was 18 percent for EOT. The ratio of unconverged ELT to unconverged EOT solutions is much higher than for the character recognition data indicating that the EOT algorithm had more difficulty determining an optimal mixture because of either increased neurons (eight versus six) or the amount of information that was transferable across the tasks was lower (as suggested earlier). In combination, both factors would provide less for the optimisation algorithm to optimise as less data was shared. The fusion table was also slightly larger so there would have also been less efficient optimisation in only 50 iterations.

6.3 Image Encoding Example

For the final set of experimental results, training is transferred between the image surface learning networks demonstrated in Chapters 3 and 4. This problem is quite different from either the digit recognition or speech recognition problems as the task is approximation rather than classification. This means that those transfer techniques that rely on class determination (such as DBT) would be of limited use in this type of task environment. Some other distinguishing features of this problem are two hidden layers and a much larger number of neurons (16 in the first layer and 8 in the second). There are much fewer overall weights however, because there are only two input neurons and one output neuron. Both the input and output data are continuous variables.

6.3.1 Experiments

The network architecture and technique given in Chapter 3 were used to learn surface approximations to ten different images scaled to be of size 42 pixels square (Appendix E.3). Hence each training set has 1764 training examples, one for each pixel. The prior knowledge set was constructed by taking the lowest error solutions out of ten training runs on each image. The symmetry in the trained networks was then broken using simulated annealing as described in Section 4.5.2 (Figure 4.13 shows the results of three of the networks used to encode images). Estimation of the training effect using

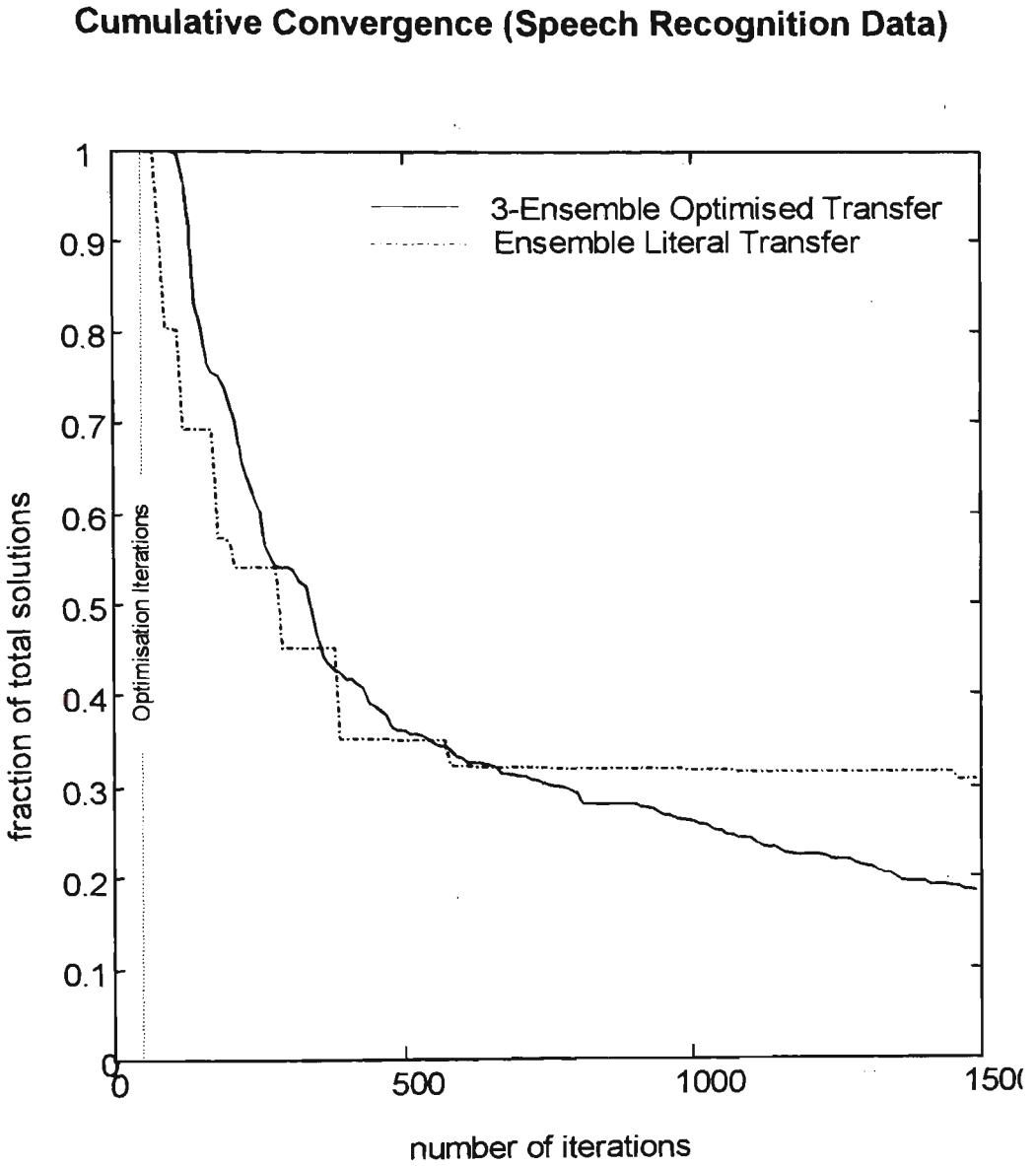


Figure 6.8 : The cumulative fraction of networks that had converged to one perfect recognition in less than a given number of iterations.

EOT was done by examining the performance of the network after a fixed number of iterations.

It is difficult for the EOT algorithm to provide a significant boost over the performance of ELT in this problem domain. When the ratio of neurons to weights is high the learning algorithm will tend to initially learn quite rapidly from the directly transferred solutions, as compared with spending equivalent cycles in the optimisation algorithm. Because of the increased number of neurons there is also a much larger fusion table to optimise requiring added optimisation iterations.

In addition, any advantage in optimisation is further eroded as the training data is complicated enough that the network can never get a very close approximation. Thus there are many weight representations that will approximate the training data to a similar accuracy. For the optimisation to provide a practical benefit in this environment the algorithm must use the bare minimum number of iterations. The zero temperature annealing performed by the standard EOT algorithm is thus not feasible, especially as the increased number of neurons means the size of the fusion table is much larger.

The method chosen to achieve this rapid optimisation is to conduct a line search between the two lowest error solutions from the prior knowledge. It was found that a restricted search along the line through weight space defined by these solutions will often yield a lower error than the minimal error prior knowledge solution. Figure 6.9 shows some typical error curves that were generated when approximating different images using the prior knowledge. The true minima tend to lie close to the lowest error (given by mixture value of one) however quite significant reductions could often be obtained with minimal searching. Note that minima occur not only between the two solutions, but also sometimes on the extension of that line (Figure 6.9 (c,f)). Algorithmically this extension means that the larger error solution is being subtracted from the minimal error solution to keep the mixture total one.

To graphically show what occurs during the mixing of the solutions, the network outputs have been displayed at regular intervals along the line between two solutions (Figure 6.10 (a,b)). Visually the effect is similar to the familiar graphical "morphing" often seen in computer generated special effects. The algorithm relies on one of the interpolated images being a closer fit to the desired training set than the lowest error prior knowledge solution.

The search algorithm first requires the calculation of the derivative at the lowest error solution (mixture value of one), as this almost always indicates which direction the desired minima lies in. Next, six points at intervals of 0.1 are sampled in that direction and the minimal value found is taken as the new network solution. More sophisticated Newtons method or golden section search algorithm's were found to be inappropriate, because there are sometimes local minima (Figure 6.9 (e)) in the error surface. The level

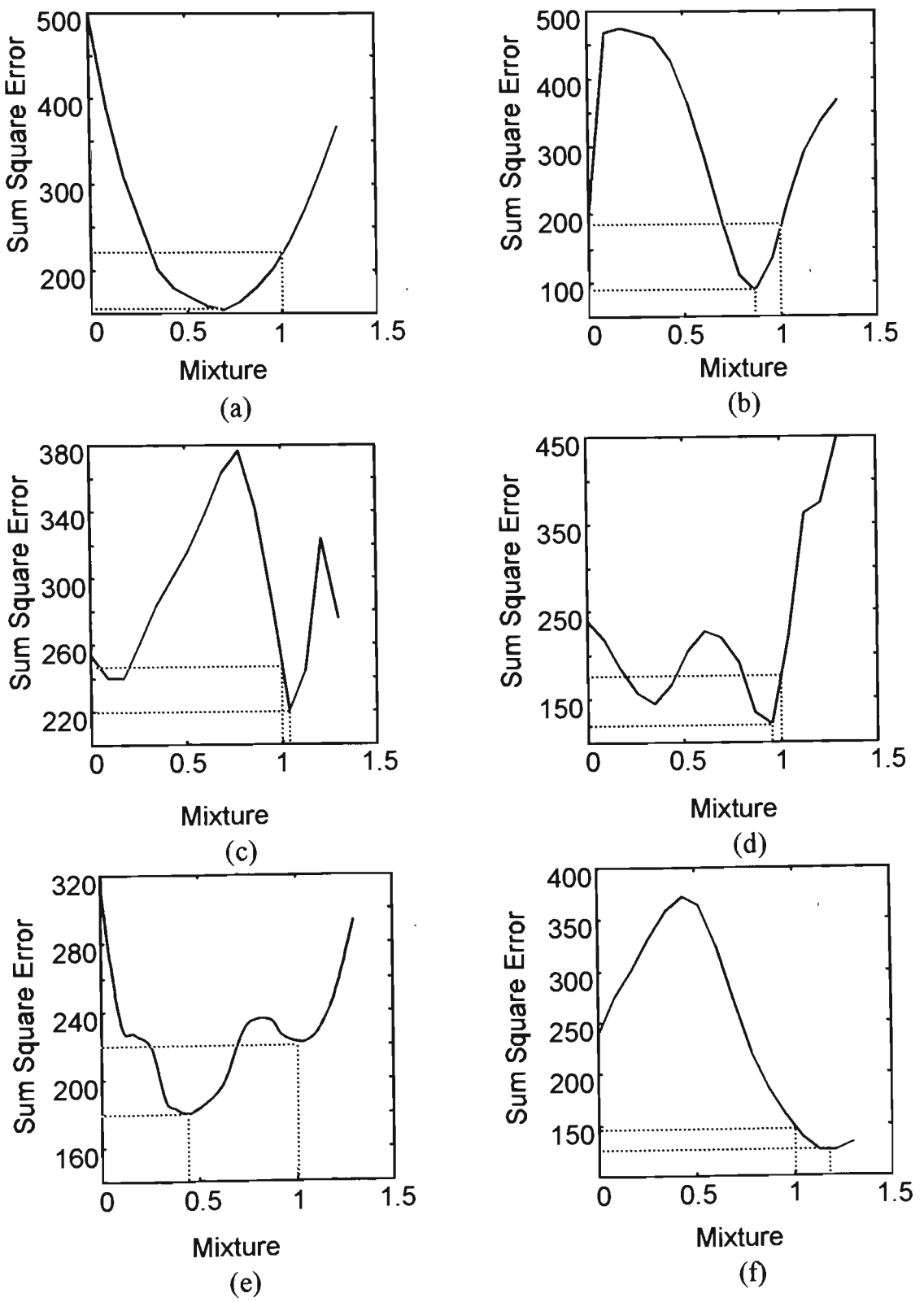
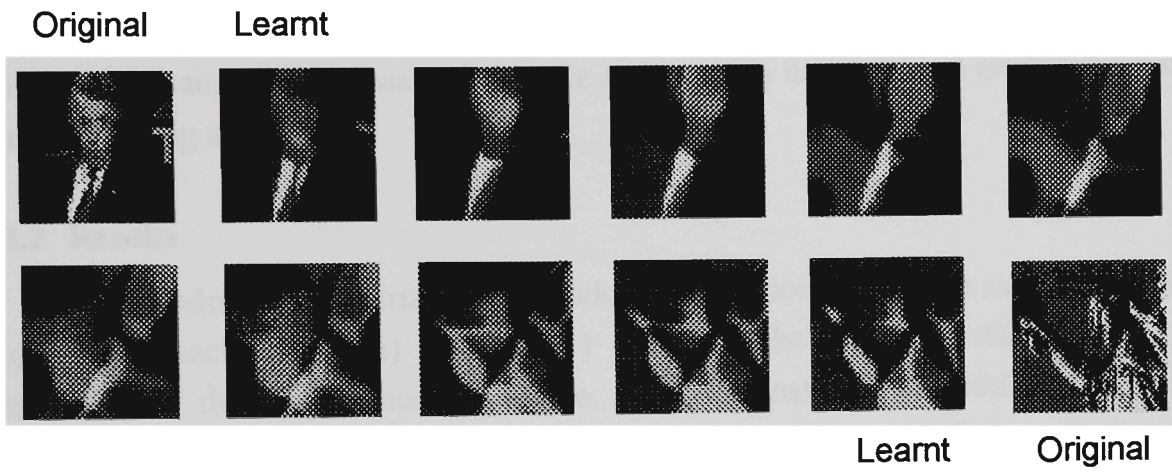
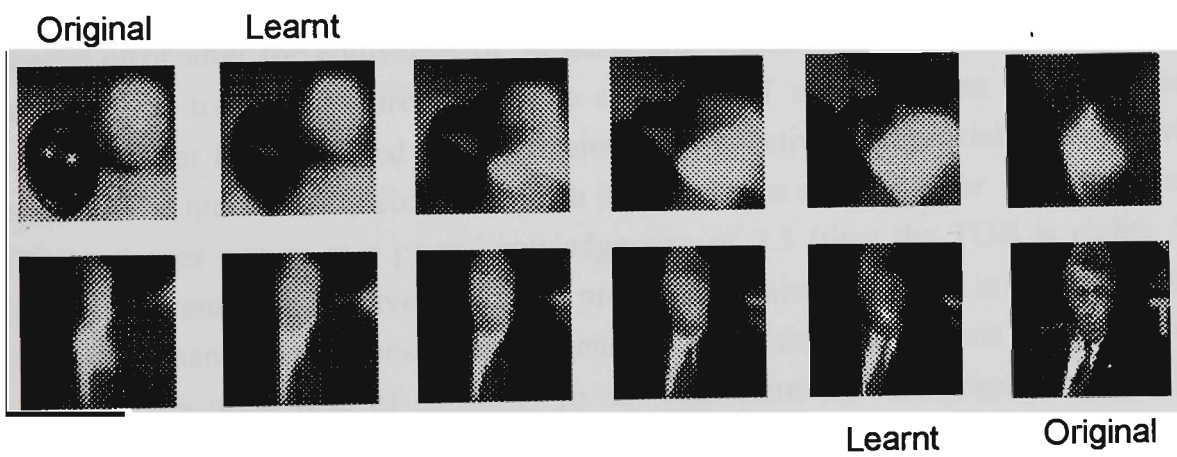


