AN INVESTIGATION OF FEEDFORWARD NEURAL NETWORKS WITH RESPECT TO THE DETECTION OF SPURIOUS PATTERNS

A THESIS SUBMITTED TO

THE UNIVERSITY OF KENT AT CANTERBURY

IN THE SUBJECT OF ELECTRONIC ENGINEERING

FOR THE DEGREE

OF DOCTOR OF PHILOSOPHY.

By

Germano Crispim Vasconcelos

July 1995

Abstract

This thesis investigates feedforward neural networks in the context of classification tasks with respect to the detection of patterns that do not belong to the same categories of patterns used to train the network. This refers to the problem of the detection and/or rejection of *spurious* or *novel* patterns.

In particular, the multilayer perceptron network (MLP) trained with the back-propagation algorithm is examined in this respect and different strategies for improving its performance in the detection of spurious patterns are considered. The problem is investigated from different points of view that vary from the modification of the multilayer perceptron network with different configurations that make it more intrinsically able to detect spurious information, to the introduction of novel auxiliary mechanisms which, when integrated with the MLP network, can provide an overall enhancement in the system's rejection capabilities.

These different network configurations are examined with respect to the characteristics of the decision regions constructed by the networks in 2-D classification problems, and the implications of these constructions for general pattern rejection are discussed. The technique of inversion in multilayer networks through gradient descent is used to observe the degree of visual correlation between the input patterns recognised as valid by the networks and training class prototypes. Practical experiments on the classification of handwritten characters are employed as a test environment for the different approaches described.

Radial basis function networks (RBFs) are also examined in the same context and an experimental comparison is made between RBFs and the different MLP configurations studied.

Publications Arising From This Work

- Vasconcelos, G.C., Fairhurst, M.C., and Bisset, D.L. (1995). *Efficient detection of spurious inputs for improving the robustness of MLP networks in practical applications.* Neural Computing & Applications 3(4), 202-212, Springer-Verlag.

- Vasconcelos, G.C., Fairhurst, M.C., and Bisset, D.L. (1995). *Investigating feedforward neural networks with respect to the rejection of spurious patterns.* Pattern Recognition Letters 16 (2), 207-212.

- Vasconcelos, G.C., Fairhurst, M.C., and Bisset, D.L. (1994). *Recognizing novelty in classification tasks.* NIPS'94 Workshop on Novelty Detection and Adaptive Systems Monitoring, Vail - CO, USA.

- Vasconcelos, G.C., Fairhurst, M.C., Bisset, D.L. (1994). *Reliability of multilayer perceptron networks for spurious pattern rejection.* Proc. 1^{st} Brazilian Symposium on Neural Networks, 29-34, Caxambu - MG, Brazil.

- Vasconcelos, G.C., Fairhurst, M.C., and Bisset, D.L. (1993). *Enhanced reliability of multilayer perceptron networks through controlled pattern rejection.* Electronics Letters 29 (3), 261-263.

- Vasconcelos, G.C., Fairhurst, M.C., and Bisset, D.L. (1993). *The guard unit approach for rejecting patterns from untrained classes.* Proc. 1993 World Congress on Neural Networks (WCNN'93), IV 256-259, Portland - OR, USA.

- Vasconcelos, G.C., Fairhurst, M.C., Bisset, D.L. (1993). *Investigating the recognition of false patterns in backpropagation networks.* Proc. 3^{rd} IEE International Conference on Artificial Neural Networks. Brighton, U.K..

Contents

List of Tables

List of Figures

Chapter 1

Introduction and Background

This thesis reports on the investigation of strategies to improve the robustness of neural network computational architectures by seeking to enhance the reliability with which they process and categorise input stimuli which cannot be considered to belong specifically to any of the classes existing a priori in the pattern environment of the networks.

The field of neural networks is an area of study in computational intelligence that was originally conceived as a consequence of the observation that the human brain performs its function in an entirely different way from conventional serial computing paradigms. The basic idea in neural networks is that a large number of processing elements each computing a simple mathematical function, but highly inter-connected one to another, can exhibit a high computational power. To the processing units is attributed the name of "neurons" and to the whole structure of interconnected elements is given the name of "neural networks" [27, 29, 33, 44, 66, 68].

Neural networks have been used with success in many diverse areas of scientific and technical disciplines including computer science, engineering, physics, medicine, cognitive science, neurophysiology and human perception.

Many models have been investigated from many different points of view that vary from the observation of the biological plausibility of the model's properties in comparison with the information processing mechanisms found in the brain to the evaluation of these models simply as computationally efficient methods that are able to solve many real world problems.

With respect to the latter characteristic, one of the practical areas in which neural networks have found extensive application is in pattern recognition where examples of systems developed at an industrial and commercial level such as for speech processing, image recognition and optical character recognition are widely available [67].

1.1 Motivation

Until relatively recently, neural networks developed for pattern recognition had been mainly evaluated with respect to issues like the time required to train the system, the representational capacity of the model and, above all, with respect to their ability to generalise over the training patterns so that similar patterns presented later to the network during its usage phase are still correctly classified by the network.

More recently, however, another important characteristic in assessing the performance of neural networks in certain practical applications has gained a lot of interest from the research community. This is related to the fact that in many situations, the method used to solve a particular problem needs to be able to differentiate between normal conditions of the environment in which the model operates and situations when these conditions reflect abnormal courses of behaviour, indicating that something is not happening as should be expected, or was previously defined, to happen. For example, if a neural network is trained to recognise a certain number of classes previously identified during the training phase, it should be able to detect later during its usage phase any pattern that differs significantly from the patterns belonging to the valid classes. In other words, the network should be able to identify the input as being completely *novel* or as a *spurious pattern of information*, never seen before.

A simple but very illustrative example of the importance of the problem mentioned above is given by Smieja and Muhlenbein [75] and is referred to as the "dog-paw test". The test reflects the hypothetical situation of a system designed for handwritten digit recognition with a pen-based device and a drawing board used to give the input information to the system. Suppose that this system is a neural network that provides a very high recognition performance with respect to the classes of digits used to train the network. A problem which then arises considers the situation where the system is left unattended in a certain place and an animal (e.g. a dog) inadvertently puts its paw onto the input board. The question that is now put is "what is going to happen" ? Clearly, the ideal response of the system would be that the paw of the dog is very different from anything that was used to train the system and therefore cannot be accepted as a valid input pattern. Although this action of response sounds like the natural procedure to be carried out, in many current neural network models there is no evidence to guarantee that this is actually what is going to happen if the problem occurs. In fact, there is not even any clue at all of what kind of output is going to be provided by the network.

Another environment where the problem of the detection of spurious patterns is seen as a very important task is in many diagnostic or monitoring systems. For example, sensors distributed throughout a plant may be used to monitor the condition of operation of a whole system, or of specific parts of the system, by the measurement of many characteristics. These variables are constantly monitored to see if they lie within their range of acceptable level of operation. If something unexpected happens the system has to be able to detect it and to sound an alarm so that appropriate remedial actions can be readily executed. This is of particular concern in some safety-critical applications where it is important that any unexpected abnormal event is readily detected.

It is therefore of considerable practical importance to be able to construct systems which are inherently able to decide when input patterns are genuine members of the a priori known classes and when they simply correspond to spurious pattern classes, never seen before. This characteristic can also be seen as an important point of reference in deciding about which neural network model is most appropriate to use in a particular application, depending on its inherent natural ability to deal with the occurrence of abnormal patterns of information. Networks which present a poor capability in this respect can then be modified or used in conjunction with an additional mechanism devised with the specific purpose of turning them into more reliable systems.

One of the first studies found in the neural network literature to investigate, in some detail, the spurious pattern problem was carried out by Linden & Kindermann (1989) [42]. The work points out the fact that one of the most used and successfully applied neural network models in pattern recognition applications, the multilayer perceptron trained with backpropagation [66], can classify with high confidence patterns with completely random characteristics as if they were as legitimate as the most genuine representative members of the training classes. Linden & Kindermann also presented an approach to improve the ability of the multilayer perceptron to detect spurious information, normally referred to as *the negative training approach*. The same idea of the approach was later re-proposed by Smieja & Muhlenbein (1992) [75] as part of their reflective neural network architecture, and Bromley & Denker (1993) also re-used the same method [10].

Vasconcelos et al (1993) have presented an alternative method to be used with the multilayer perceptron with the property of not being dependent on the use of negative examples to improve spurious pattern rejection capability. The approach has been shown to work in the particular case of random patterns [78] and it was also shown to scale up well to more practical cases [79]. This technique will be described in Chapter 3. Modifications to the standard multilayer perceptron configuration presented by Vasconcelos et al (1994) [80, 81] have also shown how the rejection capabilities of the network itself with respect to spurious patterns can be improved. Chapter 4 will present these modifications.

More recently, several techniques with different characteristics have been proposed, studied under the general name of *novelty detection* approaches. One of the possible strategies is developed from a statistical point of view and is based on the estimation of the probability density function of the training data. These methods are based on variations of, or share many similarities with, the approach of radial basis function networks [7, 52]. Some of these techniques employ semi-parametric methods using kernel functions such as a mixture of Gaussians which represent a hidden layer of nodes to estimate the density of the data (Roberts & Tarassenko (1994) [64]). A test input is detected as novel if it does not approximate sufficiently the centers represented in the kernel functions. Bishop (1994) [6] shows how an approach based on Parzen windows for estimating the density of the training data can be employed for the detection of spurious patterns and another similar idea was proposed by Leonard et al (1993) [41].

A further strategy with a different characteristic is presented by Japkowickz et al (1995) [36] and is based on the use of the *autoencoder* type of neural network [66] to reconstruct at the output layer the positive instances of the patterns presented at the input layer, and the use of this autoencoder to recognise novel instances. The idea is that an autoencoder performs data compression on the input space and patterns presented at the input layer which are not seen by the network during the training phase are poorly reconstructed at the output layer. The method has been tested on some practical tasks but it still has shown the need for the use of negative examples of spurious patterns so that the system can separate them from the valid pattern classes.

Other examples of methods applied to very practical applications such as fault detection of helicopter gearboxes and fault detection of helicopter power train are now available. Some of these methods detect novel patterns through the previous modeling of the possible faults that might occur when the system is in use (Chin et al (1994) [14] and Kazlas et al (1994) [37]) or through an unsupervised learning method (Jammu & Danai (1995) [35]) similar to the principle employed in the guard unit technique of Vasconcelos et al [78] described in Chapter 3. The employment of a recurrent network to predict the temperature of an industrial distillation column process has also been illustrated by Ploix & Dreyfus (1994) [58]. The model is used to identify any discrepancy between the predicted and the actual temperatures so as to monitor faults in the operation of the system.

Yet another method, described by Courrieu (1993) [12], uses the definition of the *convex hull polytope* of a cluster or the definition of the circumscribed sphere to encapsulate the training data. A self-organising structure that presents properties similar to the ART models [11] has also been investigated by Bairaktaris [1] which uses modifiable thresholds to support a continuous adaptation regime to novel classes. Smyth (1994) [77] has studied the monitoring of communication systems with hidden Markov models (HMM) and their use in conjunction with neural networks to model both the known and unknown states of a system, where the unknown states are defined a priori or are represented in a $(m+1)$ state whose probability is determined through a Bayesian approach.