Figure 6.9 : Some typical error curves for the approximation of an image using the mixing of two networks that encode different images.



(a)



(b)

Figure 6.10 : View of the network output as the weights (in the same symmetry region) are moved from one encoded image to another.

of accuracy given by these algorithms is also not required. A total of eight iterations (two for the derivative calculation and six for the search) are used during the optimisation. If the initial difference in sum squared error[3] between the two prior knowledge solutions was less than ten then the optimisation was not used, as it is expected that the cycles would be better spent in the learning algorithm. The ELT algorithm was given 50 learning iterations whilst the optimised transfer had 50-8 = 42 iterations. The assumption is that the cost of the line search is equal to or less than the learning algorithm. This is reasonable because the line search is simple and requires no backward error propagation like the learning algorithm.

6.3.2 Results

Three hundred training trials were conducted per prior knowledge size (one hundred and fifty for each technique). Figure 6.11 compares the literal transfer result to that obtained after the optimisation procedure. This demonstrates the initial sum squared error before the application of the training algorithm for these techniques. Both curves show well defined negative exponential relationships with the optimised transfer at a reduced error.

The performance improvement after training was measured by determining the sum squared error after the equivalent of 50 iterations, which was 42 learning iterations for the optimised transfer (Figure 6.12). The straight ELT algorithm has an almost linear decrease whilst the optimised transfer maintains a negative exponential decrease which appears to be quite closely correlated with its initial sum squared error. The intersection of these curves occurs at a prior knowledge size of 7.5 (thus the TOB is eight). This graph is different to those given for other problem domains because it is the sum squared error rather than the iterations to convergence which is the performance criteria. Despite this there is a good deal of similarity to the equivalent style of graphs for the other problem domains (compare with Figure 6.1 & Figure 6.5).

6.3.3 Transformation Seeded Weight Space

The effects of transformation seeding on the optimisation ability were examined using eight rotations of the problem domains at steps of 45 degrees. Other transformations, of the kind shown in Chapter 3 could be used also to further increase

[3] To convert the sum square error into mean square error divide by the total number of training examples (42 x 42 =1764)

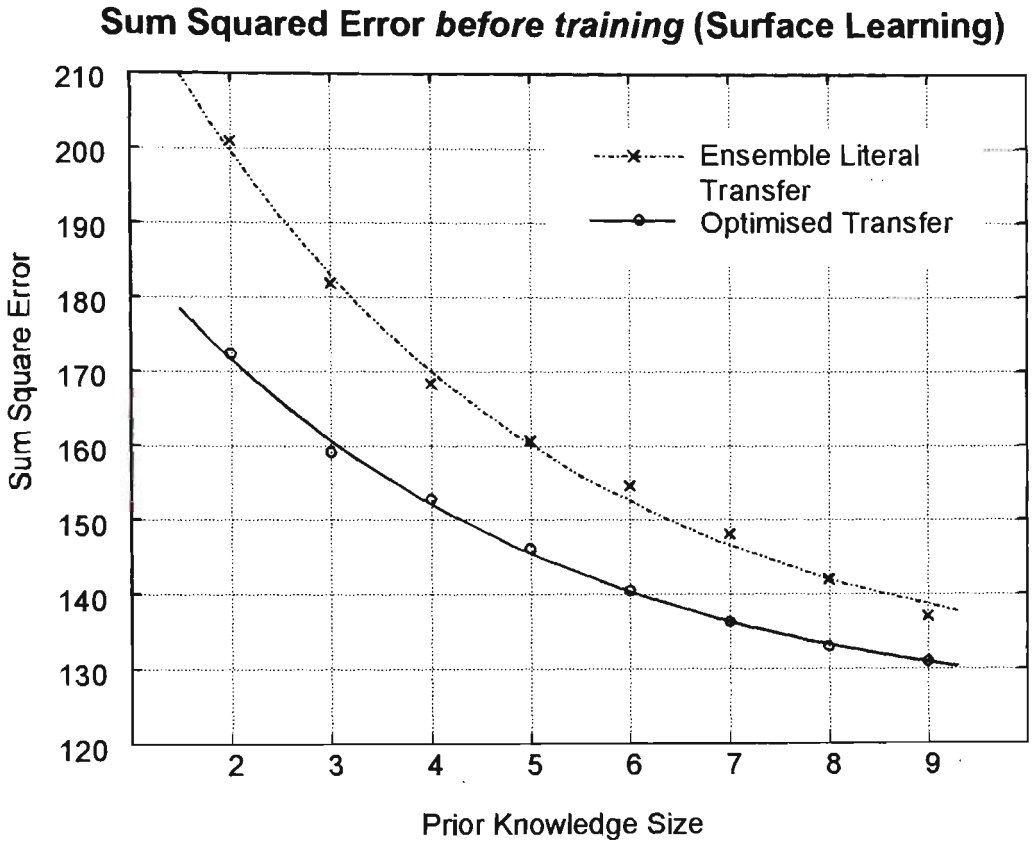


Figure 6.11 : Sum square error *before* the use of the training algorithm with standard optimisation.

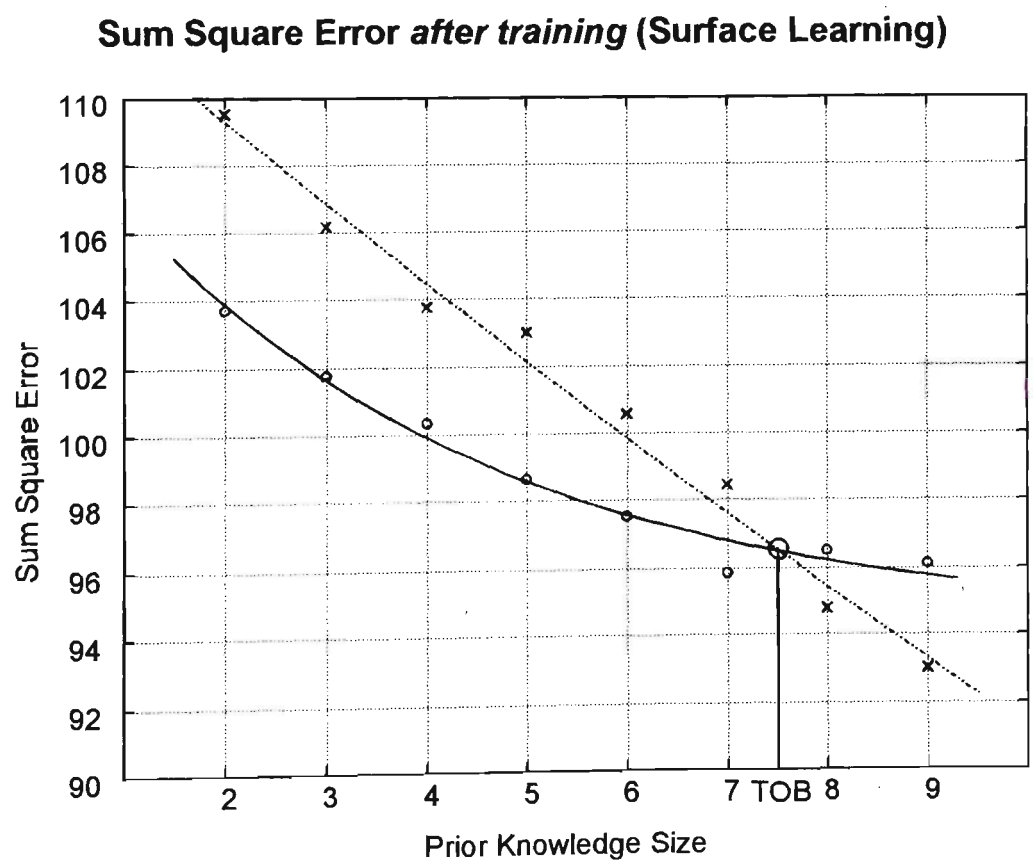


Figure 6.12 : Final sum square error after training for the equivalent of 50 iterations.

the diversity of the prior knowledge. However given the generality of the training images, rotations of the problem domain were the most appropriate as the transform bounds are better defined than for the other affine transformations. The number of solutions in weight space is thus eight times the size of the untransformed prior knowledge.

Each new solution that was added to the prior knowledge had its weights modified using the rotation equation from table 3.2. This transformation is quick because it is operating on relatively few weights. The extra computational load comes when determining the lowest error solution.

The initial sum squared error for the standard optimised transfer and for the transformation seeded optimisation are shown in Figure 6.12. The transformation seeded algorithm shows a significant improvement after a prior knowledge size of two, although the initial errors at a size of two are roughly the same for both techniques. Surprisingly the sum squared error value after training (for the same number of iterations), reveals a wide discrepancy between the final solution errors (Figure 6.13) for this situation. The reason for this is that sum (or mean) square error is not always a reliable estimator of a source networks suitability for transfer. This is because weight space is not smooth. Initialisation near one minima may prevent the learning algorithm from reaching another part of weight space where a better weight solution for a given problem lies. So, although both techniques yield similar error values initially, the transformation seeded weight space solution is on average better suited to transfer to the new target task. Alternative error metrics such as the absolute error or even the sensitivity to training data perturbations may provide enhanced prediction of which network will provide reliable transfer.

The transformation seeded optimisation error appears to approach an asymptote after a prior knowledge size of five. This indicates that there is little benefit to be derived in this problem domain from having more than forty prior knowledge solutions, at least when they have been obtained using the transformation seeding as above. Notice that the error reaches this asymptote much faster in the trained data than in the initialised data. This is because the learning algorithm is finding close to the minimum average error that it is practically able to achieve given a limited number of training cycles.

6.4 Chapter Summary

The experimental results demonstrated in this chapter for the ensemble transfer techniques are quite consistent across the different problem domains. Negative exponential relationships were shown for all transfer techniques with increasing prior knowledge size. In each of the problem domains the transfer optimisation algorithms

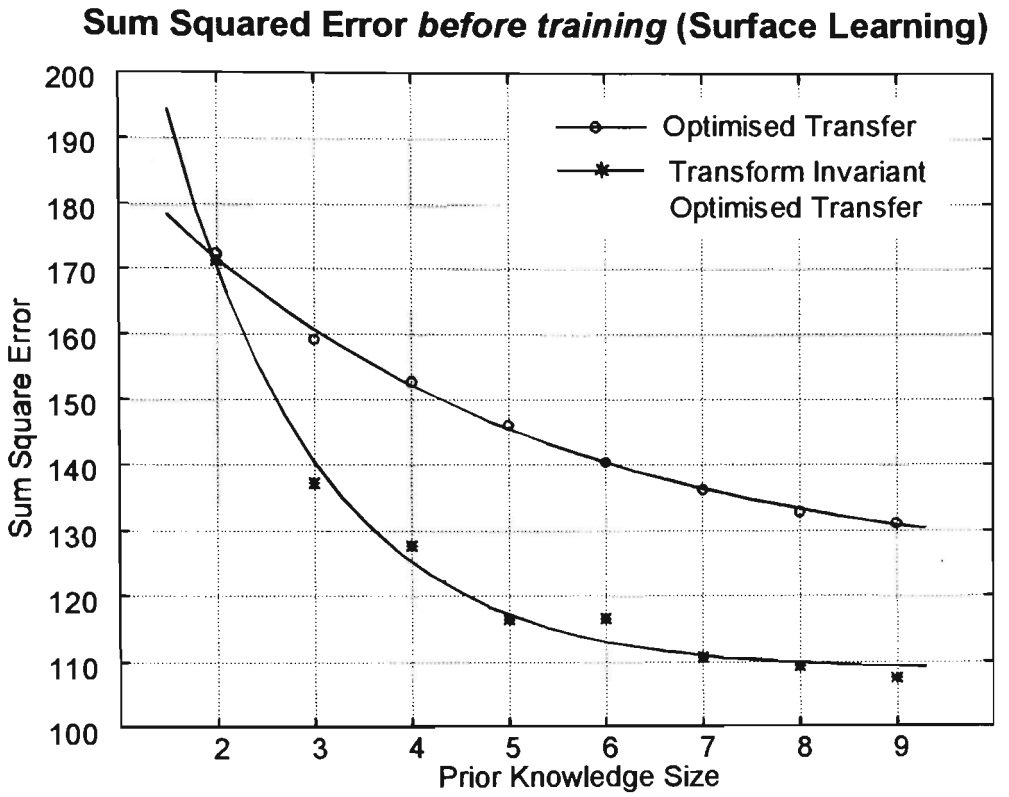


Figure 6.12 : Sum square error *before* the use of the training algorithm for Transformation Seeded Weight Space (8 different rotations per prior knowledge)

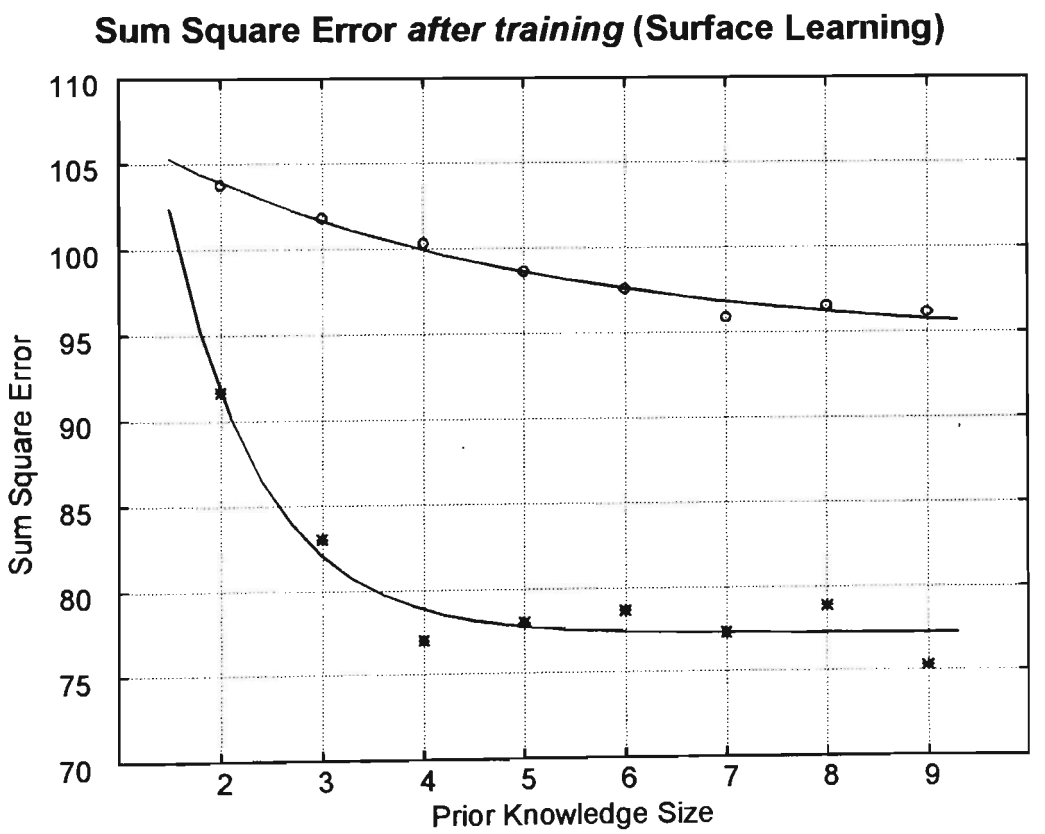


Figure 6.13 : Final sum square error after training both transformed and untransformed prior knowledge for 42 iterations.

achieved a performance improvement over literal transfer up to the optimisation bound. For the classification problems it was also shown that the optimisation produced increased stability during transfer with consequently less unconverged solutions.

Based on the observed performance of the optimisation procedures some recommendations for their practical use can be made :

- The optimisation procedure is useful in situations where there is a limited set of prior solutions available (this is often the case in real world situations).

- Higher regularity in the problem domain leads to improved algorithm performance.

- Optimisation of the prior knowledge increases the stability of learning when compared with literal transfer.

- Any ensemble prior knowledge optimisation procedure, when its performance is compared to ensemble literal transfer, will have some bound (TOB) on the prior knowledge size. Beyond this point the use of the optimisation procedure produces a similar or worse performance to literal transfer.

- It is easier to optimise problems when the ratio of weights to neurons is large, and where the learner has enough representation power to achieve a low empirical error.

- The initial error is not always a good indication of subsequent training speed. An exponential decrease in initialisation error will not always produce a proportional decrease after training.

Chapter 7

Conclusion

In this thesis issues concerning the representation of knowledge by neural networks have been examined. This has provided the background for the construction of practical algorithms for the transfer of knowledge from ensembles of pre-trained networks to new tasks. Both practical and theoretical results have been shown to support the point of view that the knowledge encoded in the weight space of ensembles of trained networks can be used to build a useful map of the representation space.

In addition to providing some background on previous research on transfer Chapter 2 demonstrated that the distribution of prior knowledge solutions in weight space is, in practice, limited by certain physical constrains of the network architecture. This was augmented by empirical results demonstrating the distribution of weights for a simple one dimensional approximation problem.

The concept of transform invariant weight space was introduced in Chapter 3 where the weight space is made invariant to transformations in the problem domain. After demonstrating how certain similarity preserving transformations could be detected in the problem domain it was shown how these transforms affected the knowledge that is encoded by the network. Algorithms for achieving this transformation invariance in weight space for affine transforms in the problem domain were derived for two particular problem types, orthogonal function approximation and auto-correlated classification.

Chapter 4 examined neural network symmetry and its effect on the structure of weight space. The importance of symmetry in understanding network representation was explained and then supported by techniques for visualising these symmetries using weight sharing. Following this equations were derived for both permutation and sign symmetry boundaries. The insight gained from this analysis was used to give a stochastic method of decomposing permutation symmetries in weight space. This algorithm was tested on two problems. Firstly showing how the variation in weight representation for an individual problem could be directly examined and secondly how different solutions could be assessed for equivalence on the neuronal level.

Algorithms to exploit the new weight representations given by the transform invariant and symmetry broken weight vectors were developed in Chapter 5. These

algorithms rely on the ability to be able to mix the symmetry broken weight vectors, since neurons in the same position now usually encode similar functionality. A general technique for the mixing of these solutions was demonstrated by using a fusion matrix to optimise the transfer from the different prior knowledge weight vectors.

Chapter 6 compared this new algorithm to ensemble literal transfer on two different problem domains, where it demonstrated a sustained performance increase over ELT of two to three times, and significantly improved the stability of the transfer. A modified version of the algorithm was used to transfer training amongst image approximation networks. Reduced errors after a training for a fixed time were again found over literal transfer. These results revealed some general conclusions regarding the use of ensemble optimisation techniques which are summarised at the end of Chapter 6.

7.1 Major Contributions

- Concepts relating to environment consistency and transform invariant weight space. The weights are stored in a normalised form to allow transfer to occur independent of similarity preserving transforms in the problem domain (Section 3.3).

- Derivation of equations that relate generalised affine transforms in the problem domain to the transformations that occur in the weight space representation, for orthogonal function approximation. An explanation was also presented for transformations that can occur in the weights of auto-correlated classification problems in response to affine transforms in the problem domain (Section 3.4 & 3.5).

- The use of a neural network for encoding images by learning an approximation to the surface as a good method for directly visualising the representational ability of the network (Section 3.4.1. & 4.6.2.).

- Transformation Seeded Weight Space (Section 3.5.).

- Symmetry visualisation for both sign and permutation symmetry using weight sharing (Section 4.3.).

- Proofs showing both the number and equations of permutation and sign symmetry boundaries. From this a demonstration is provided of the imbalance in weight space between sign and permutation symmetries (Section 4.4).

- A method of decomposing permutation symmetries using simulated annealing. This annealing procedure was supported by two proofs. (Section 4.5.2).

- Techniques for comparing networks solutions (Section 4.6.1 & 4.6.2.).

- The introduction of ensemble literal transfer where multiple solutions are stored and the best one used for literal transfer. A demonstration using this technique that there is a negative exponential decrease in approximation ability with increasing prior knowledge size or functional diversity (Chapters 5 & 6).

- A generalised neuron level 'fusion' optimisation technique for mixing the weight solutions contained in the prior knowledge based on a two stage optimisation procedure (Section 5.3.4.).

- A new metric named Transfer Optimisation Bound (TOB) was introduced for the point at which an ensemble transfer optimisation method breaks even in performance with ensemble literal transfer (Section 6.1.2.2.).

- Performance improvements of *ensemble optimised transfer* over *ensemble literal transfer* were measured on three different problem domains
  - Average training speed increases of two to three times.
  - Sustained average positive transfer performance.
  - Significantly enhanced stability during learning.

7.2 Future Research

In the course of this thesis a number of areas have been found where additional research maybe warranted. Of these, some are direct extensions to the work described in the thesis whilst others are less directly related to the main thesis topic.