The main objective of this thesis is to study the class of feedforward neural networks, in particular the multilayer perceptron network, in the context of classification tasks with respect to the detection of spurious patterns. One of the aims of this work is to investigate possible modifications applied to the multilayer perceptron in order to transform it into a structure that presents an overall better performance when dealing with spurious information. Another strategy investigated, and one which has a fundamentally different nature, is the use of mechanisms in combination with the multilayer perceptron so that an integrated and more reliable architecture is obtained as a result of the combined structure.

One very important aspect to note about the solutions examined in this thesis is that they follow the basic principle that any enhancement in the spurious pattern detection abilities of a model should, to be of real practical benefit, be based only on the information provided by the patterns in the training classes rather than, as suggested by some methods, on the employment of negative examples of possible spurious classes to define the boundaries between the valid and the invalid classes.

Before passing on to the description of the chapters, it is opportune to mention that the issue of the spurious pattern problem is treated in the methods described in this thesis as a question of the *rejection* of spurious patterns. However, the observations and conclusions made are perfectly in accordance with the more general concept of the identification of spurious information as simply a different category from those previously known. It is clear that there are practical situations in which the interest remains in the grouping of these different categories instead of in their simple rejection.

It is also important to add that the investigation carried out in the present work makes use of the processing of visual patterns, in the practical application of the classification of handwritten characters, as a test environment. However, the ideas and concepts are completely general and can be extended to the processing of patterns of different natures. This type of application is used because it provides the benefits of testing network performance in the context of a widely recognised practical problem where many models of neural networks have been applied.

1.2 Overview of the Thesis

This thesis is organised in seven chapters in total, each investigating the problem from a different perspective. In this chapter, the motivation for carrying out this research is described together with a review of all the work recently carried out that has a relation with the subject of the thesis. Fundamental background information about the multilayer perceptron trained with backpropagation is also presented, since this is the principal model studied in the thesis.

Chapter 2 begins explaining the reasons for the inadequacies of multilayer perceptron networks in dealing with spurious patterns and a technique of iterative inversion of multilayer networks is described as an important tool for visualising the appearance of the patterns confidently classified as valid by the network. The technique first proposed to overcome the spurious pattern problem, the technique of negative training, is also investigated in this chapter and experiments in the application of the classification of handwritten characters are used to assess its performance. These experiments are employed in all the other chapters of the thesis as a means of comparison between the different models studied.

Chapter 3 introduces the idea of an auxiliary mechanism integrated and parallel to the standard multilayer perceptron, the *guard unit mechanism*, developed with the specific purpose of dealing with spurious information. The main objective of this chapter is to show how the combination of two networks driven by different purposes can bring practical benefits in terms of an enhanced overall classification performance.

In Chapter 4, different approaches are considered for transforming the architecture of the multilayer perceptron network itself into a more inherently suitable network for rejecting patterns different from the training classes. It is shown how each one these modifications can alter the decision regions created by the network in a pattern classification task and the implications of these modifications for spurious pattern detection are discussed.

In Chapter 5, the model of a feedforward network known as *radial basis function networks* (RBF) is described and this type of network is compared to the network configurations discussed in Chapter 4. The reason for this comparison is the fact that RBF networks appear as very natural candidate structures for the reliable rejection of spurious inputs and, therefore, their comparison with the standard multilayer perceptron as well as to the other versions of the MLP considered in the thesis offer a realistic way of examining the possible advantages and disadvantages of the different approaches.

Chapter 6 considers the problem of detecting spurious patterns from a very different perspective when compared to the previous chapters. It introduces a mechanism based on the ideas of *bootstrapping* for continually modifying the responses of a network across the pattern space. It is shown how this mechanism can, through "on line" adaptation, gradually enhance a network's ability to reject spurious patterns. Practical experiments carried out with the mechanism show that it need not only be used in conjunction with the standard multilayer perceptron but it can also be integrated with the other configurations described in Chapter 4.

Finally, Chapter 7 presents overall conclusions on the contributions of this overall programme. This chapter also discusses possible future investigations to be carried out as a continuation of the present work.

Providing the initial starting point for the thesis, the next section describes the single-layer perceptron model as well as the more general and powerful version that evolved from it, the multilayer perceptron trained with backpropagation. The information presented here is not intended to be a complete and exhaustive description of all the aspects of the model and only the most relevant information for its operation understanding is considered. For a very complete discussion of the features of the perceptron and of the multilayer perceptron see Rumelhart et al [66] and Haykin [27].

1.3 The Perceptron

One of the first and simplest forms of feedforward network was proposed by Frank Rosenblatt in 1962 [65] and was called the *Perceptron*. This model had a great influence in the historical development of neural networks as a research field and was proposed for the purpose of pattern classification.

In the simplest form, the perceptron is a model composed of an input layer, a single layer of connections and an output layer of processing elements of the type of neuron proposed by McCulloch & Pitts [49], as a simplified model of the biological neuron. For ease of explanation, the discussion in this section will be restricted to a perceptron with a single output node since the extension to the case of more than one neuron is a straightforward matter.

The operation of the neuron is described as a linear combination of its inputs applied to the connection weights and its comparison to a threshold, followed by the application of the result as the input to a linear threshold function (a hard limiting function) (f). Denoting the inputs of a neuron by $x_1, x_2, \ldots, x_n$, the connection weights by $w_1, w_2, \ldots, w_n$ and the threshold by θ, Figure 1.1 presents the schematic representation of the neuron.

The objective of the neuron is to classify the input into one of two classes C_1 and C_2 depending on the value of the weighted sum of the inputs ($net = \sum_{i=1}^{n} w_i \cdot x_i - \theta$). If net is positive then the input is classified as belonging to class C_1 and the neuron produces an output equal to $+1$ and if it is negative then the corresponding class is C_2 and the neuron gives an output of -1. The classification operation executed by the neuron creates a separation boundary between the two

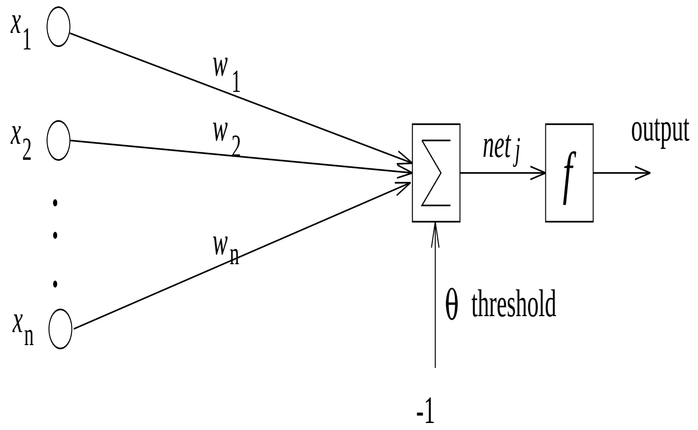


Figure 1.1: Single perceptron

classes represented by a *hyperplane* which is defined when $\sum_{i=1}^{n} w_i \cdot x_i - \theta = 0$. In a typical classification problem involving more than two classes the number of processing nodes in the perceptron can be increased so as to represent the different classes.

The most important aspect about the operation of the perceptron is how the set of weights $w_1, w_2, \ldots, w_n$ are estimated. This is achieved by the application of an error-correcting scheme known as the *perceptron learning algorithm*. Given a sample of training patterns from classes C_1 and C_2 presented to the perceptron, the procedure adjusts iteratively the weights of the network after each pattern presentation. For the case where a pattern is correctly classified no correction is made to the weights and it is only when a misclassification of a pattern occurs that the weight vector is updated. Hence, at a given iteration t, if pattern $\vec{x}_p$ in the training set is correctly classified by the weight vector $\vec{w}(t)$ no correction is made to the weight vector, as defined by :

$$\begin{cases} \vec{w}(t+1) = \vec{w}(t) & \text{if } \sum_{i=1}^{n} w_i \cdot x_i - \theta \geq 0 \text{ and } \vec{x}_p \text{ belongs to } C_1 \\ \vec{w}(t+1) = \vec{w}(t) & \text{if } \sum_{i=1}^{n} w_i \cdot x_i - \theta < 0 \text{ and } \vec{x}_p \text{ belongs to } C_2 \end{cases}$$

Otherwise, if pattern $\vec{x}_p$ is incorrectly classified, weight $\vec{w}(t)$ is updated according to the rule :

$$\begin{cases} \vec{w}(t+1) = \vec{w}(t) - \eta\vec{x}_p & \text{if } \sum_{i=1}^{n} w_i \cdot x_i - \theta \geq 0 \text{ and } \vec{x}_p \text{ belongs to } C_2 \\ \vec{w}(t+1) = \vec{w}(t) + \eta\vec{x}_p & \text{if } \sum_{i=1}^{n} w_i \cdot x_i - \theta < 0 \text{ and } \vec{x}_p \text{ belongs to } C_1 \end{cases}$$

where the term $0 < \eta \leq 1$ controls the rate of adaptation.

The interesting characteristic about this learning scheme, shown by Rosenblatt in his perceptron learning convergence theorem [65], is that, given that there exists a solution weight vector which separates the pattern set, there is a guarantee that the perceptron learning algorithm will always converge to it in a finite number of iterations. This guarantees that a solution of global minimum error is obtained if such a weight vector exists. The very limiting aspect of the perceptron, however, is the fact that it can only represent input/output associations of patterns that come from a linearly separable set. For example, it is shown that for the simple binary XOR function there is no set of weights that can be found to separate the input patterns into the proper sets to define the function.

This drawback was the most important criticism raised by Minsky and Papert in their classic book *Perceptrons* [53] which actually culminated in a great pessimism about the future of the neural networks field in the seventies. Although it has long been understood that the limitations of the perceptron only apply to networks with a fixed architecture of a single layer and that the use of *intermediate* layers could make the network able to compute more complex functions, it took several years to renew the interest in the area with the development of an efficient learning algorithm called the *error-back propagation algorithm* or the *generalised delta rule*, for training perceptrons with multiple layers.

1.4 Multilayer Perceptron and Backpropagation

The very basic idea of the backpropagation algorithm is reminiscent of the work of Paul Werbos in 1974 described in his PhD thesis [83]. However, it was only later in 1986 with Rumelhart, Hinton and Williams that the algorithm became widely popularised through the classic neural network book *Parallel Distributed Processing* [66]. A similar generalisation of the algorithm was also developed independently by Parker in 1985 [56] and another algorithm of similar characteristics was presented by LeCun in 1985 [39].

The multilayer extension of the perceptron model, commonly referred to as the *multilayer perceptron (MLP)*, consists of a set of input units (sensory units) constituting an input layer, one or more intermediate layers (hidden layers) of processing elements and an output layer of processing elements, as illustrated in Figure 1.2.