An enhanced annealing algorithm for decomposing sign symmetries as well as permutation symmetries would allow comparisons to be drawn about the weight representations of unipolar compared to bipolar activation functions in networks. It is speculated that bipolar activation functions may allow even greater neuron consistency in weight space, and consequently improved transfer results.

Theoretical results regarding the effect that the permutations of the neuron spaces have on a spherical weight space representation would probably aid in the construction of a deterministic algorithm for breaking symmetries. It is not clear however that an NP

algorithm for this symmetry breaking exists, especially in the presence of more than a single hidden layer. This effect is more complicated for sign symmetries than permutation symmetries because of the ripple effect that occurs to all adjacent layer connections when changing the sign on one neuron space.

The suggested applications at the end of the symmetry chapter are worthy of further investigation, particularly those to do with providing explainable or knowledge based neural networks. It should be possible to use the symmetry decomposition to derive new techniques for the incorporation or explanation of neural network solutions.

The use of image approximation networks for unsupervised feature extraction is suggested by an examination of Figure 4.13, especially when coupled with the ability to ensure that neurons in the same positions in different networks represent similar functions.

The method for optimising the fusion matrix currently relies on a stochastic technique for the final stage. A fully deterministic algorithm would be likely to produce better results, given a limited number of iterations. A quasi-Newton's method algorithm similar to conjugate gradient [Mol93] may be appropriate especially when used with the lowest error subset of the prior knowledge.

Some transfer problems are associated with incorrectly placed high magnitude hyperplanes. Techniques such as DBT (Section 2.3.1) try to get around this problem by determining the relevancy of a hyperplane to a particular problem and rescaling the weights accordingly. It is possible that the EOT algorithm could also benefit from such hyperplane rescaling.

Finally some research on relaxing the assumptions about the homogeneity of the network architectures would increase the generality of this work. Many networks can be visualised as being subsets of a larger "mother" network in which all the other weights are zero (even recurrent networks). This common sense statement simply means that every small network can be fleshed out to be a larger one with the same functionality. In some cases it should thus be possible to transfer prior knowledge even when the network architectures are physically dissimilar.

7.3 Potential Applications

Often many problems require only a single network solution that generalises well on a target training set. Thus the application of prior knowledge techniques such as described in this thesis may appear limited in practice. However there are many tasks on which this sort of ensemble based training transfer algorithm can be gainfully employed.

Modular neural networks [JJ91] are probably the best general example of where prior knowledge can be used in non-contrived circumstances. A modular neural network

is one in which the training data is split up (functionally decomposed) and several networks are trained individually on the split data in order to speed training and improve generalisation. After training these networks are glued together into a larger network and final retraining done on the entire problem domain with the large network. This type of technique has been used to provide significant increases in training speed over a single large network [FVL93]. It is suggested that even greater speed increases could be attained in this situation by using ensemble based transfer techniques. The first network to be trained would obviously receive no benefit but the more network modules that were used the larger the base of prior knowledge that would be available. It can been seen, for instance, that in a single system designed to recognise the different speakers from Chapter 5 the EOT techniques would result in an average speed improvement of about 10 times over training the networks with literal transfer initialisation (the integral under the speed-up curve), and significantly more still over using random initialisation.

If on-line trainable neural networks are ever to be included in intelligent consumer products then there will be the requirement for very quick adaptation to new learning tasks. As the domain of knowledge is often well constrained in these applications (eg cars) it would be possible for the manufacturer to ship the product with several pre-trained neural networks for specific environments, and then use these weight vectors as the prior knowledge for rapid adaptation to the new users environment.

When a robot, or other autonomous agent, is active over long periods of time, the ability to make effective use of any available prior knowledge is an valuble quality. This has been termed 'life long learning' [TM95]. The techniques developed in this thesis could doubtless adapted for use under these circumstances.

Bibliography

[Abu93] Abu-Mostafa, Y. S."Hints and the VC Dimension", *Neural computation*. Vol. 5 No. 2, pp 278-288 ,Mar 1993.

[Abu95] Abu-Mostafa, Y.S. "Hints." *Neural computation.* Vol 7 No 4, pp 639- 671., Jul 1995.

[AR91] Al-Mashouq, K.A. Reed, I.S. "Including Hints in Training Neural Nets." *Neural computation.* Vol 3 No 3, pp 418-427. 1991.

[AS92] Albertini, F. Sontag, E.D. "Uniqueness of Weights for Neural Networks" *Tecnhical Report, Rutgers University New Brunswick* pp 1-13, 1992.

[AS93] Albertini, F. Sontag, E.D. "For Neural Networks Function Determines Form" *Neural Networks* Vol 6. No 7. pp. 975-990, 1993.

[Bax95] Baxter "Learning Internal Representations" *PhD Thesis Flinders University*, 1995.

[BB90] Bochereau, L. Bourgine,P. "Rule Extraction and Validity Domain on a Multilayer Neural Network", *IJCNN*, pp 97-100, 1990.

[BB93] Barnard, E Botha, E.C. "Back-Propagation Uses Prior Information Efficiently" *IEEE Transactions on Neural Networks* Vol 4 No 5 Sept 1993.

[BBV93] Blacher, S. Brouers, F. Van Dyck, R. "On the use of fractal concepts in image analysis." *Physica A*. Vol. 197 No. 4, pp 516-530, Aug 1993.

[BCS94] Berger, T. W. Chauvet, G. Sclabassi, R. J. "A Biologically Based Model of Functional Properties of the Hippocampus." *Neural networks* Vol 7 No 6 / 7 1031-1064, 1994.

[BH89] Baum, E.B. Haussler, D "What Size Net Gives Valid Generalisation?" *Neural Computation* Vol 1, pp 151-160. 1989

[BHS92] Barkai, E. Hansel, D. Sompolinsky, H. "Broken Symmetries in Mulitlayered Perceptrons" *Physical Review A*, pp 4146-4161, March 1992.

[BR92] Brown, R.H. Ruchti, T.L. "Gray Layer Technology: Incorporating A Prior Knowledge into Feedforward Artificial Neural Networks" *International Joint Conference on Neural Networks* Vol I pp 806-811, June 1992.

[CH91] Chen, A. M. Hecht-Nielsen, R. "On the geometry of Feedforward Neural Network Weight Spaces" *Neural computation*. Vol. 5 No. 6. Nov 1991.

[Che91] Chen, D S. "Function Representation and Approximation by Neural Networks" PhD Thesis, *University of Michigan* 1991.

[CJL94] Cardell,S.N. Joerding,W. Li,Y. "Why Some Feedforward Networks Cannot Learn Some Polynomials", *Neural Computation* Vol 6, pp 761-766, 1994.

[CJS92] Chen, D.S., Jain, R.C, Schunck, B.G. "Surface reconstruction using neural networks." *Proceedings IEEE Computer Society Conference on Computer Vision and Pattern Recognition* pp 815-17, Jun 1992.

[CK95] Cho, S.-B. Kim, J.H. "Multiple Network Fusion Using Fuzzy Logic." *Ieee transactions on neural networks*,Vol 6 No 2 pp 497- 501, Mar 1995.

[CLH93] Chen, A. M. Lu, H. Hecht-Nielsen, R. "On the Geometry of Feedforward Neural Network Error Surfaces" *Neural computation*. Vol. 5 No. 6. pp 910-916. Nov 1991.

[Coz95] Cozzio,R. "Neural Network Design using A Priori Knowledge", *PhD Thesis Swiss Federal Institue of Technology*, 1995.

[CR94] Cox, M.T. Ram, A. "Choosing Learning Strategies to Achieve Learning Goals". *Proceedings of the 1994 AAAI Spring Symposium on Goal Driven Learning* ,1994.

[DC91] DeMers, D. , Cottrell, G. "Non-Linear Dimensionallity Reduction" *Technical Report 0114* Dept. Of Computer Science, University of California San Diego, pp 1-8, 1991.[Del89] Deller, J.R. "Set Membership Identification in Digital Signal Processing", *IEEE ASSP Magazine*, pp 4-19, 1989.

[DH73] Duda, R.O. Hart, P.E. "Pattern Classification and Scene Analysis" Wiley, New York. 1973.

[Dua85] Duagman J.G. "Uncertainty relation optimized by two dimensional visual cortical filters" J. Opt. Sco. Am. Vol 2. No 7, July 1985.

[DA94] Dunstone E.S, Andrew, J.P," Super High Scale Invariant Image Compression" , *International Symposium on Speech, Image Processing & Neural Networks* , HongKong, Vol 2 pp 397-400, April 1994.

[DG94] Dunstone, E.S., Gray J, "Neural Network Hardware for Parallel Image Encoding", *Parallel Computing : Technology and Practice*, IOS Press, pp 284-285, Nov 1994.

[DG95] Dunstone, E.S. Gray J, "MISD Parallel Image Encoding and Processing", *5th Australian Conference on Neural Networks*, Sydney, pp 205-208, Feb 1995.

[Dun94a] Dunstone E.S.,"Image Processing Using a Surface Learning Neural Network", *First IEEE International Conference on Image Processing,* Austin Texas, Vol 3, pp 912-916, Nov 1994.

[Dun94b] Dunstone, E.S. "Approaches to Breaking Symmetries in Artificial Neural Networks" *1994 International Symposium on Neural Processing* pp 68-75, Tawain, Dec 1994.

[Dun94c] Dunstone, E.S. "Neural Network Weight Space Analysis using Hypersphere Weight Perturbation" *1994 International Symposium on Neural Processing*, Tawain, pp 75-80, Dec 1994.

[Dun94d] Dunstone, E.S. "Neural Network Adaptive Influence Theory: A theoretical Framework for the Transfer of Training" *1994 International Symposium on Neural Processing*, Tawain, pp 81~86, Dec 1994.

[Dun95a] Dunstone, E.S. "Face Recognition using low frequency Gabor information", *6th Australian Conference on Neural Networks,* Sydney, pp 94-97, Feb 1995.

[DO95] Dunstone, E.S. , Ogunbona, P. "A Neural Model for the Transfer of Prior Training", *6th Australian Conference on Neural Networks*, Sydney pp 233-236, Feb 1995.

[DW87] Dennis, J. E. Woods, D. J. (*Matlab Reference Guide* : 1992) "New Computing Environments: Microcomputers in Large-Scale Computing", edited by A. Wouk, *SIAM*, pp. 116-122. 1987.

[Fah88] Fahlman, S.E. "An Empirical Study of Learning Speed in Back-Propagation Networks", *Technical Report CMU-CS-88-162*, Carnegie Mellon University , Sept 1988.

[FC90] Fleming, M. Cotrell, W. "Face Recognition using Unsupervised Feature Extraction" *ProcIJCNN* Vol. 2. pg 65-70, 1990.

[FL90] Fahlman, S.E. , Lebiere, C. "Cascade Correlation Learning Architecture", *Technical Report CMU-CS-90-100*, Carnegie Mellon University , Feb 1990.

[FVL93] Fogelman-Soulie F, Viennet, E., Lamy, B. "Multi-modular Neural Network Architectures", *International Journal of Pattern Recognition and Artificial Intelligence* Vol. 7. No 4. pp 721-755, 1993.

[Gar87] Gardner, E. "Maximum Storage Capacity in Neural Networks.", *Europhysics Letters*, Vol 4. No 4. pp 481-485, 1987.

[GBD92] Geman, G. Bienenstock, E. Dorsat, R. "Bias vs Variance Dilemma" *Neural Computation* Vol 4, pp 1-58, 1992.

[GG84] German, S. German D. "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images." *IEEE Transactions on Pattern Analysis and Machine Learning* Vol-6 , pp 721-741, 1984.

[GR94] Gupta, M.M. Rao, D.H. "On the principles of fuzzy neural networks. (Invited Review)." *Fuzzy sets and systems.*, Vol 61 No 1 pp 1-18 , Jan 1994.

[Gre95] Greenspan, R. J. "Understanding the Genetic Construction of Behavior." *Scientific American.* Vol 272 No 4 pp 72 - 79, Apr 1995.