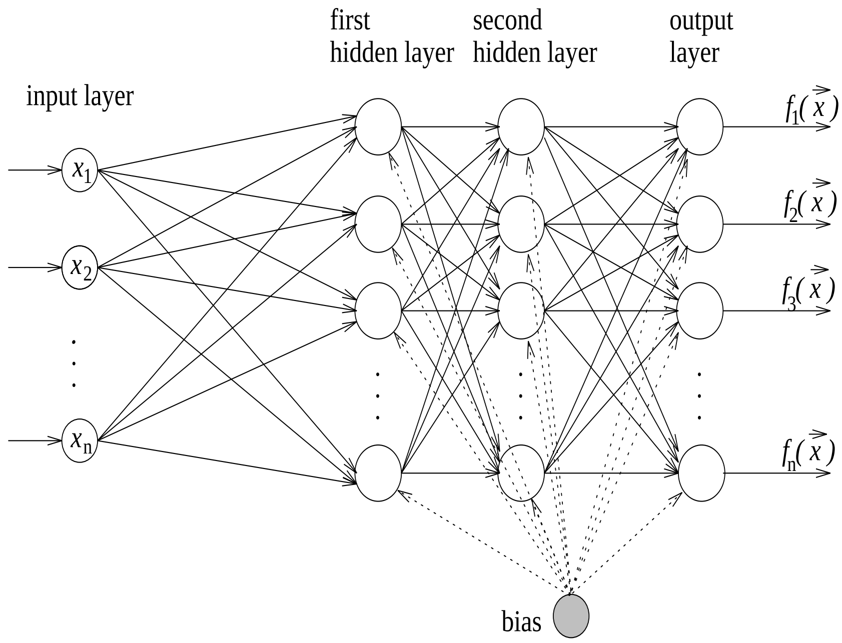


Figure 1.2: Multilayer perceptron network

Multilayer perceptrons have been successfully applied in many real world applications such as the learning of the pronunciation of English text [69], optical character recognition [38], speech recognition [62], steering of an autonomous vehicle [60], radar target detection and classification [28], control [84] and medical diagnosis of heart attacks [4]. One important result about this network is that a configuration with only a single hidden layer of processing elements has been shown to be able to uniformly approximate any continuous function [15], providing the model with the property of universal approximation.

The operation of the network has two distinct passes through the various layers. In a first pass, the forward pass, an input pattern is presented in the sensory units of the network (input layer) and the processing units compute, layer by layer, activation functions until a set of outputs is finally obtained at the output layer of the network. The backward pass begins with the measuring at the output layer of the error observed between the actual output produced by the network and a given desired output. The errors are then propagated back, hence the name error-back propagation, from the output layer to the input layer in a layer by layer basis and the weights at each layer are adjusted so as to minimise the difference between the network's current output and the target output.

The backpropagation algorithm is in fact a method for implementing *gradient descent* in weight space for training a feedforward network. This implies that the objective of the method is the efficient computation of the partial derivatives of a certain function $F(\vec{w}; \vec{x}_p)$ implemented by the network (which approximates the function desired to solve the task) with respect to the components of the weight vector $\vec{w}$ for a given input vector $\vec{x}_p$. The employment of these derivatives for adjusting the weights of the network guarantees the minimisation of the error at the network's output.

There are basically two modes for updating the network's weights, *per sample training* and *batch training*. In per sample training the changes are performed after the presentation of each training example and in batch training the weights are updated only after each presentation of the whole sample of training patterns.

Although in principle either of the two methods can be employed, it has been shown that the choice of which strategy to choose depends on the particular task [29].

One of the consequences of having a gradient descent procedure for training a multilayer perceptron is that it introduces, in contrast with the single perceptron, the existence of multiple local minima of the error function in addition to global minima. This is the case of any procedure based on "hill climbing". The shape of the error function can present many valleys and while during the training process the network can reach any one of these valleys, it is possible that not far from that point a deeper valley with a better minimum could have been obtained. The issue of the effect of local minima on the practical use of the backpropagation approach is still not completely explained but Rumelhart et al [66] claim that this is in fact rarely a practical problem. This observation has been in some sense corroborated by the many practical applications in which the multilayer perceptron has been successfully applied.

Defining the Generalised Delta-Rule

The operation of each processing unit u_j in the network is defined by a *propagation rule* representing the activation of the unit with respect to its inputs (o_{pi}), where $o_{pi} = x_i$ if unit u_i connecting to unit u_j is an input unit,

$$net_{pj} = \sum_i w_{ji} o_{pi} \tag{1.1}$$

and by a semi-linear *activation function* f_j which provides the output of the unit :

$$o_{pj} = f_j(net_{pj}) \tag{1.2}$$

The error at the output layer of the network is given by :

$$E_p \;=\; \frac{1}{2}\sum_{j=1}^{n}(t_{pj} \;-\; o_{pj})^2. \tag{1.3}$$

where o_{pj} is the actual output of unit u_j and t_{pj} is the desired output.

In order to minimise the network error, weight changes are defined according to the derivative of the error function E_p with respect to the weights w_{ji},

$$\Delta_p w_{ji} \;\propto\; -\frac{\partial E_p}{\partial w_{ji}}$$

This derivative is then rewritten using the chain rule as the product of two factors, one representing the change in error as a function of the change in the net_{pj} input to the unit and the other representing the effect that changing a particular weight has on the net_{pj} input,

$$\frac{\partial\, E_p}{\partial\, w_{ji}} \;=\; \frac{\partial\, E_p}{\partial\, net_{pj}}\,\frac{\partial\, net_{pj}}{\partial\, w_{ji}}. \tag{1.4}$$

Now, through Equation 1.1 the second factor in Equation 1.4 results in :

$$\frac{\partial\, net_{pj}}{\partial\, w_{ji}} \;=\; \frac{\partial}{\partial\, w_{ji}}\,\sum_{k}\, w_{jk}o_{pk} \;=\; o_{pi}.$$

With respect to the first factor in Equation 1.4, which reflects the change in the error as a function of the change in the net_{pj} input, it can be defined as :

$$\delta_{pj} \;=\; -\frac{\partial\, E_p}{\partial\, net_{pj}}, \tag{1.5}$$

and Equation 1.4 finally becomes :

$$-\frac{\partial\, E_p}{\partial\, w_{ji}} \;=\; \delta_{pj} o_{pi}. \tag{1.6}$$

which says that for reducing the value of E_p (the implementation of gradient descent) the changes in the weights have to be proportional to $\delta_{pj} o_{pi}$ or, in other words :

$$\Delta_p w_{ji} \;=\; \eta \delta_{pj} o_{pi} \tag{1.7}$$

The only variable that remains to be calculated is the value of δ_{pj} for each one of the units. Making use of Equation 1.5, and again of the chain rule, δ_{pj} can be written as :

$$\delta_{pj} \;=\; -\frac{\partial\, E_p}{\partial\, net_{pj}} = -\frac{\partial\, E_p}{\partial\, o_{pj}} \frac{\partial\, o_{pj}}{\partial\, net_{pj}} \tag{1.8}$$

Looking first at the second term in this equation, it is obtained from Equation 1.2 that :

$$\frac{\partial\, o_{pj}}{\partial\, net_{pj}} \;=\; f_j'(net_{pj}), \tag{1.9}$$

which is the derivative of activation function f for unit u_j.

Considering now the first factor in Equation 1.8, two cases have to be considered. For the case where u_j is an output unit it is seen according to the definition of E_p :

$$\frac{\partial\, E_p}{\partial\, o_{pj}} = -(t_{pj} - o_{pj}),$$

that the value of δ_{pj} is given by :

$$\delta_{pj} = (t_{pj} - o_{pj}) f'_j(net_{pj}) \tag{1.10}$$

In the case where unit u_j is not an output unit, the values of δ_{pj} can only be calculated through a recursive procedure where the errors at a given layer are estimated as a combination of the errors in the following layer. The chain rule is applied to define :

$$\frac{\partial\, E_p}{\partial\, o_{pj}} = \sum_k \frac{\partial\, E_p}{\partial net_{pk}} \, \frac{\partial net_{pk}}{\partial o_{pj}}$$

which using Equation 1.1 yields :

$$\frac{\partial\, E_p}{\partial\, o_{pj}} = \sum_k \frac{\partial\, E_p}{\partial net_{pk}} \, \frac{\partial}{\partial o_{pj}} \, \sum_i w_{ik} o_{pi}$$

and using Equation 1.5 produces :

$$\frac{\partial\, E_p}{\partial\, o_{pj}} = -\sum_k \delta_{pk}\; w_{jk}. \tag{1.11}$$

Finally, substituting Equation 1.9 and Equation 1.11 in Equation 1.8 it is obtained :

$$\delta_{pj} = f'_j(net_{pj}) \sum_k \delta_{pk}\; w_{jk} \tag{1.12}$$

for any unit u_j that is not an output unit.

It is seen that Equation 1.12 together with Equation 1.10 provide a procedure for computing all the δ_{pj} values and consequently the weights in the network can be modified according to Equation 1.7. The errors have to be calculated in the output units first and then propagated back to the units in the preceding layers so that their weights can also be altered. This method constitutes the generalised delta rule for a feedforward type of neural network where the processing units compute semi-linear activation functions.

It was seen that in the case of the single layer perceptron the units compute a linear threshold function. This function can no longer be applied in the perceptron with multiple layers since a continuous differentiable function is necessary for the generalised delta rule to work. The reason for this is that the computation of the δ values for each unit in the multilayer perceptron requires the derivative of the activation function to be calculated and this therefore imposes the constraint that the function should be continuous. The most commonly encountered continuous differentiable activation used in the multilayer perceptron is the *sigmoidal nonlinearity* in its various forms. One of these forms is the *logistic function* :

$$o_{pj} = \frac{1}{1 + exp(-net_{pj})} \tag{1.13}$$

where the output of the function lies in the range [0,1],

while another very common choice corresponds to an asymmetric form with respect to the origin known as the *hyperbolic tangent* :

$$o_{pj} = tanh(-net_{pj}) \tag{1.14}$$

in which case the outputs of the units fall in the range $[-1, 1]$.

The choice of which function to use depends upon each particular application. In some cases one of the two may be preferable to the other but in many cases either of them can be applied.

Considering the logistic function, the derivative is easily obtained by differentiating both sides of Equation 1.13 with respect to the net_{pj} input :

$$\frac{\partial o_{pj}}{\partial net_{pj}} = f'(net_{pj}) \tag{1.15}$$

$$= \frac{exp(-net_{pj})}{[1 + exp(-net_{pj})]^2}$$

Making use of the original equation of the activation function (Equation 1.13) this expression can be rewritten eliminating the exp term, and it becomes a very simple equation :

$$f'(net_{pj}) = o_{pj}(1 \; - \; o_{pj})$$

which, as can be seen, depends only on the output of the processing unit for its calculation.