[Hag93] Hagan, M. (c.f. Matlab™ Neural Network Toolbox 1994) "The Levenberg-Marquardt learning rule", 1993.

[Hal93] Halliday, M. A. K. "Towards a language-based theory of learning." *Linguistics and education.* Vol 5 No 2 pp 93-117, 1993.

[Hau92] Haussler, D. "Decision Theoretic Generalisations of the PAC Model for Neural Networks and Other Applications", *Information and Computation* Vol 100. No 1, pp 78-150 , Sept 1992.

[HC93] Hsieh, K-R. Chen W-T "A Neural Network Model which combines unsupervised and supervised learning" *IEEE Transactions on Neural Networks* vol 4 No 2. Mar 1993.

[Hec89] Hecht-Nielsen, R. "Neurocomputing", *Addison-Wesley Publishing* , 1989.

[HFZ92] Herget, F., Finnoff, W. Zimmerman H.G. "A comparison of Weight Elimination Methods for reducing complexity in Neural Networks" *International Joint Conference on Neural Networks* Vol 3. pp 980-986, Jun 1992.

[HH93] Hush ,D.R. , Horne, B.G. "Progress in Supervised Neural Networks" *IEEE Signal Processing Magazine* pp 8-38, Jan 1993.

[Hin93] Hinton, G. "Tutorial on Neural Networks", Workshop held at Sydney University, Feb 1993.

[HKP91] Hertz, J. Krogh, A. , Palmer, R.G. "Introduction to the Theory of Neural Computation", Addison-Wesley Publishing, 1991.

[HM94] Happel, B.L. Murre, J. "Design and evolution of a modular Neural Network Architecture" *Neural networks*, Vol 7 No 6 / 7, pp 1031-1064, 1994.

[Hou91] Houle,G. "On the use of A Priori Knowledge to Charcter Recognition", *IEEE Conference on Pattern Recognition,* pp 1415-1420, 1991.

[IZ93] Ibarra, O.H. Zheng, Q. "Some Efficient Algorithms for Permutation Graphs " Journal of Algorithms Vol 16 No 3. pp 198-204, May 1994.

[Jai81] Jain, A.K. "Advances in Mathematical Models for Image Processing", *Proceedings of the IEEE*, Vol. 69. No 5., pp 503-528, May 1981.

[JC92] Jordan, F. Clement, G. "Using Symmetries of a Multi-layered Network to reduce the Weight Space" *International Joint Conference on Neural Networks* Vol 2. pp 391-396, 1994.

[JJ91] Jacobs, R.A. Jorden, M.I. "A competitive modular connectionist architecture", *Advances in Neural Information Processing Systems 3*, San Mateo, CA Morgan Kaufmann, pp 767-773, 1991.

[JS95] Jackson, S.A. Sharkey, N. E."Adaptive Generalization in Dynamic Neural Networks" *Univeristy of Sheffield*, Technical Report 1995.

[KGV83] Kirkpatrick,S., Gelatt, C.D. Vecchi, M.P. (1983) "Optimisation Simulated by Annealing" *Science* 220, 671-680. Reprinted in Anderson and Rosenberg [1988]

[KK94] Kûrková, V., Kainen, P. "Functionally Equivalent Feedforward Neural Networks", *Neural Computation* 6 ,543-558 ,1994.

[Kei87] Keifer J.C. "Introduction to Statistical Inference" Springer-Verlag, New York, 1984.

[KMV94] Konon, W.K. Maurer, T. Von der Malsburg, C. "Fast Dynamic Link Matching Algorithm for Invariant Patern Recognition", *Neural networks*, Vol 7 No 6 / 7, pg 1019-1030, 1994.

[Koh89] Kohonen, T. "Self Organization and Associative Memory" (3rd ed.). Berlin Springer-Verlag. 1989.

[Kru92] Kruschke, J. K. "ALCOVE: An Exemplar-Based Connectionist Model of Category Learning." *Psychological review.* Vol. 99 No. 1, May 1992.

[Kru93] Kruschke, J K."Human Category Learning: Implications for Backpropagation Models." *Connection science* Vol 5 No 1, pp 3-36, 1993.

[Lai86] Laird, J. "Universal subgoaling" *in Universal subgoaling and chunking : The automatic generation and learning of goal hierarchies* (eds Larird, J. Rosenbloom, P., Newell, A.) Kluwer pp 3-131. 1986.

[Lor91] Loritz, D "Cerebral and cerebellar models of language learning." *Applied linguistics*. Vol 12 No 3 pp 299-310, Sep 1991.

[LSS94] Li, R., Shen, Y., Specht, S.M. "The Hippocampus: A Biological Model for Studying Learning and Memory". *Progress in neurobiology*. Vol. 44 No. 5 pp 485, Dec 1994.

[LVV+93] Lades, M. Vorbrüggen, J. Von der Malsburg, C (et al.). "Applying dynamic link matching to Object recogntion in real world images" *ICANN (pre-print)*, 1993.

[Mas95] Maass, W. "Perspectives of Current Research about the complexity of Learning on Neural Nets" *NeuroCOLT Technical Report Series - Royal Holloway University of London* , NC-TR-95-003, Mar 1995.

[MO94] Monasson, R. O'Kane, D. "Domains of Solutions and Replica Symmetry Breaking in Multilayer Neural Networks" *Europhysics Letters*, Vol 27 no 2, pp 85-90, 1994.

[Mol93] Moller, M. F. ``A scaled conjugate gradient algorithm for fast supervised learning'', *Neural Networks*, Vol. 6, No. 4, pp. 525-533, 1993.

[MP95] Mitra, S. Pal, S.K. "Fuzzy Multi-Layer Perceptron, Inferencing and Rule Generation." *Ieee transactions on neural networks*. Vol 6 No 1 pp 51-63 Jan 1995.

[MS93] Macintyre, A . Sontag, E.D. "Finiteness Result for Sigmoidal Neural Networks" *Proc. 25th Annual Symp. Theory Computing San Diego*, May 1993.

[MT94] Mahadevan, S. Tadepalli, P. "Quantifying Prior Determination Knowledge Using the PAC Learning Model", *Machine Learning* Vol 17 no. 1, pp 70-105, Oct 1994.

[Mur95] Murre, J.M.J. "Transfer of learning in backpropagation and in related neural network models" to appear in *Connectionist Models of Memory and Language* J. Levy, D. Bairakitaris & P. Cairns (eds) London UCL Press, 1995.

[MW92] Minai, A.A. Williams, R.D. "Perturbation Response in Feed-Forward Neural Networks" *International Joint Conference on Neural Networks*, Vol III pp 857-862 Jun 1992.

[NM92] Naik, D.K., Mammone, R.J. T.L. "Meta-Neural Networks that learn by learning" *International Joint Conference on Neural Networks* Vol I pp 437-444, Jun 1992.

[NP94] Nadal, J-P, Parga, N. "Duality Between Learning Machines: Abridge Between Supervised and Unsupervised Learning" *Neural Computation* pp 491-507, May 1994.

[OB94] On Plomin, R. Bergeman, C. S. "The nature of nurture: Genetic influence on "environmental" measures." *The behavioral and brain sciences*. Vol. 17 No. 4 pp 750-763, Dec 1994.

[Ohl93] Ohlsson , S. "The Interaction Between Knowledge and Practice in the Acquisition of Cognitive Skills" In S. Chipman, A. Meyrowitz (eds) *Foundations of Knowledge Acquisition, Cognitive Models of Complex Learning, Kluwer Academic Publishers* pp 147-208, 1993.

[ON78] O'Keefe, J. Nadel, L. "The hippocampus as a cognitive map" Oxford New York Clarendon Press, 1978.

[Phi93] Philip, S. "Effect of Representation on Error Surface" *Fourth Australian Conference on Neural Networks* pp 86-89 Feb 1993.

[Pij92] Pijpers, M. Alder, M.D. "Affine Transformations of the Speech Space", Proceedings of the *Fourth Australian Conference on Speech Science and Technology*, Brisbane, Australia, pp. 124-129, Dec 1992.

[PL94] Picard, R.W. Liu,F. " A new wold ordering for image similarity", *IEEE ICASP* Vol 5, pp 129-132, 1994.

[Pra93] Pratt, L.Y. "Transfering Previously Learned Back-propations results to new learning tasks", *PhD Thesis , Rutgers University* May 1993.

[Pri84] Price, G.B. "Multivariable Analysis", Springer-Verlag New York, 1984.

[PS94] Partridge,D. Sharkey, N.E. "Neural Computing for Software Reliability", *Expert Systems*, Vol 11. No 3, Aug 1994.

[PT90] Pentland, A. Turk, M. "Recognition in Face Space", *SPIE Intelligent Robots and Computer Vision* Vol 9, 42-54, 1990.

[Rad92] Radford, M. N. "Bayesian Training of Backpropagation Networks by the Hybrid Monte Carlo Method" *Tech Report CRG-TR-92-1, Department of Computer Science, University of Toronto* pp 1-21 Apr 1992.

[Rol95] Rolls, E.T. "Learning mechanisms in the temporal lobe visual cortex.", *Behavioural brain research*. Vol 66 No 1 / 2 pp 177 , Jan 23 1995.

[RM86] Rumelhart, D.E. McClelland, J.L. (eds) "Parallel Distributed Processing : Explorations in the Microsturcture of Cognition" , Vol 1. Cambridge, MA : MIT Press, 1986.

[Sch93] Schmidhurber, J. "A neural network that embeds its own Meta-Levels", *International Conference on Neural Networks*, pp 407-412,1993.

[Sha91] Sharkey, N.E. "Connectionist representation techniques. ", *The Artificial Intelligence Review* Vol. 5 No. 3, pp 143, 1991.

[Sha93] Shawe-Taylor, J. "Symmetries and Discriminability in Feed forward Neural Architectures " *IEEE Transactions on Neural Networks* Vol 4, No 5. Pp 816-826 Sept 1993.

[Spa91] Spann,M. "Figure/Ground Separation using Stochastic Pyramid Relinking" *Pattern Recogntion* Vol 24. No 10 pp 993-1002, 1991.

[SSS+90] Schwartz, D.B., V.K. Salaman, S.A. Solla J.S. Denker. "Exhaustive Learning' *Neural Computation* 2, pp 371-382, 1990.

[Sus92] Sussmann, H.J. "Uniqueness of the Weights for Minimal Feedforward Nets with a given Input-Output Map", *Neural Networks* Vol 5, pp 589-593, 1992.

[TM93] Thrun, S. Mitchell, T. "Integrating Inductive and Explanation Based Learning" *Advances in Neural Information Processing Systems 5* - Morgan Kaufmann 1993.

[TM95] Thrun, S., Mitchell, T. M., "Lifelong robot learning" *Robotics and autonomous systems.*", Vol. 15 No. 1 / 2 pp 25, 1995.

[Toh93] Toh, H.S. "Weight Configurations of Trained Perceptrons" *International Journal of Neural Systems* Vol 4. No 3 pp 231-246, Sept 1993.

[TS93] Towell, G. G. Shavlik, J. W. "Extracting Refined Rules from Knowledge Based Neural Networks." *Machine learning*, Vol. 13 No. 1 pp 71 -101 Oct 1993.

[VC71] Vapnik, V.N. and Chervonenkis "On the Uniform Convergence of Relative Frequencies of Events to Their Probabilites". *Theory of Propability and Its Applications* Vol. 16. pp 264-280. 1971

[VMR+88] Vogl, T.P. Mangis, J.K. Rigler, A.K., Zink,. W.T. Alkon, D.L. "Accelerating the convergence of the back-propagation method", Biological Cybernetics, Vol, 5.8, pp. 257-263, 1988. (Matlab™ Neural Network Toolbox 1994)

[WT91] Wejchert, J. Tesauro, G. "Visualizing processes in neural networks." *IBM Journal of Research and Development* Vol 35, pp 244-253, 1991.

[Wer93] Werner, F. "On the heuristic solution of the permutation flow shop problem by path algorithms.", *Computers and Operations Research*, Vol 20 No 7, pp 707-711, Oct 1993.