One important characteristic of the sigmoidal function is that its derivative $f'(net_{pj})$ reaches its maximum when $o_{pj} = 0.5$ and its minimum when o_{pj} is at one of its extreme values ($o_{pj} = 0$ or $o_{pj} = 1$). This means that, since the adaptation of the weights of a unit is proportional to the value of its derivative, the units that will change most are the ones which have not been committed yet to respond with an output close to either 0 or 1. This characteristic of the function is pointed out by Rumelhart et al [66] as an important factor for providing some stability to the error-back propagation algorithm.

Generalisation

One of the major features of neural networks, and particularly of the multilayer perceptron, that has made them so popular in many practical applications is their inherent ability to generalise over the training classes and to identify successfully input patterns similar (but not identical) to the ones seen during the process of training. The use of hidden layers in the multilayer perceptron makes possible the extraction of features represented in the connection weights which are naturally defined by the network as "useful" for differentiating between patterns from different classes. Patterns are mapped to classes according to the criterion of "most similar to" and, consequently, this introduces flexibility to classify correctly patterns that appear similar to the training patterns but are not exactly the same. However, this characteristic is also responsible for the multilayer perceptron not having any limit to the kinds of patterns that can be accepted as valid. In other words, the network is not good at extrapolating from the training data.

The issue of improving the generalisation performance of a network may be viewed from two different perspectives [27]. In the first, the architecture of the network is fixed a priori and the problem that remains is the estimation of the training set size needed to obtain a good generalisation performance. Theoretical estimations have been derived to determine appropriate training set size based on the number of connection weigths in the network but this serves only a guidance purpose since generalisation is very much problem dependent and, in practice, the number of patterns necessary to be used is many times smaller than the theoretical number. From the second point of view, the size of the training set is fixed and the issue that remains is to find the best network architecture that can provide a good generalisation. In this case, techniques known as *network-pruning techniques* have been proposed where the objective is to improve generalisation by the minimisation of the network size [27]. This is based on the concept that a network with a minimum size is less likely to learn noise or anomalies in the training data and will tend therefore to generalise better to unseen data. In any situation, another technique useful for improving generalisation is the statistical tool of *cross-validation*. This procedure has become more widely used more recently and consists of defining a validation set of patterns in addition to the training set, which is used to evaluate the performance of the network during the evolution of training. Training is then stopped at a point where the best generalisation performance is provided with respect to the validation set and the network can now be evaluated on the test set. One of the benefits of cross-validation is to avoid the problem of *over-fitting* where the network simply "memorises" the training data, which can result in poor generalisation capabilities.

Variations of Backpropagation

One of the major criticisms of the multilayer perceptron trained with backpropagation which relates to the stochastic nature of the learning process is that it requires many repetitions of the data set in order to allow the network to learn the appropriate task. A great deal of effort has been expended on research into the development of different strategies for speeding up learning in multilayer perceptrons. One of the simplest approaches, and one which is often very effective, is the introduction of a *momentum term* into the rule for adapting the weights [66]. The method consists of giving some importance to past weight updates for the modification of the weights at the current step. Equation 1.7 is modified to :

$$\Delta_p w_{ji}(t+1) \;=\; \eta \delta_{pj} o_{pi} + \alpha \Delta_p w_{ji}(t) \tag{1.16}$$

where α is the momentum parameter.

The idea with this modification is to give the weights some momentum (or inertia) so that they tend to move in the direction of the average "downhill trend" in the gradient descent process instead of oscillating in direction from one side to the other when looking for a solution to the problem. This happens because gradient descent can oscillate widely if the weights adaptation rate (η) is too large and it can, on the other hand, be very slow if η is too small. The inclusion of the momentum term aims at achieving a balanced course of adaptation.

Many other more complex methods have also been investigated, each of which focuses on a different aspect of the adaptation process. One natural path to explore is to have the learning rate (η) and the momentum parameter adjusted dynamically during training and/or to have these parameters separated for each weight. Some examples of methods development following this principle and some other ideas are the Delta-Bar-Delta method [34], the RPROP algorithm [63], the Quickprop algorithm [20], the Search-Then-Converge technique [16] and Silva & Almeida's adaptation by sign changes [72]. Other methods that have also been proposed are based on the calculation of second order derivatives of the cost (error) function E_p for optimising the learning scheme such as the conjugate-gradient method and the Newton's method. A detailed description of all these techniques and many more can be found in [27, 71] and an experimental comparison between them can found in [71].

1.5 Conclusion

This chapter has presented the initial motivations for the development of the work to be reported in this thesis and has provided some background information about the main model for the networks investigated. In the following chapters, a more detailed examination of the multilayer perceptron in the spurious pattern problem is initiated and strategies for enhancing the rejection performance of the model are considered.

Chapter 2

Recognition of Spurious Patterns

2.1 Introduction

In this chapter, the problem of the detection of patterns not belonging to the *a priori* defined training classes is investigated in the context of feedforward neural networks. More specifically, the multilayer perceptron network trained with backpropagation [44, 47, 66] is examined in this respect. The method referred to as the inversion of multilayer networks by gradient descent is described, which can be employed as a useful tool for the visualisation of the characteristics of the patterns confidently recognised by a network in a classification task.

The reasons for the inherent unreliability of the standard MLP in relation to the spurious patterns problem are explained and a technique for the enhancement of its rejection capabilities, known as the *negative training approach*, is described. It is shown, however, that this approach has limited practical use because it does not guarantee the definition of uniform decision boundaries encapsulating the valid training classes. The practical application of the classification of handwritten alphanumeric characters is considered, as an example, to examine the performance of the networks described.

Pattern recognition has been one the most successful areas in which neural networks have found practical application [7, 25, 32, 43, 47, 69]. In this application area, the models developed are frequently evaluated with respect to their ability to correctly classify patterns belonging to the classes on which they have been trained. However, a further important characteristic in assessing the use of neural networks in systems requiring high reliability is the ability of the model employed to deal with the occurrence of patterns that do not share real membership with any of the training classes present in the application domain. Many networks have been shown to provide very good classification performance when they are tested with patterns of the same classes as those in the training set but, until recently, little attention had been paid to the question of how these models will behave if a pattern completely dissimilar to the training examples is presented to the network.

The importance of developing appropriate methods for dealing with this kind of problem is particularly clear in applications where decision-making about the identity of the input patterns demands high reliability. In such situations, some form of rejection mechanism needs to be present in the environment either based on the built-in properties of the network itself, or through the introduction of an additional process integrated with the system.

It has been shown that one of the most effective networks used in pattern recognition, the multilayer perceptron architecture trained with backpropagation, can classify with a high degree of confidence random patterns as if they were authentic members of the trained classes [40, 42, 78]. Few solutions have been proposed to overcome the problem, and these have usually been based on the "negative training approach", an approach which makes use of negative examples of random patterns distributed through the pattern space, presented to the network during its training phase. This approach was first introduced by Linden &

Kindermann [42], subsequently investigated by Smieja & Muhlenbein [75] and also re-applied by Bromley & Denker [10]. Here, this approach is described and practical experiments are conducted to assess its efficiency. These experiments will also serve as a test of comparative performance with the other methods investigated in this thesis.

2.2 The Inversion of Multilayer Networks

Neural networks and other pattern classifiers have been compared with respect to many characteristics such as classification performance, training time, memory requirements and speed of classification, to mention just a few [26, 31, 40, 44]. Some results obtained in practical tasks have even demonstrated that under appropriate circumstances, such as the provision of a large enough training set, similar classification performance can be achieved by different classifiers such as the MLP network, K-nearest neighbours and radial basis function networks [40].

Another important aspect of system performance concerns the ability of the method employed to distinguish and reject patterns which are very distinct from the classes used to train the network. A comparative study developed by Lee in [40] on the practical application of the classification of handwritten characters has shown that while approaches based on K-nearest neighbour classification and radial basis function networks are quite capable of readily rejecting patterns with random shapes, the MLP network trained with backpropagation can accept these patterns, with a high degree of confidence, as valid members of the classes.

Williams in [87] and Linden in [42] have described how this phenomenon of finding random "false" patterns classified by the MLP network can be investigated through the technique of *network inversion*, after it has being submitted to a training phase. The method consists of clamping the network weights after it has been trained on a certain task and of modifying the input pattern initially fed into the input matrix through gradient descent according to the least-mean-square error between the current network output and a given output target. In other words, the input pattern is successively modified until the output reaches the target. Figure 2.1 illustrates a typical MLP network and the process of network inversion after training.

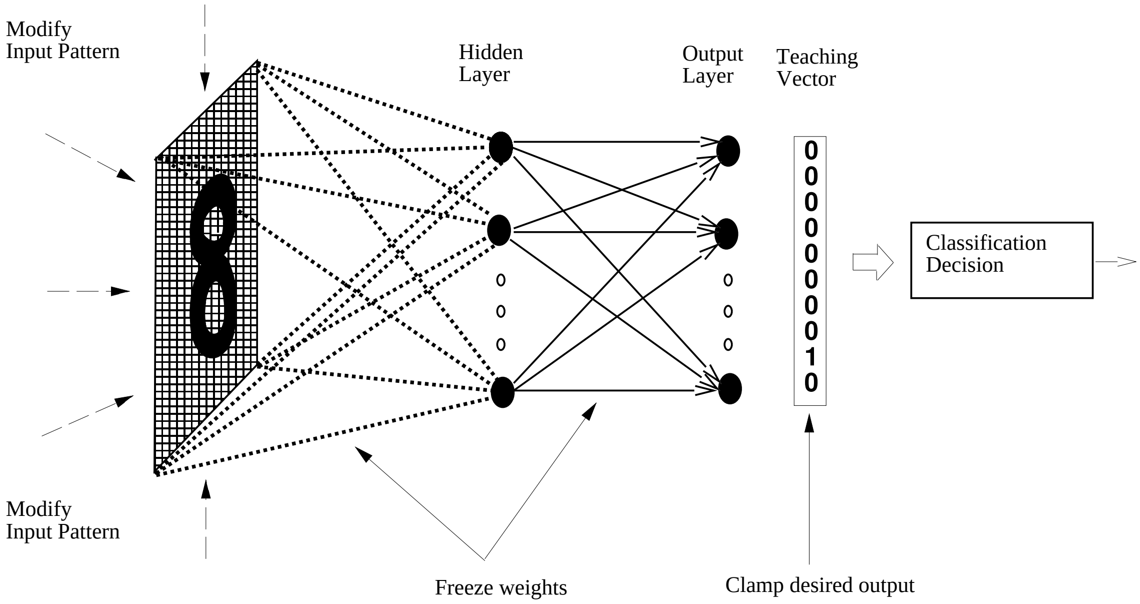


Figure 2.1: Standard MLP network and network inversion

Consider the problem of the recognition of handwritten characters. For example, if it is required to modify an initial random pattern in the input matrix in order for to it be recognised as a '8', the network's target output for class '8' is clamped into the network's teaching vector and, with the weights of the network frozen, the input pattern is modified until it has been correctly classified as belonging to class '8'. The process stops when the global error between the current network output and the target output is less than a specified value.

In what follows, a more mathematical and detailed description for the method is given.

Description of the Method

Let $\vec{x}_p$ and $\vec{t}_p$ be the input matrix and the target output vector of the network, respectively. The least mean square error (LMS) between the target (t_{pk}) and current (o_{pk}) outputs of the network is used as a means of searching for an input $\vec{x}_p$ which when presented to the network makes it respond approximately with the desired output $\vec{t}_p$. The LMS error is represented by :

$$E_{LMS} \quad = \quad \sum_{k \, \in \, O} (t_{pk} \; - \; o_{pk})^2.$$

Now, let $\vec{x}_p(0)$ and $\vec{o}_p(0)$ be the initial values of the input matrix and the output vector, respectively. In order to modify the input matrix, the outputs of the network are first computed by running a feedforward pass through the network :

$$\vec{o}_p(0) \quad = \quad F(\vec{x}_p(0)).$$

The error signals are then calculated at the output layer and backpropagated to the preceding layers until they reach the input layer (input matrix). For all input units i that represent the input vector $\vec{x}_p$, the error signals are given by :

$$\delta_{pi} \quad = \quad \frac{\partial \, E_p}{\partial \, net_{pi}} \tag{2.1}$$

In this equation, another component, net_{pi}, is introduced to prevent input activations increasing to arbitrarily large values. The reason for this is because if the derivatives of the error over the activations (a_{pi}) in the input vector were computed directly, they would eventually drive those activations to lie outside the hypercube, which constrains the input pattern space. In order to prevent this from happening, the input activations a_{pi} are clamped and each net_{pi} is computed by the inverse of the sigmoid activation function. In this case, used in the non-symmetric form :

$$net_{pi} = f^{-1}(a_{pi})$$

$$net_{pi} = -ln(\frac{1}{a_{pi}} - 1)$$

which is the inverse of the sigmoid function :

$$a_{pi} = f(net_{pi})$$

$$a_{pi} = \frac{1}{1 + e^{-net_{pi}}}$$

Using Equation 2.1, it is now possible to finally calculate the error related to each input unit and use it to modify net_{pi} by gradient descent, according to :

$$\Delta\ net_{pi} = \eta\ .\ \delta_{pi}$$

where η corresponds to the learning rate for modifying the input space.

According to the value of net_{pi} each input activation a_{pi} is computed again providing a new input vector. By proceeding in this way, the process is repeated iteratively several times creating a sequence of input vectors $\vec{x}_p(0)$, $\vec{x}_p(1)$, ..., $\vec{x}_p(n)$ which minimises the distance between the output vector and the target vector until it is considered sufficiently small. This may be expressed mathematically by the condition :

$$(\vec{t}_p(n) - \vec{o}_p(n))^2 < \epsilon \ \bigtriangledown .$$

Application of the Method

Following this description, experiments of network inversion were carried out with the standard MLP network after it had been trained on the problem of the classification of handwritten digits. The database used to train the networks corresponds to separate sets of machine printed and handwritten characters extracted from postcode information on envelopes in the UK mail.

Initial experiments were carried out using the database of machine printed characters. Several trials were simulated initialising the input matrix with different patterns. The initialisation procedure consisted, conveniently, of varying the density of 'on' and 'off' pixels in order to transform the input pattern in a very random manner. Other initialisation methods, such as for example the random combination of different parts of patterns from the training set, could also have been equally applied. None of these particular methods, however, is expected to have any different effect in the results obtained (see [40]). Figure 2.2 shows an example of input vector initialisation.

Figure 2.2: An example of input matrix initialisation

Two variations for the input hypercube were considered in the network. In one of the variations the activation values in the input matrix lie in the range $[0, 1]$ and, in the second, the activations lie in the interval $[-1, 1]$. For this second case, the sigmoid function is still the same but a small alteration in the inverse function is necessary in order to make the output values of the inverse function to fall in the interval $[-1, 1]$, instead of in the original interval $[0, 1]$. The inverse function is given in this case by :

$$net_{pi} \quad = \quad -ln(\frac{1 \quad - \quad a_{pi}}{1 \quad + \quad a_{pi}})$$

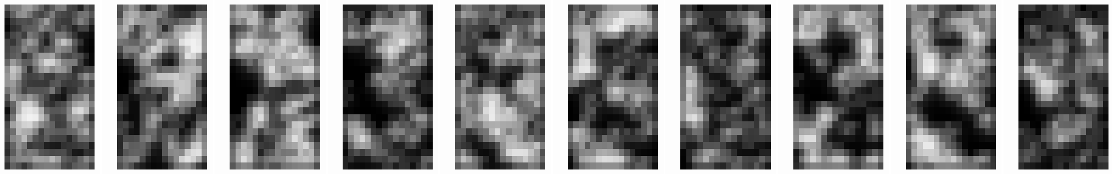
The test of these two variations had the objective of verifying whether or not the substitution of '0' by '-1' as input value would have any effect in the reduction of the problem of recognising false patterns. This was based on an initial supposition that because pixels in the input matrix with activation value '0' do not contribute to changes in the network's weights they could be responsible for under utilisation of the information present in the training data.

Figure 2.3 shows 20 patterns generated by inverting the network, the first row (a) corresponding to space $[0, 1]$ and the second (b) to space$[-1, 1]$. As can be clearly seen there is a high degree of deformation in those patterns recognised (from left to right) as belonging to the classes of handwritten digits '0' to '9'. There is also no visually discernible characteristic that could imply any difference in performance with respect to the two cases examined. As will be seen in the next section of this chapter, the reasons for the MLP network's unreliability are of a different nature.

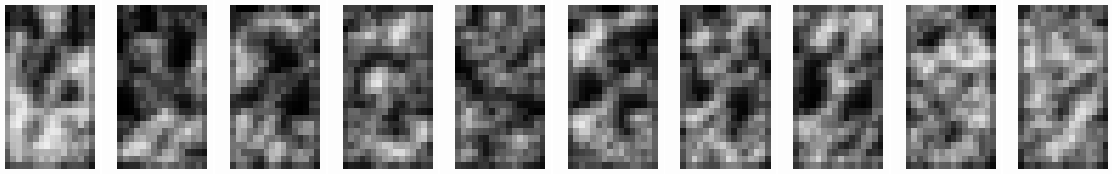
The same sort of experiment was repeated, this time, for the set of handwritten digits. In this case, patterns in the training database are even less uniform in their shape. Figure 2.4 displays examples of the inverted patterns obtained.

2.3 The Reasons for the MLP Unreliability

In order to understand the factors which influence the general rejection performance of a network in a classification problem and, consequently, the reasons for the acceptance as valid of patterns with completely random appearance, an examination of the network's operation at the level of the function computed by



(a)



(b)

Figure 2.3: False digits for an MLP trained with machine printed digits

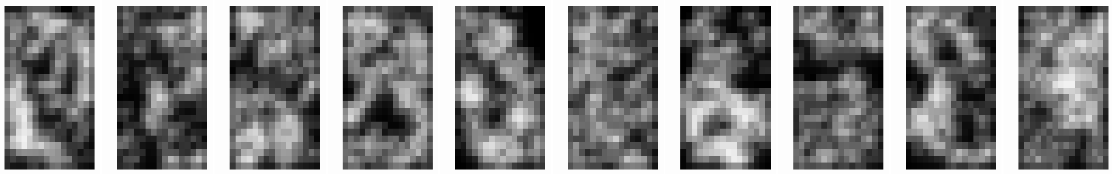


Figure 2.4: False digits for an MLP trained with handwritten digits

the processing units in the network is necessary. With respect to the standard MLP, one important feature is the fact that each unit in the network implements a global mapping through a non-linear discriminant function that divides the input space into two portions bounded only by the extreme limits of the input space. This is the result of the combination of the weighted sum of the inputs as the network's propagation rule ($net = \sum_j w_{ij}x_j$) and of the sigmoid as its activation function ($(1 + exp(-net))^{-1}$ or $tanh(net)$), which creates a global receptive field such as the one shown in Figure 2.5, for the 2-dimensional case. This property can bring advantages in terms of maximising generalisation capabilities, but together with the backpropagation learning algorithm, it is primarily responsible for the confident recognition of spurious inputs observed in MLP networks.

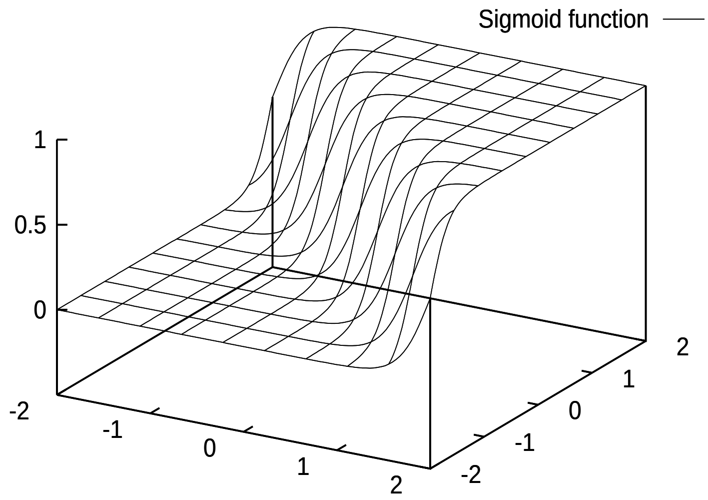


Figure 2.5: Combination of the inner product with the sigmoid activation function

The following experiment, which visualises the classification surfaces generated by a MLP network applied to a classification problem in the 2-dimensional space, provides a useful way to investigate network rejection capacity. The experimental procedure consists of training a one (or more) hidden layer(s) MLP with a sample of training patterns in a 2-class problem, and of testing its classification response for a large matrix of points covering the input space. The classification decision associated with each point in the matrix is determined in accordance with a *confidence level* imposed at the network's output. A pattern is classified as belonging to one of the 2 classes if the output (in the range [0,1]) for that class exceeds that of the other by a chosen confidence level; otherwise, the pattern is rejected.

The result of the experiment using backpropagation with a high confidence level (0.65), shown in Figure 2.6, demonstrates that the vast majority of the points, even those most dissimilar to the training patterns (i.e at the extreme edges of the input space), are still confidently classified by the MLP network (points represented by the elevated surfaces in the diagram). As can be seen, only the points falling in between the training data points are rejected (points situated in the valley of the diagram).