APPENDIX A

Learning and Generalisation Bounds

A.1. Introduction

It is possible to place bounds on the generalisation ability of a learner for a given number of training samples. The following appendix draws on explanations and results from two main sources [HKP91, Bax95] and intended as a brief introduction to the calculation of these bounds.

Generalisation ability may be quantified as the probability that the ideal function f is approximated exactly by the learners hypothesis,

$$g(f) = \text{Prob}(f(x) = h(x))$$

This generalisation ability is independent of the actual training examples used but is however biased by the representational power of the learner. For a given number of training samples p, sampled from the problems underlying probability distribution $P(x)$, we can define the generalisation ability to be $g_p(f)$. This generalisation ability will be greater, even as p goes to infinity, than the ideal case $g(f)$ because the learners functional diversity biases the range of representable functions that learning rule can examine.

It is possible for there to be almost perfect generalisation on the training set, $g_p(f) = 1$, but for the error on the other values not sampled to be large, leading to the actual generalisation to be very poor, in the worst case $g(f) \rightarrow 0$ (this is due to over-fitting the data). If a learner has good generalisation on a training set it is said to be *empirically correct* as it satisfies the available empirical data. It is this estimation which is used as the basis for the adjustment of weights in a learning algorithm.

Consider a loss function l [Bax95] that measures performance of the learner on a training set[1]. Then the error of the learner on the Probability function P is,

$$E(l,P) = \int_{X \times Y} l(y,h(x))dP(x,y). \tag{A.1}$$

[1] For the mean square error, $l = (h(x)-y)^2$.

As the loss function approaches zero then this error may be written as a function of the choosen hypothesis and the probabilty distribution.

(A.2) $$E(h,P) \approx \int_{Z} h(z)dP(z).$$

The learner has no knowledge about the distribution $P(z)$ only about the samples $\vec{z}$ which are selected from the sample space Z. In the absence of prior knowledge about the nature of $P(z)$ the best estimate of the true loss is the average of the observed losses.

(A.3) $$E(h, \vec{z}) = \frac{1}{p}\sum_{i=1}^{p} l_h(z_i).$$

PAC learning can be used to provide answers to two main questions [Bax95]: Under what conditions can a hypothesis h be produced by a learner such that it has a small empirical loss (eq. A.3.), and how confident can one be that the empirical loss will be close to the true loss (eq. A.1.) ?

These two problems help to define weaker learning conditions on which PAC learning is based. Probably consistent (PC) learning is where the learner performs equally well on both the training and testing set, although it does not account for the actual quality of that learning, and empirically correct (EC) learning which simply specifies that the learner should perform well on the training set. Together PC and EC learning are used to imply PAC learning. For a more complete description of the mathematical reasoning behind these descriptions the reader is referred to either Haussler [Has92] or Baxter [Bax95].

A.2. Probably Consistent Learning Bounds

To place bounds on PC learning one must calculate the expected deviation between the true and empirical loss. The following definitions are necessary to give an appreciation for the meaning of these bounds.

A pseudo metric is a measure in which points which are zero distance apart using the metric may still be different points. The pseudo metric which is defined by,

(A.4) $$d_p(f,g) = \int_{Z} |f(z) - g(z)|\, dP(z)$$

gives a measure of the distance between the two functions f and g, drawn from the function space $\mathcal{F}$, over the whole training set Z. Because of the integral, $d_{p.}$ may be zero even when f and g are different functions. The function space $\mathcal{F}$ provides mappings which transform the training set Z to the set of possible outcomes of dimension M. $\mathcal{F} : Z \rightarrow [0, M]$.

A *pseudo metric space* is defined by the pair ($\mathcal{F}$, $d_{p.}$). The concept of metric spaces is directly related to that of fractal dimension where space is recursively broken into smaller self similar pieces. The metric d_p is used to define the number of balls of radius ε which are needed to cover the hypothesis space with respect to the capacity of the network[2]. The *ε-cover* of ($\mathcal{F}$, $d_{p.}$) can then be defined as any subset $F \subseteq \mathcal{F}$ such that the distance between a function f in the whole of function space $\mathcal{F}$, and a function f' in part of that space F, is less than or equal to ε or $d_{p.}(f, g) \leq \varepsilon$. The capacity C of the function space $\mathcal{F}$ is then given in terms of the maximum of the size of the smallest *ε-cover*, $N(\varepsilon., \mathcal{F}, d_{p.})$.

To be guaranteed with probability 1-δ that there will be a deviation d_v[3] of less than α between a learners true and empirical loss a sample of size p is required, such that,

$$p > \frac{8M}{\alpha^2 v} \ln\left(\frac{4C(\alpha v / 8, l_H)}{\delta}\right). \tag{A.5}$$

This result implies that to be probably consistent, the learners capacity $C(\alpha v / 8, l_H)$ must be finite. Thus relationships are established between the capacity of the hypothesis space, the number of training examples and the ability of the learner to be probably consistent.

A.3. Vapnik-Chervonenkis dimension (VC dim)

The Vapnik-Chervonenkis dimension deals with the number of training samples required to bound poor generalisation. By considering an arbitrary function that is implementable using the networks architecture rather than a specific set of weights that match a training set, the following on this probability can be shown [VC71],

---

[2] More specifically the VC dimension (See Section B.3).

[3] d_v is another metric space which is defined on $\Re^+$ by $d_v(x, y) = \frac{|x - y|}{v + x + y}$. This space has the advantage over Euclidean distance that it is a relative measure which scales dependant on the size of x and y.

(A.6) $$\text{Prob}\left(\max_f \left|g_p(f) - g(f)\right| > \varepsilon\right) \leq 4m(2p)e^{-\varepsilon^2 p/8}$$

Equation A.6. relates the probability of worse case estimation error for a given number of samples to an equation involving a *growth function m* that is dependant on the number of training examples. The growth function is defined as the maximum number of dichotomies or combinations for which hypotheses can be generated on any set of p examples. For p examples there are maximum of 2^p hypothesis that could be made concerning the classification of these points. However the sort of hypothesises that can be generated are limited by the representational power of the network so $m(\text{p}) \leq 2^p$. For situations where the growth of $m(\text{p})$ is restricted to being less than exponential, the bounds on the probability of poor generalisation can be made arbitrarily small by providing enough training examples.

The Vapnik Chervonenkis dimension relies on the fact that this growth function can be proved to be always equal to 2^p up to some number of training examples, after which it starts to slow down. The size of training set for which this occurs is known as the VC dimension, d_{VC} . If the VC dimension is infinite then the network will never be able to generalise, however for a finite d_{VC} the growth function will always be such that,

(A.7) $$\text{m}(p) \leq p^{d_{VC}} + 1.$$

Thus the growth is no longer exponential in p after the point d_{VC} . This then allows the right hand side of equation A.6. to be useful in bounding the generalisation error. Theoretical results using these bounds have proved, for instance, that the upper bound on the VC dimension of a feed-forward network with M threshold nodes and W weights is [BH89]

(A.8) $$\text{d}_{\text{vc}} \leq 2\ \text{W}\ \log_2\ (\text{e}^1\text{M})$$

Using this result, the upper bound of the size of the training set, for generalisation error of ε, was calculated to be $\frac{W}{\varepsilon}\log\frac{M}{\varepsilon}$, (given a error on the training set of ε/2). Numerous results of this nature have been derived however direct application of such results is often difficult because the bounds tend to require far more training examples than is practically possible to obtain, or is computational feasible to learn. There is some current research (see [Maa95] for a comprehensive review) that is attempting to address these problems and provide more practical and realistic learning bounds.

APPENDIX B

Spherical Throngs of Optima

An Mei Chen's and Robert Hecht-Nielsen's [CH91,CLH93] papers on the geometry of feed-forward neural network weight space were the first serious attempt to analyse the structure of weight space from the perspective of symmetries. Some of these results from this papers are reproduced below, reworded so as to be consistent with the notation used in this thesis. The more detailed proofs are not reproduced here and the curious reader is referred to the given references.

THEOREM B.1 [Chen]: *All equioutput transformations are combinations of interchange (permutation) and sign flip transformation.*

PROOF [Chen] *: By induction. Look first at the output layer. Since each unit is linear , the only weight transformations that can leave the output unchanged are sign flips and interchanges. This constrains the final hidden layer to interchanges of neurons and sign flips of the output signal. But these can only be achieved via unit weight vector interchanges and unit weight vector sign flips. The same argument holds for each earlier hidden layer.* □

THEOREM B.2 [Chen]: *The set of all equioutput transformations form a non-Albelian group G (a subgroup of the reflection group of the weight space) of order,*

$$\prod_{i=2}^{K-1}(M_i!)(2^{M_i}).$$

This result is the same as shown for the total number of symmetries in Chapter 4. Chen goes on to state that the set of all such transformations form a group G because they are a subset of the reflection group (the group of all geometric reflections) and also because the product of any two transformations in the set is another transformation in the set.

Following from the existence of a the group G some geometric properties of these groups are derived.

THEOREM B.3 [Chen]: *There exists a minimal sufficient search set which is a cone. Assuming it is Borel measurable, this cone occupies a volumetric fraction of weight space equal to the inverse of the order of the group G.*

This property is shown by considering the weight vectors that lie on the unit hypersphere sphere.

THEOREM B.3 [Chen]: *There exists a minimal sufficient search set which is a cone. Assuming it is Borel measureable, this cone occupies a volumetric fraction of weight space equal to the inverse of the order of the group G.*

THEOREM B.4 [Chen]: *The average distance between each element of G and the next closest element is less than d, where d goes to zero as the number of weights increases. The value of d can be derived from the following equation*

$$\frac{A}{\#G} = \frac{2^{q-1}\pi^{(q-2)/2}\left(\frac{q-2}{2}\right)!}{(q-1)!}d^{q-1}$$

where q is the number of weights in the network.

This equation was derived from the area of a hyper-sphere of dimension equal to q. Because the solutions are equally spread, the area for each symmetry region is just divided by the number of symmetric regions. This gives the actual surface area occupied by any one region from which the distance to an adjacent region d can be calculated.

It is also reported that empirical results suggest the radius $|\mathbf{w}|$ is usually less than $\sqrt{q}$. A theoretical explaination for this curious result is however not provided.

APPENDIX C

Transformation Invariant and Symmetry Proofs

This appendix contains proofs for all theorems from the thesis. Definitional terms are not redefined here, so the reader is referred to the relevant section to provide the relevant context for these proofs.