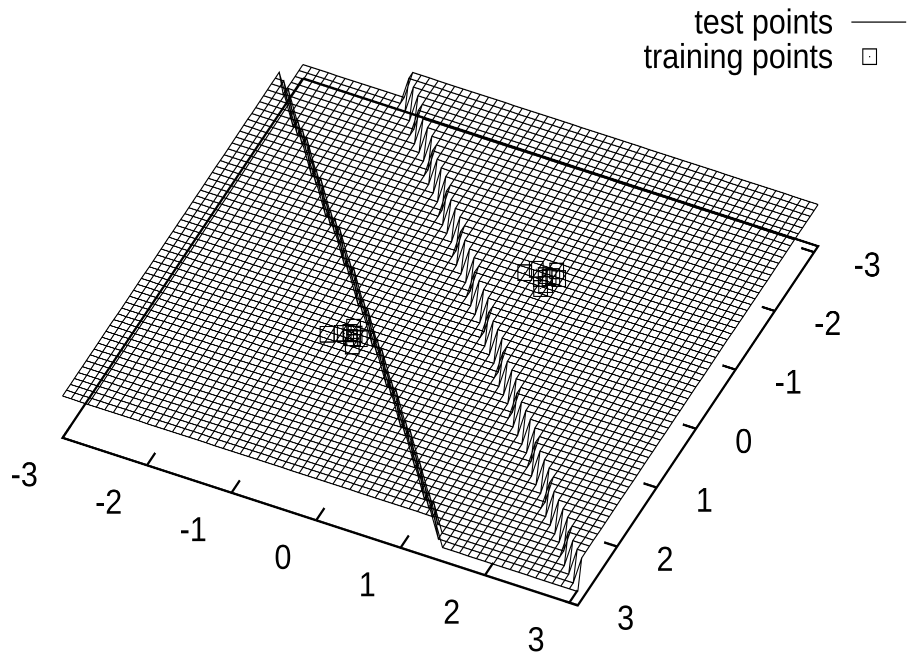


Figure 2.6: Decision regions for the standard MLP

The application of the same procedure, this time in a problem involving 9 classes, produces a similar result to that obtained for the 2-class problem in terms of the creation of open decision regions, as illustrated in Figure 2.7. This situation also demonstrates that the only rejection areas created in the pattern space correspond to the regions located in between the training classes, and any pattern falling outside these rejection areas is accepted as valid by the network.

The combined reasons for this are, first, that the discriminant function used by the processing units in the network corresponds to hyperplanes and, second, that the only constraint involved in the determination of the placement of the hyperplanes is the characteristic of the learning algorithm, which only places hyperplanes directly between classes rather than (as might be desired) around them.

This means that open decision regions are created in those parts of the input space for which no information is available.

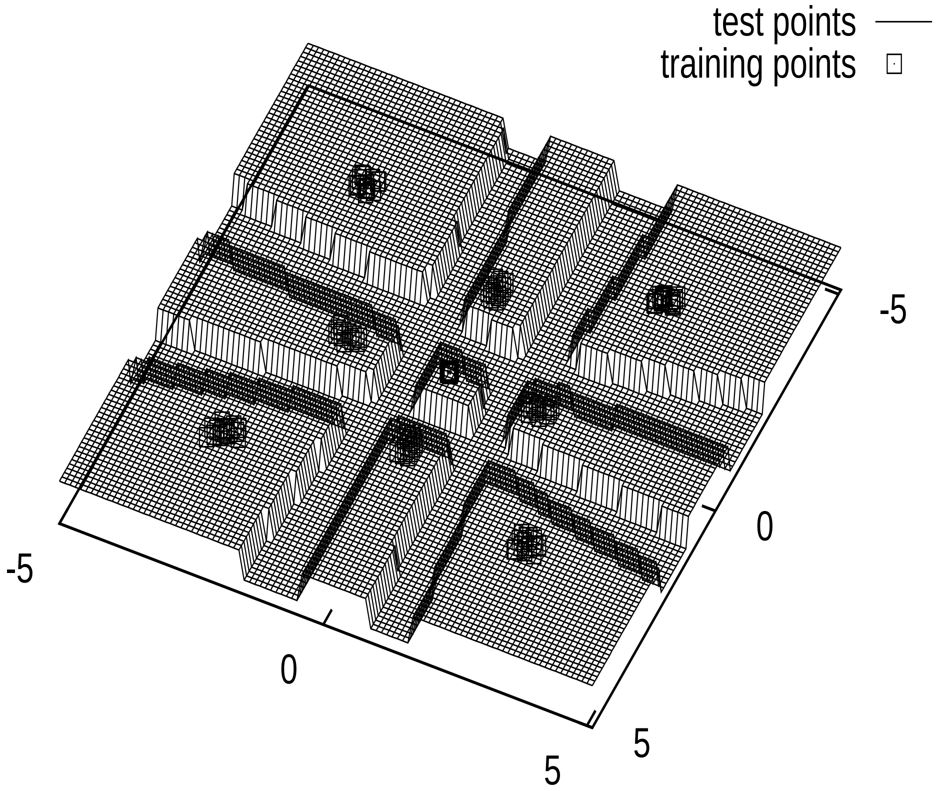


Figure 2.7: Decision regions for the standard MLP (9 classes)

This characteristic of standard MLPs trained with backpropagation is not always taken into account when practical applications are considered. Although MLP networks are known to be capable of creating any form of complex closed decision regions [43], this experiment demonstrates that this does not occur with backpropagation when the training data only represents specifically the classes to be learned, and hence does not provide information about the space *surrounding* the classes.

In many practical applications it is clearly important that spurious extra-class patterns are detected as such, and rejected. For example, in a factory process controlled by a neural network it is important to detect abnormal operating conditions so that the appropriate remedial action can be executed. It is therefore of considerable practical benefit to construct systems which are inherently able to decide when input patterns are genuine members of the known classes and when they simply correspond to spurious pattern classes.

2.4 The Negative Training Approach

One of the possible methods for improving the rejection capability of MLP networks consists of presenting "negative" examples of random patterns to the network during its training phase and of teaching the network to reject them [42, 75]. This can be done in one of two ways : through the introduction of an additional unit in the output layer of the network, representing the "rejection" class and mapping the random patterns to this unit, or through the minimisation of the responses of all the network's output units when a random pattern is presented to the network. The purpose of this approach is to create attractors in the pattern space generated by the random examples so that when another random pattern is presented it is more likely to be classified as "garbage" than as a valid member of the trained classes.

The problem with this method is that there is no guarantee that other patterns both different from the training classes and also from the negative examples will not be accepted by the network. The decision regions generated to separate the other parts of the input space from the training classes are not uniform and are very much dependent on the expectation that the random patterns arbitrarily chosen to represent unseen patterns correspond to good representatives of the portions of the input space desirable to be considered as rejection areas.

The effect of applying this method in the 2-class problem described above, in this case not with random patterns but with points carefully chosen to illustrate important characteristics, can be visualised in Figure 2.8. Here, a group of negative points surrounding the training classes is included in the original training set and the network is trained to reject these peripheral points. It is clear that the network is now able to generate closed decision regions because it has information about the whole input space. In real world applications, however, there is generally very little (or no) a priori information about the kinds of spurious patterns that should be rejected by the network and, therefore, there is great difficulty in finding an effective way of choosing the negative training examples.

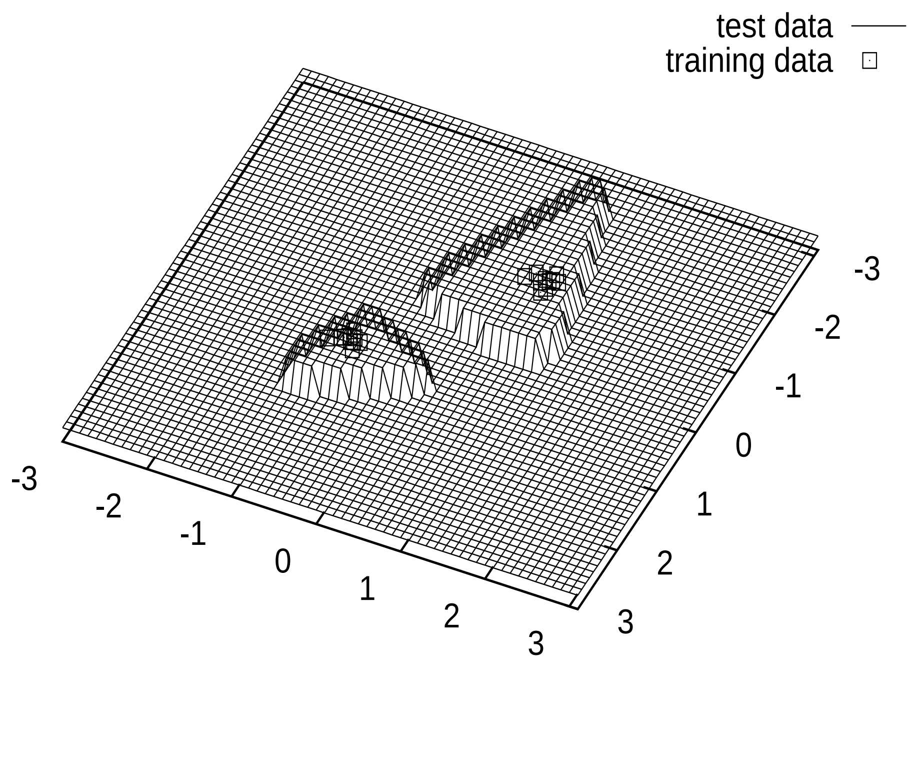


Figure 2.8: Decision regions for the MLP with negative training

2.5 Evaluating Negative Training

2.5.1 Network inversion

The process of network inversion can now be repeated to investigate the effects of the negative training approach on the tolerance of the network with respect to the input patterns classified as valid. As described earlier, the network is inverted after having been trained to recognise digits. In addition to the original training classes, random binary patterns are created on-line during the training process and are presented as examples of what the network should reject in its usage phase.

Inverted patterns accepted by the network trained on the set of machine printed digits can be visualised in Figure 2.9. In this case, if compared carefully to the results previously obtained with the standard MLP architecture, it is seen that some improvement in the overall appearance of the digits classified is introduced. However, it can be observed that the patterns accepted by the network are still very indistinct. There is not much contrast in the shapes of the patterns obtained so that they could clearly be identified as numerals.

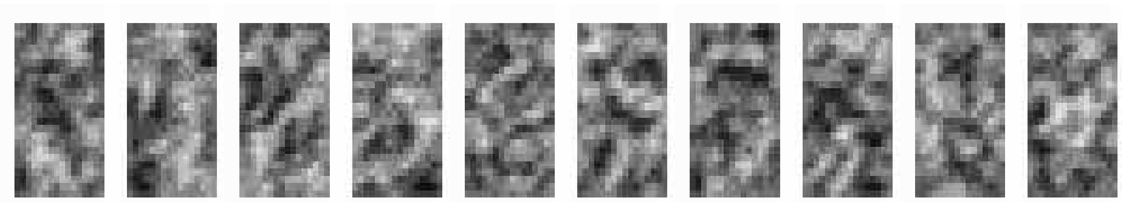


Figure 2.9: Inverted digits for the MLP with negative training

2.5.2 Experimental results

In order to further compare the approach of negative training (MLP^{neg}) with the standard MLP network in the rejection problem, extensive experimental work was conducted to measure recognition and rejection performances of both networks in the classification of handwritten characters. The experiments consist of training the networks to recognise handwritten digits, and of testing their performance in the rejection of spurious patterns using for this purpose alphabetic letters presented to the network during the recall phase.