C.1. Affine Transformations in Weight Space (Section 3.4)

THEOREM 3.1 : *In a finite MLP network topology, where the input is connected to the first layer only, affine transformations T_A in the problem domain Z can be achieved by a function $k_A(\mathbf{T}.\mathbf{w})$ where $\mathbf{T}.\mathbf{w} = W^l$, if and only if $l=1$.*

PROOF :

for case $l=1$,

The transformation to the solution space for $l=1$ is $k_A(W^1)$. *We wish to show that the equivalent affine transform* T_A *is generated in the problem space : The output of each layer in a network is an affine transformed copy of the layer below, modified by the activation function A. Provided that A is deterministic and non constant, a linear modification to the first layer weights produces an equivalent linear modification in the output of layer two* X^2,

$$T_A(X^2) = A_1(k_A(W^1)X_o)$$

This change then propagates through all layers to the output.

$$T_A(Z) = T_A(X^{L-1}) = A_{L-1}(W^{L-1}A_{l-1}(.....W^2A_2(W^1A_1(k_A(W^1)X_0))..)$$

Because the inputs are directly modified by any transformations in the first layer weights it is apparent that the whole of the problem domain will be similarly transformed.

for case $l>1$,

When the transformation occurs in layers other than the first it is operating on signals which have already been passed through the one way activation function A. In other words it is not possible to reconstruct the input with only a knowledge of the outputs from these neurons. Thus there can be no way of constructing an affine transformation on the input by a modification to these layers

Modifications to the weights in any layer other than the first, $k_A(W^l)$ *thus cannot perform an affine transform on the problem domain.*

$$T_A(Z) \neq A_{L-1}(.... A_l(k_A(W^l)A_{l-1}...W^2A_1(W^1X^0)..), \qquad 2 \leq l < L$$

■

C.2. Surface Approximation Affine Transformations (Section 3.5.1)

TRANSLATION BY $(t_1, t_2, \ldots t_n)$,

$$\begin{aligned} N &= (x_1+t_1)\,w_1 + (x_2+t_2)\,w_2 + \ldots + (x_n+t_n)\,w_n + \beta \\ &= x_1 w_1 + x_2 w_2 + \ldots + x_n w_n + (t_1 w_1 + t_2 w_2 + \ldots . t_n w_n + \beta) \end{aligned}$$

$$k_{translation}(t_1, t_2, \ldots, t_n) = \begin{cases} \beta = \beta' + \sum_{i=0}^{n} t_i w_i \\ w_j = w_j \end{cases}$$

■

SCALING BY $(s_1, s_2, \ldots s_n)$

$$N = s_1 x_1 w_1 + s_2 x_2 w_2 + \ldots . + s_n x_n w_n + \beta$$

$$k_{scaling}(s_1, s_2, \ldots, s_n) = \begin{cases} \beta = \beta' \\ w_j = s_j w_j \end{cases}$$

■

SHEAR BY $(h_1, h_2, \ldots h_n)$

$$\begin{aligned} N &= (x_1 + h_2 x_2 + \ldots h_n x_n)\, w_1 + (x_2 + h_1 x_1 + h_3 x_3 . + \ldots h_n x_n)\, w_2 + \ldots \\ &\qquad (x_n + h_1 x_1 + \ldots h_{n-1} x_{n-1})\, w_n + \beta \end{aligned}$$

$$= x_1 (w_1 + \sum_{j=2}^{n} h_j w_j - h_1 w_1) + x_2 (w_2 + \sum_{j=1}^{n} h_j w_j - h_2 w_2) + \ldots . + \beta$$

(3.1)
$$k_{shear}(h_1, h_2, \ldots, h_n) = \begin{cases} \beta = \beta' \\ w_j = w_j - h_j w_j + \sum_{i=1}^{n} h_i w_i \end{cases}$$

■

ROTATION BY (θ), (2D case)

For clarity only the two dimensional rotation case derived here. (The n-dimensional case can be derived from the multi-dimensional spherical coordinate equations in Chapter 9 of [Pri84]).

$$
\begin{aligned}
x_1 w_1 + x_2 w_2 + \beta &= w_1 (x_1 \cos(\theta) + x_2 \sin(\theta)) + w_2 (x_2 \cos(\theta) + x_1 \sin(\theta)) + \beta \\
&= w_1 x_1 \cos(\theta) - w_1 x_2 \sin(\theta) + w_2 x_2 \cos(\theta) + w_2 x_1 \sin(\theta) + \beta \\
&= x_1 (w_1 \cos(\theta) + w_2 \sin(\theta)) + x_2 (w_2 \cos(\theta) - w_1 \sin(\theta)) + \beta
\end{aligned}
$$

$$
k_{rotation}(\theta) = \begin{cases} \beta = \beta' \\ w_1 = w_1 \cos(\theta) + w_2 \sin(\theta) \\ w_2 = w_2 \cos(\theta) - w_1 \sin(\theta) \end{cases}
$$

■

C.3. Permutation Symmetry Boundaries (Section 4.4.1)

THEOREM 4.1 : *All permutation symmetric regions within a layer are divided by* $\frac{n_l!}{2}$ *boundaries which are defined by the equality of the permutations of the neuron spaces* ψ *within that layer.*

PROOF : *Consider the complete set of neuron spaces in a particular layer l,* Ψ_l. *To examine all permutations of the vector* Ψ_l *we can define a tree structure where at each layer the vector is split into all possible vectors with different starting combinations for the remaining part of the vector. The number of nodes at depth k is then,*

$$\frac{n_l!}{(n_l-k)!}$$

At layer n_l-2 *the nodes are binary, and contain every possible combination of two of the neuron spaces* ψ *from the* Ψ_l. *The equality of the vectors in these nodes defines the symmetry boundaries.*

$$\psi_{ml}=\psi_{pl}\ ,\ m\neq p\ \ \forall\ \ m,p\in\{1..n_l\}$$

where the number of such boundaries is equal to the number of nodes at layer n_l-2.

$$\frac{n_l!}{(n_l-(n_l-2))!}=\frac{n_l!}{2}.$$

■

C.5. Sign Symmetry Boundaries (Section 4.4.2)

THEOREM 4.2 : *In a network layer with n neurons and bipolar activation functions, there are* $2^{(n-1)}$ *sign symmetric boundaries which are defined by the neuron spaces being equal to the zero weight vector.*

PROOF : *In networks with bipolar activation function neurons, sign symmetric boundaries are defined by the ability to multiply both fan-in and fan-out weights by minus one and leaving the function of the neuron unchanged. Thus the boundaries may be expressed by,*

$$\psi_{il} = -\psi_{il} \quad \forall\, i, l$$

This equation is only satisfied when,

$$\psi_{il} = 0 \quad \forall\, i, l \quad .$$

Because the sign symmetric regions constitute all possible dicotomies of the sign on the neuron space vectors there are 2^n *sign symmetry regions. The symmetric regions are partitioned by the symmetry boundaries so the number of such boundaries will be*
$2^n / 2 = 2^{(n-1)}$
■

C.6. Topology of Symmetry Regions (Section 4.4.3)

THEOREM 4.3 : *The symmetry boundaries in a bipolar activation function network define non symmetric regions in weight space which extend outward from the origin. The hyper-spherical surface areas described by weight vectors of a fixed magnitude will be elongated by a factor of,*

$$\prod_{k=1}^{L} \frac{n_k!}{2^{n_k}}.$$

PROOF : *Permutation symmetry planes have a super symmetric axis where all weights are equal, which is at 45 degrees to all the positive axes. The sign symmetry planes however are orthogonal, since they lie along all axes of weight space and so have no super symmetric axis, hence the permutation symmetry planes will always lie at 45 degrees to the positive side of all sign symmetric planes. From Lemma 4.1 all the symmetry planes pass through the origin, so the minimal solution slices must extend outwards from the origin.*

There is a large imbalance in weight space between the number of sign symmetries and permutation symmetries. This imbalance has the effect that the symmetry regions in high dimensional space will tend to be elongated along the sign symmetry axis by a factor $r_{p/r}$ *which is the ratio of permutation symmetries to sign symmetries in the network,*

$$r_{p/s} = \frac{N_p}{N_s} = \prod_{k=1}^{L} \frac{n_k!}{2^{n_k}}$$

■

C.7. Symmetry Angles (Section 4.4.3)

THEOREM 4.4 : *The minimal angle between a reference weight vector* $\mathbf{w}_r$ *and any other weight vector* $\mathbf{w}$ *occurs if and only if both weight vectors lie in the same symmetry region, assuming that neither vector lies on a symmetry boundary.*

PROOF : *Let* $\mathbf{w}_t$ *be the symmetric copy of* $\mathbf{w}$ *that lies in the same symmetry region as* $\mathbf{w}_r$, *and* $\mathbf{w}'_t$ *be another symmetric copy of* $\mathbf{w}$ *in any of the adjacent symmetry regions. Consider the triangle* $(\mathbf{w}_t, \mathbf{w}_r, \mathbf{w}'_t)$ *which is formed by the points in high dimension space defined by these three weight vectors. The adjacent plane of symmetry dividing the two symmetric copies must intersect this triangle perpendicular and equi-distant between the boundary defined by* $\mathbf{w}_t$ *and* $\mathbf{w}'_t$, *by definition. The other boundary intersection must occur such that* $\mathbf{w}_r$ *lies on the same side of the symmetry boundary as* $\mathbf{w}_t$, *as they are in the same symmetry region. Thus, provided that none of the weight vectors lies on the symmetry boundary, the distance to the non symmetric copy will always be larger,*

$$|\mathbf{w}_t - \mathbf{w}_r| < |\mathbf{w}'_{t'} - \mathbf{w}_r|$$

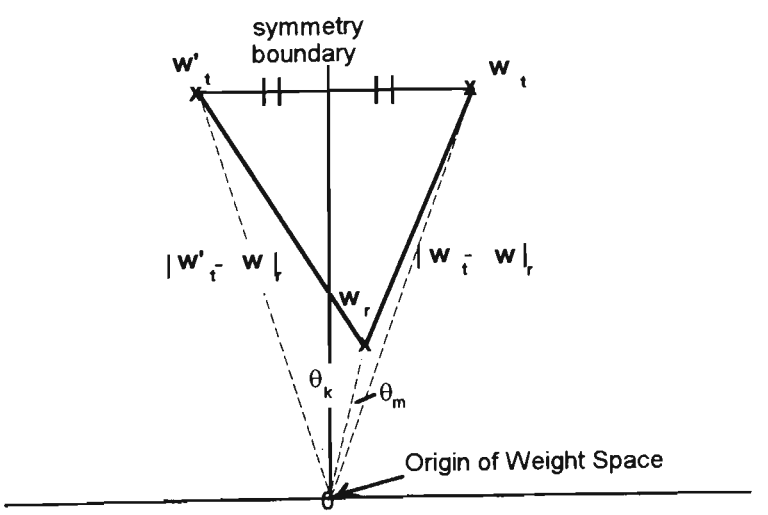
This inequality always implies that the angle θ_m *between these two vectors in the same symmetry region will always be smaller than the angle between the reference vector and a vector in an adjacent symmetry region* θ_k,

$$E(\mathbf{w}_r, \mathbf{w}_t) < E(\mathbf{w}_r, \mathbf{w}'_t) \quad \textit{(see 4.6)}$$

$$\textit{thus } \theta_m < \theta_k.$$

For any symmetric region which is not directly adjacent, θ_k *will be still larger and so the inequality still holds.*

■



Diagramatic representation of weight space symmetry regions.

THEOREM 4.5 : *The set of angles Φ between all symmetric weight vectors of $\mathbf{w}_t$ and a given reference weight $\mathbf{w}_r$ contain no equal angles provided that $\mathbf{w}_r$ is neither equidistant between any two adjacent symmetry regions nor lies on any symmetry boundary.*

PROOF : *Consider a symmetric copy of $\mathbf{w}_t$ that lies in the same symmetry region as $\mathbf{w}_r$. The closer $\mathbf{w}_t$ is to any one symmetry boundary the further away it is from all other adjacent symmetry boundaries, it is then at least double this distance from all other symmetric copies of $\mathbf{w}_t$. Now if $\mathbf{w}_t$ does not lie exactly equidistant between any two symmetry regions or on a symmetry boundary, then none of the angles between any symmetric copies of $\mathbf{w}_r$ and $\mathbf{w}_t$ can be equal either. Thus all angles in Φ will be unique.*
■

APPENDIX D

Ensemble Optimised Transfer Pseudo Code

This appendix contains pseudo code which may provide assistance in understanding some of the algorithms described in this thesis. The code is based on implementations of the algorithms written using Matlab™, and hence uses a similar notation for matrix references.

There are two main routines for the prior knowledge transfer. One for adding weight vectors to the prior knowledge data (`Create_Prior`) and the other to calculate the new optimised weight vector for a given set of training data (`Ensemble_Optimised_Transfer`). Some minor details of these algorithms have been omitted to preserve readability.

For environments where the detection of transformations on the training data is not attempted the functions `Detect_Transforms` and `Correct_Transforms` may be omitted. For transformation seeded weight space one needs to change `Create_Prior` so that for each new solution added to the prior knowledge the other transformations of this solution are also added.

Comments are preceded by a percentage sign and language constructs are shown in bold. In the simulated annealing code 'Δ' is used to denote the numeric change in a variable from one iteration to the next.

D.1

```
function Create_Prior

% Inputs
%    Weights : The set of weights to be added to the prior knowledge
%    Prior_Knowledge : Any previously created Prior Knowledge
%    (Optional) Training_Data : Training Data used to create Weights
% Outputs :
%    Prior_Knowledge : The new Prior Knowledge with Weights added
%

% Break the symmetry in the incoming weights
if Prior_Knowledge not empty
   NewWeights ← Break_Symmetry(Prior_Knowledge, Weights)
end
% If needed, detect transformation invariance in problem domain
Transform ← Detect_Transforms(Training_Data)
% ... then correct in weights
NewWeights ← Correct_Transforms(Transform, NewWeights)
% Add the weights to the prior knowledge
Prior_Knowledge ← [Prior_Knowledge  NewWeights]
```

D.2

```
function Ensemble_Optimised_Transfer

% Inputs :
%    Training_Data : Training data for which a solution is needed
%    Prior_Knowledge : Available Prior Knowledge
%    Iterations : Iterations of Algorithm
% Outputs :
%    New Weights : The new weight set found using the prior knowledge
% Constants :
%    Fusion_increment : Increment for fusion table result
%    max_fusion_table_size : Maximum number of entries in Fusion Table
```

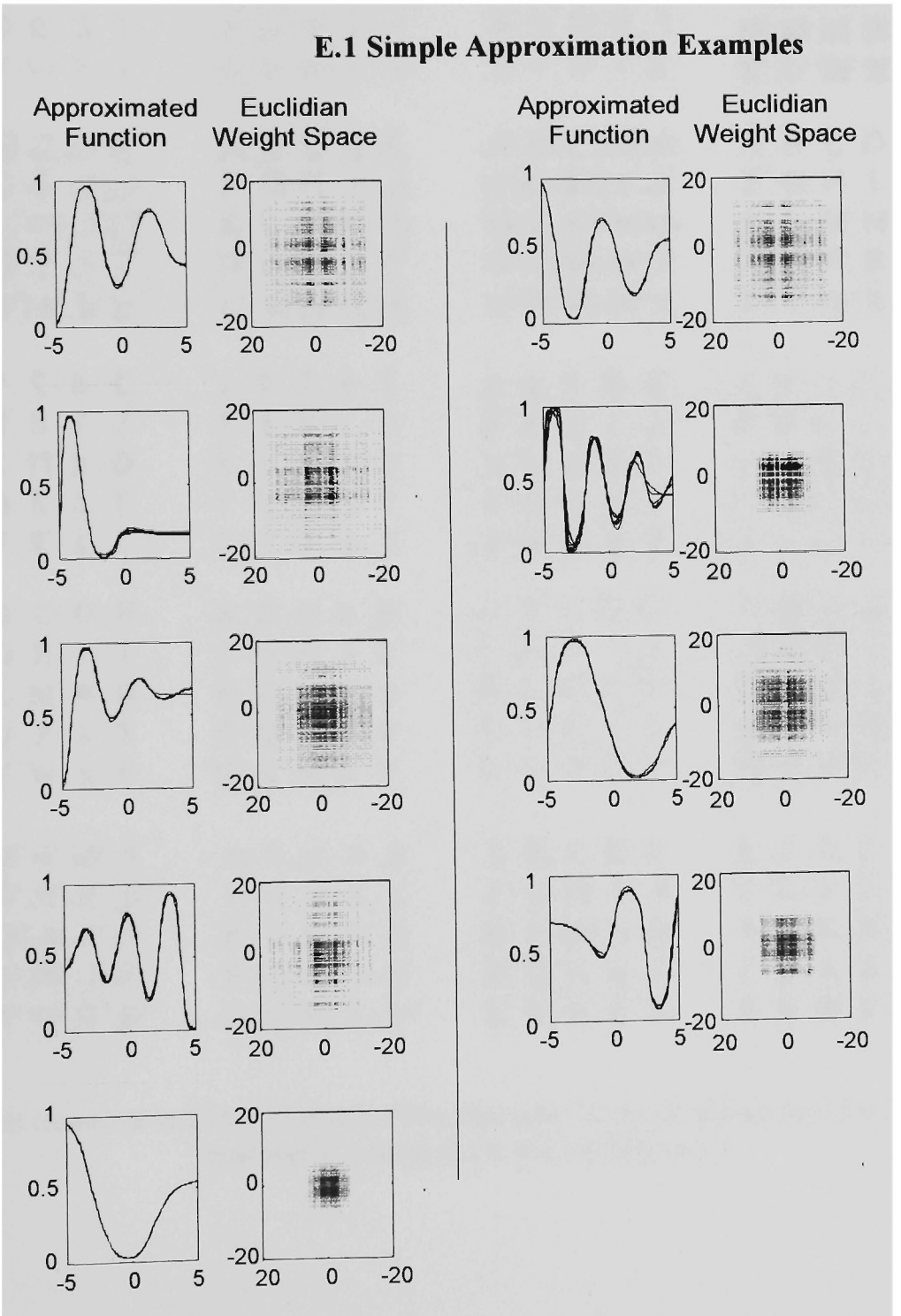
```
Transform ← Detect_Transforms(Training_Data)
for all Prior_Knowledge,
    % transform prior knowledge solutions to
    % be transform invariant if needed
    NewWeights ← Correct_Transforms(Transform, NewWeights)
    % Calculate response
    Error ← Calculate_Network_Response(NewWeights, Training_Data)
    % Add to an array of network errors
    Error_Table  ← [Error_Table Error]
end

% find lowest error subset of solution and return prior knowledge
Prior_Knowledge ← sort ( Prior_Knowledge, Error_Table )
Min_Prior_Knowledge ← Prior_Knowledge[1..max_fusion_table_size]

% simplex mixing of networks
mixture ←
    simplex_search(Min_Prior_Knowledge,Training_Data,Iterations*0.5)
Fusion_Maxtix ← create_tensor (Min_Prior_Knowledge, mixture)

freq_table= (1/Error_Table) / sum (1/Error_Table)

% zero temperature annealing of fusion matrix
Best_Fusion_Matrix ← Fusion_Maxtix
for 1 .. iterations*0.5,

     % stochastic increase in one fusion table entry
     neuron_num  ← rand(1..max_neurons)
     prior_num ←  freq_table [rand(1)]
     Fusion_Matrix[neuron_num, prior_num] ←
        Best_Fusion_Matrix[neuron_num,prior_num] + fusion_increment
     % make sure prior solution column sums to one
     Best_Fusion_Matrix  ← renormalise(Best_Fusion_Matrix)
     % Determine new weights
     NewWeights ←  Cross_Product( Fusion_Matrix,Min_Prior_Knowledge )

     % new network error
     Error ← Calculate_Network_Response( NewWeights, Training_Data )

     % if its an improvement keep the change
     if Error < last_Error,
         Best_Fusion_Matrix ←Fusion_Maxtix
     end
end
NewWeights ←  Cross_Product( Best_Fusion_Matrix,Min_Prior_Knowledge )
```

D.3

```
function Break_Symmetry(Prior_Knowledge, Weights)

% Inputs :
%    Prior_Knowledge : Prior Knowledge all in same sym region
%    Weights : Weights to be moved to this symm region
% Outputs :
%    New_Weights : New weights in same symm region as Prior Knowledge
% Constants :
%    Inital_Temperature : The Initial Temperature
%    alpha : Temperature decay

% get a set of symmetry broken reference weights
% from prior knowledge (could be any stored solution so take first)
RefWeights ← Extract_Weights(Prior_Knowledge,1)

Temperature ← Inital_Temperature
iterations ← 0   % number of iterations
accept ←  0;     % number of accepted temperature changes

NewWeights ← Weights ;

while iterations - last_accept  < max_itr_accept,

   iterations = iterations + 1 ;
   if  iterations > max_iterations
     return
   end;

   % swap two random neurons in the same layer
   RefWeights  ← swap_neurons(NewWeights)

   % angle between vectors
   Angle ← Calc_Angle(RefWeights, Weights)

   % determine annealing probability
   anneal_probablity  ← exp(-Δ Angle / Temperature);

    if Δ Angle > 0 or rand<anneal_probability,
      % accept new angle
      NewWeights  ← RefWeights;
      last_accept  ← iterations;
      accept  ←  accept +1;
```

```
      if accept = rate,
          accept ← 0;
          Temperature=alpha*Temperature; % decrease temperature
      end
    end
end
```

APPENDIX E
TRAINING EXAMPLES

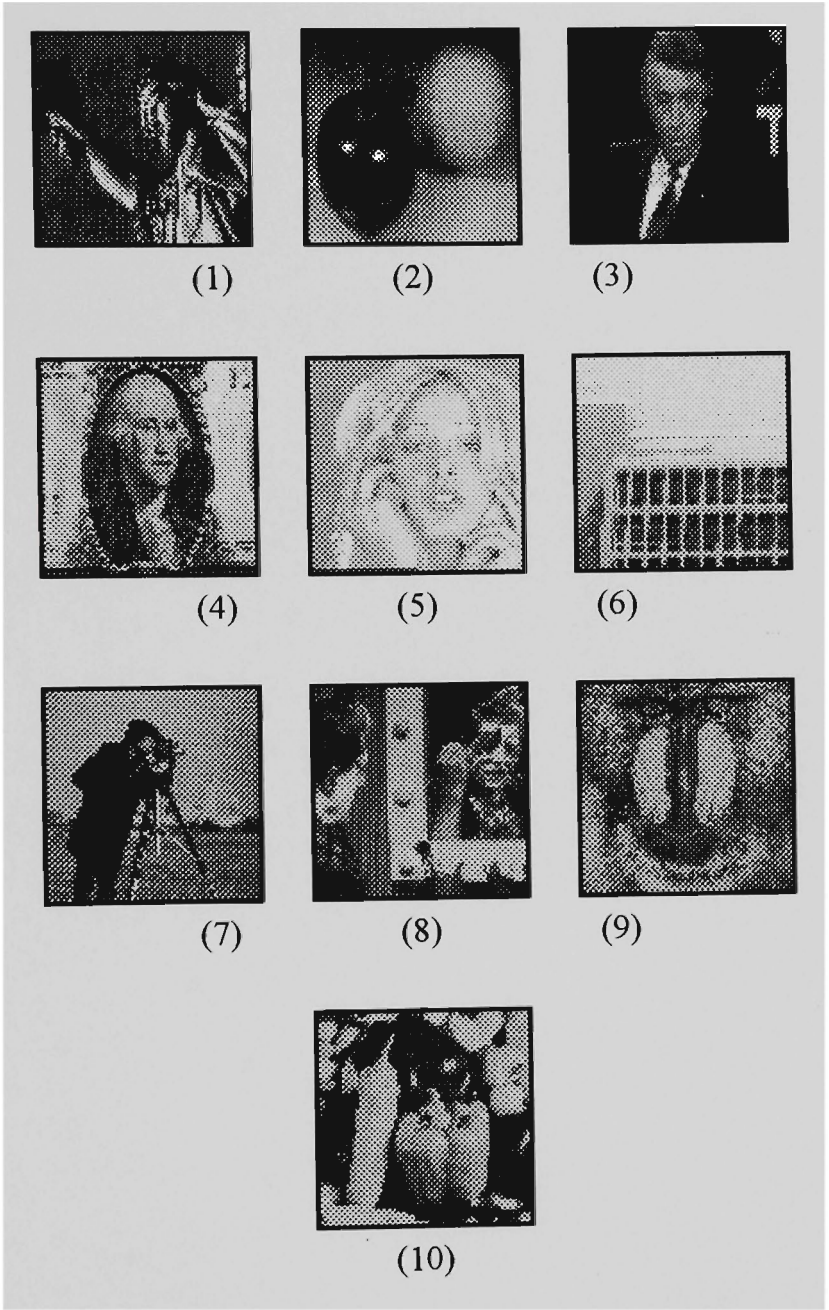


Fifty solution vectors are shown per problem plus associated weight shadows.

E.2 Character Recognition Training Data

Each character is 23 by 23 pixels. (The character 'Z' is not shown here for clarity, however it is included in the training sets.)

E.3 Surface Learning Training Data



Training Data For the Surface Learning Networks.