The database used to train the networks is composed of 1000 handwritten digits and the recognition performance of each network is tested on an independent set of 2000 digits. Each character in the database is presegmented and size normalised as a binarised image matrix of 16 $\times$ 24 pixels. Hence, the input layer of the network is formed of 384 (= 16$\times$24) units, while the output layer consists of 10 processing units to allow discrimination among the 10 valid classes '0' through '9'. The network is fully connected from the input layer (input matrix) to the hidden layer and from the hidden layer to the output layer.

A second database consisting of 7800 handwritten alphabetic letters is used to test the network's ability to reject patterns not belonging to the training classes. Figure 2.10 shows typical samples of digits and letters present in the database to illustrate the general form and variability of the data used. The experiments carried out in this chapter, and most of the experiments carried out in the remaining chapters of this thesis, were implemented using the Rochester Connectionist Simulator [24] running on a Sun Sparc-2 workstation.

All the networks are single-hidden-layer networks tested initially with different numbers of units in the hidden layer. Tables 2.1 and 2.2 present the classification results obtained for networks with 80 hidden units. These were the network configurations further explored since they presented a slightly better performance than the others and they allow a more consistent comparison with other MLP configurations described in future chapters. The experiments are repeated with different values of the confidence level imposed at the network's output for acceptance of a classification decision. A pattern is classified as belonging to one of the

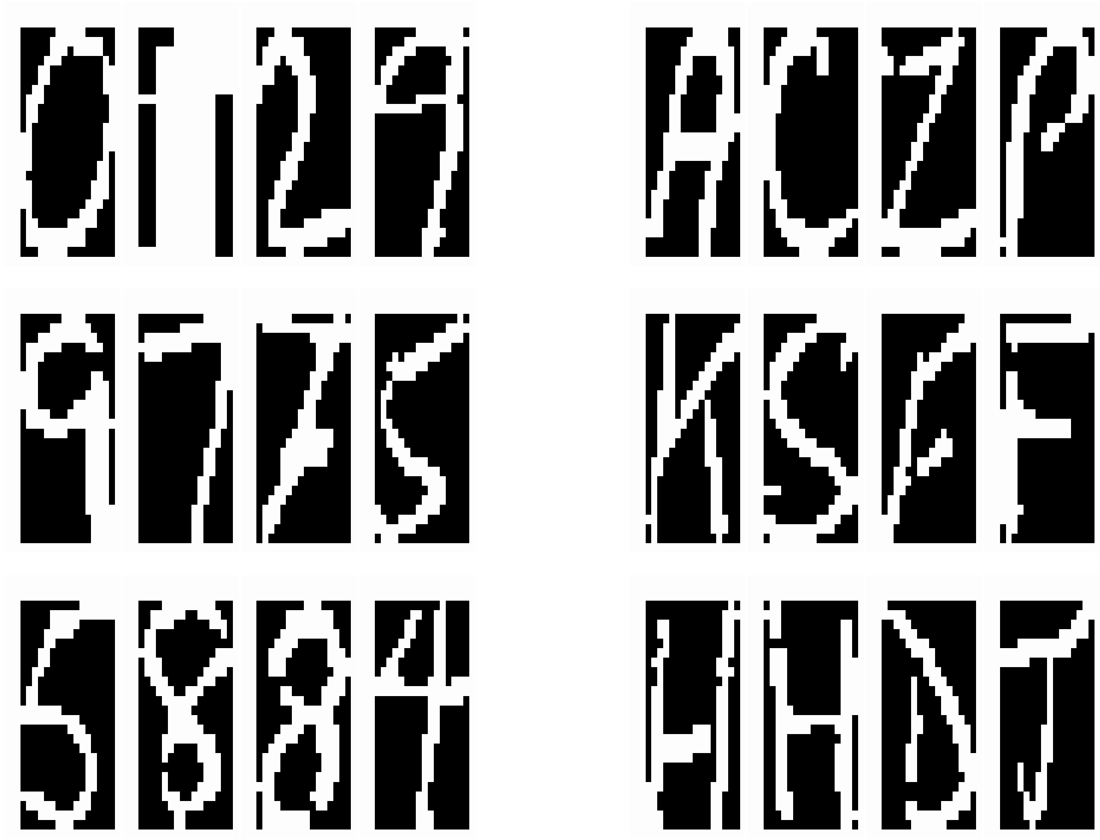


Figure 2.10: Handwritten digits and letters

classes if the output unit representing that class exceeds those of the others by the given confidence level. Otherwise, the pattern is rejected. Table 2.1 shows the correct recognition rates for the digits and also the proportion of letters rejected by the networks. Finally, Table 2.2 illustrates the percentage of digits erroneously classified together with the percentage of digits rejected by the network.

Although it would be more fair if the results obtained with the standard MLP could be compared to other types of classifiers in the same problem, it can be judged from Table 2.1 that its spurious pattern rejection performance is indeed very poor. For example, for the value of 0.45 for the confidence level, the network is able to reject only 46 % of the entire set of alphabetic letters and, in this case, its correct recognition for the digits drops from 92 % to 80.8 %. In another example, for the small value of 0.15, the network does not reject more than 17 % of the

	Digit Recognition Rate		Letter Rejection Rate	
confidence	MLP	MLP^{neg}	MLP	MLP^{neg}
0	92.0	91.5	0	0
0.15	89.4	87.2	17	28
0.25	88.2	86.2	25	36
0.35	84.4	80.3	32	48
0.45	80.8	79.4	46	56
0.55	81.3	80.2	48	53
0.65	72.9	78.4	64	60

Table 2.1: Digit recognition rate vs letter rejection rate

letters and also, for the other cases, the performance of the network cannot be considered satisfactory.

The results observed with the MLP^{neg} network show a relatively inconsistent pattern of behaviour. In some cases, a substantial improvement in the rejection performance is noticed when compared to the standard MLP, while in other cases only a small increase in the rejection rate is observed. In other instances no significant difference is apparent. This suggests that this approach is very dependent on the extent to which the negative random patterns chosen during the training of the network are representative of the parts of the pattern space considered as rejection areas. The experiments reported here correspond to the best case achieved with this approach. For example, for a confidence level of 0.15 the rejection rate is improved by 11 %, for a confidence level of 0.55 the improvement is by 5 %, whereas for a confidence level of 0.65 there is no improvement at all (indeed, in this case, there is actually a decrease in the rejection rate). With respect to the digit (valid class) recognition rates, some small degradation in performance is also introduced, as can be seen in Table 2.1. The average reduction is by 1.8 % for the first 6 values of the confidence level whereas for the value of 0.65 there is

	Digit Error Rate		Digit Rejection Rate	
confidence	MLP	MLP^{neg}	MLP	MLP^{neg}
0	8.0	9.5	0	0
0.15	5.3	5.7	5.3	7.1
0.25	5.3	4.3	6.5	9.5
0.35	3.7	3.5	11.9	16.2
0.45	2.6	2.2	16.6	18.4
0.55	3.0	2.6	15.7	17.2
0.65	1.7	1.7	25.4	19.9

Table 2.2: Digit error rate vs digit rejection rate

actually an increase in performance. With respect to the letter rejection rates, the average increase is by 10.5 % for the values of 0.15, 0.25, 0.35, 0.45 and 0.55 in the confidence level.

The other rates measured, the digit error rates and digit rejection rates described in Table 2.2, have shown in some of the cases some difference in the results observed (for example, for the values of 0.25, 0.45 and 0.55 of the confidence level) although this does not correspond to any substantial change in performance. It is important to note that for a network developed with the intent of creating more rigorous decision boundaries, separating the training classes from the other parts of the pattern space, there should be a tendency for an increase in the rejection of valid patterns but also a reduction in the rate of misclassified patterns, which is not very apparent in this case.

Another important point to emphasize is that this test of rejection performance is a strict one since letters are similar in many respects to digits. In all the experiments carried out it has been observed that certain pairs of numerals and letters such as 0/O, 1/I, 8/B, 5/S, 2/Z, etc. are very difficult to differentiate. In a practical situation, for example, it is impossible to differentiate between '0's (the numeral) and 'O's (the letter) unless some contextual information is provided. The results presented therefore reflect a particularly difficult task environment.

Figure 2.11 gives a broad picture of the likelihood of confusion between letter/digit combinations. This figure represents the confusion matrix for one of the simulations with the standard MLP network with the confidence level set to 0.25. In this representation a grey scale coding is used to indicate the degree to which patterns from a letter class are being accepted as belonging to a specified digit class. Here, a very dark square corresponds to a large proportion of inappropriate classifications, while a white square signifies a very small number of letters being accepted as the corresponding digit class. As expected, it is seen that some of the letter classes such as 'O', 'I', 'X', 'S' and 'Z' are strongly recognised by classes '0', '1', '1', '5' and '2', respectively. These classes accepted as genuine digits 288, 202, 208, 168 and 261 out of a total of 300 patterns present in the database of each letter class.

2.6 Conclusion

This chapter has introduced the problem of the recognition of spurious patterns and the MLP network has been investigated in this respect. The factors that influence the poor performance of MLP networks in the rejection of spurious information have been explained through the development of a very useful visualisation experiment for the 2-dimensional case. Practical experiments conducted on the classification of handwritten characters in the difficult problem of separating letters and digits have also been used to back up the observations made in the 2-dimensional problem. These experiments will also be employed in the next chapters to examine the performance of the different networks proposed.

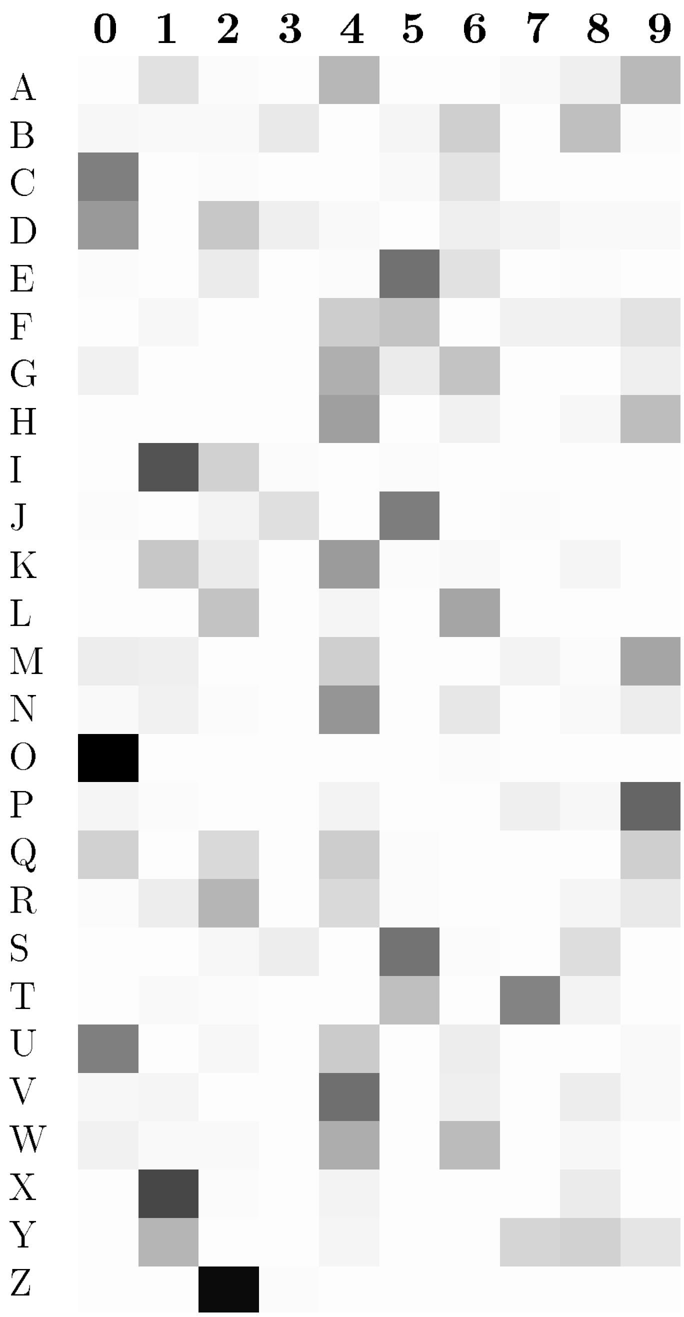


Figure 2.11: An example of confusion matrix

The technique of the inversion of multilayer networks through gradient descent was presented both because it is to be used as a tool in the following chapters of this thesis, and also to illustrate the problems related to the standard MLP architecture. This technique has proved to be a useful tool for the investigation of the tolerance of the networks with respect to the appearance of the patterns classified as genuine by the network.

The approach of negative training was also examined in this chapter and it is possible to observe that this method does not provide very reliable solutions in practical applications. This is due to the fact that the approach is very dependent on the assumption that the negative patterns used to train the network provide a good representation of the parts of the input space desired to be considered as rejection areas, and this is often not a valid assumption in practice.

Chapter 3

The Guard Unit Approach

3.1 Introduction

In this chapter, an approach for improving the performance of MLP networks with respect to the rejection problem is examined based on the use of an auxiliary network acting in parallel and integrated with the standard MLP architecture. This integrated network has the specific goal of rejecting spurious patterns and it is shown both experimentally and theoretically how it can work efficiently in this respect while maintaining the network's ability to classify valid patterns.

3.1.1 The guard unit approach

The idea presented in this chapter is that of basing the rejection of invalid patterns on the information provided *only* by the original patterns present in the legitimate training classes rather than on the information acquired by the presentation of "negative" examples (in contrast with the method of negative training presented in Chapter 2). The use of an independent set of processing units, called *guard units*, attached to the standard multilayer perceptron (MLP) network is proposed which have the specific objective of preventing patterns with arbitrary characteristics from being classified as belonging to the defined training classes. Each guard unit models an amalgamation of the patterns from the class that it represents through a very simple one-shot learning procedure during the network's training phase. Thus, during the network's recall phase, it accepts or rejects a pattern presented in the input matrix depending on whether or not it has a minimum similarity with that composite representation. The guard units are not responsible for deciding to which training class an input pattern might belong, that is the function of the MLP architecture. Instead, their function is more simply to classify the input pattern as a "valid" pattern or not.

The approach of guard units aims to combine the advantages of two different types of network each with its own characteristics. The first of these is the standard MLP, which has good generalisation capabilities as a result of, among other things, the feature based representations defined in the hidden layer(s) of the network. The second network is a single layer type structure, the guard unit part of the system, which generates template-like representations of the training classes and uses them to assess the similarity between the inputs and the training classes. While the standard MLP can provide good overall generalisation performance it is unable to prevent the classification of patterns distinct from the training classes. With respect to the guard units, on the other hand, they are expected to perform very poor in terms of generalisation ability but are defined with the objective of limiting the regions of the pattern space associated with the training classes and can provide, therefore, a much better spurious pattern rejection capacity.

A similar method was independently developed by Burgess et al [8] where a hyper-box is defined to enclose the set of training patterns of a class, based on the orientation of the principal components of that class and estimation of the size of the bounded box through the set of training examples. Similarly to the guard unit method the algorithm is used in conjunction with a neural network (the "perceptron cascade") to obtain an improved overall classification performance.

The approach of guard units can be shown by Vasconcelos et al [78] to work very effectively in the particular case of the rejection of random patterns and it is also shown that the technique scales up well to practical cases, using for illustrative purposes its application in the classification of alphanumeric characters [79].

This chapter will describe the guard unit approach and present experimental results comparing the recognition and rejection rates attainable by both a standard MLP network and an enhanced architecture, operating on the same problem of the classification of alphanumeric characters described in the previous chapter. The mechanism by which the inclusion of guard units makes the network more efficient in rejecting "invalid" patterns is described and explained, and the practical implications of the advantages of using the enhanced architecture are established.

3.2 Single Guard Units

A set of special-purpose processing elements, the guard units, is defined which are responsible for checking if the input patterns presented to the network have a minimum similarity with the defined training classes. These guard units are independent of the conventional network architecture, but are functionally integrated with it. The resulting (integrated) network is illustrated in Figure 3.1. Each possible class of patterns is associated with a separate guard unit and during the network's training phase each guard unit sees only patterns belonging to the class that it represents. As with the units in the first hidden layer of the MLP at the heart of the processing network, each guard unit is fully interconnected with the input layer and its output provides an additional input, along with the classification decision output from the MLP network, to a final decision mechanism. There is no competition among the guard units during a classification task, instead each guard unit provides an overall “accept” or “reject” decision with respect to an input pattern, depending on its similarity with the connection weights feeding into the guard unit.

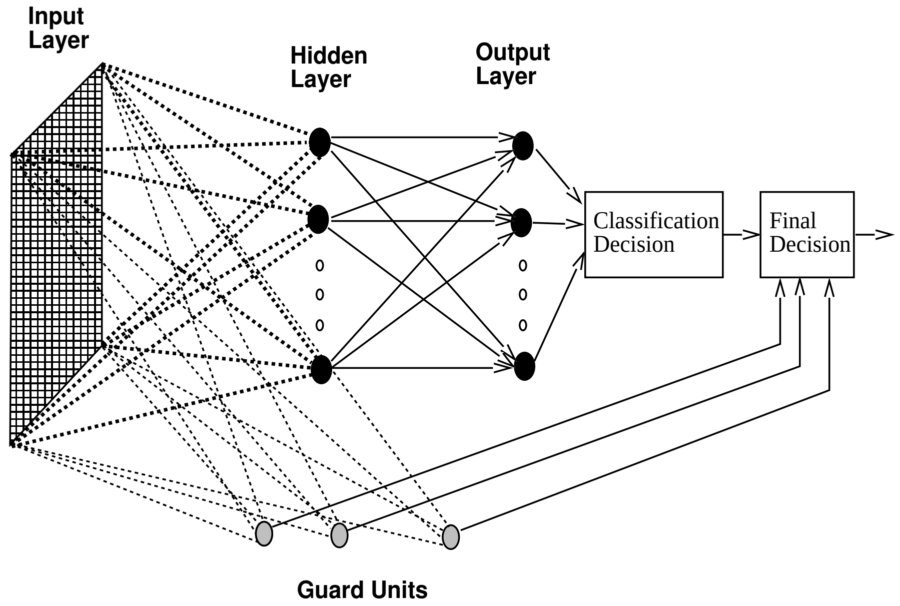


Figure 3.1: Multilayer perceptron network with guard units

The standard backpropagation learning algorithm is used to train that part of the network corresponding to the conventional MLP, while the guard units themselves are trained using a very simple “one-shot learning” procedure. This consists of defining the weight vector for each particular guard unit by taking the mean pixel values over all training patterns belonging to the class. This strategy is executed automatically during the network’s training phase and results in an amalgamation of the class training patterns. It requires only a single processing step over the entire training set for its definition, making the training of guard units a simple operation. Thus, if $\vec{w}_i$ denotes the weight vector of guard unit i and $\vec{x}_{pi}$ denotes a training pattern from class i, then $\vec{w}_i$ is given by :

$$\vec{w}_i = \frac{\sum_{p=1}^{n_i} \vec{x}_{pi}}{n_i}$$

where n_i is the number of training patterns in class i.

The crucial role of the guard units, however, is in the network's recall phase. In this phase they operate by providing an output proportional to the similarity between their weight vectors and the input pattern. This requires the definition of an appropriate criterion of similarity between the two vectors in order for an acceptance or rejection to take place. In the experiments described in this thesis two variations have been considered. The first is based on the calculation of the inner product between the vectors and, therefore, defines a *linear guard unit* (LGU), while the second is based on the use of a simple Euclidean distance metric, defining a *Euclidean guard unit* (EGU). Other similarity measures such as, for example, the Mahalanobis distance [70] could also have been applied. Because the effectiveness of the distance measure employed will depend on the statistical distribution of the training data, in some situations the Mahalanobis distance could generate better representations of that distribution. The disadvantage is, however, the fact that it is a bit more complex to calculate than the simple inner product or the Euclidean distance.

It should be noted that this method of computing a similarity measure can be related to other approaches that limit the region of the input space considered to belong to each class, such as nearest neighbour [13, 88] or Bayesian classification methods [19]. However, the method presented here has a much lower degree of computational complexity which, as will be shown, nevertheless works very effectively in practice.

The output of the guard units is taken into account by the system in computing a final decision about the identity of an input pattern. For each guard unit that did not accept the input pattern as having a minimum similarity with its weight vector, a '0' output is issued. Otherwise, the guard unit's output is '1'. The final decision device is responsible for checking the response of the guard unit associated with the class to which the MLP processor would, if operating alone, assign the input. In the case of a '0' output, then no matter what the output of the winning unit in the output layer, the final decision is to reject the pattern as not belonging to that class. However, if the guard unit's output is '1' then the normal classification process of the MLP architecture is accepted as the final decision defining the identity of the input pattern.

3.2.1 Linear guard units (LGUs)

In order to decide whether an input pattern has a minimum similarity with the training patterns, or not, it is necessary to define a point of reference. One s-traightforward approach is to use the inner product of the guard unit's weight vector by itself ($\vec{w}_i \cdot \vec{w}_i$), as this vector represents a point in the input space expected to be located near the middle of the region in which the training patterns of each class are located. Since it is not expected that input patterns will be rigorously similar to the amalgamated training patterns it is necessary to moderate the inner product $\vec{w}_i \cdot \vec{w}_i$ with the introduction of a relaxation parameter (ρ), which can be calculated as a certain percentage of the original inner product ($\vec{w}_i \cdot \vec{w}_i$). Obviously, the larger the value of ρ chosen the more flexible is the boundary defined by each guard unit. This represents the threshold for considering the input pattern as having a minimum similarity with the guard unit's representation. Equation 3.1 denotes the output function (o_{pi}) for input pattern $\vec{x}_p$ for LGUs.

$$o_{pi}(\vec{x}_p) \quad = \begin{cases} 1 & \text{, if } \vec{x}_p \cdot \vec{w}_i \ \geq \ (\vec{w}_i \cdot \vec{w}_i \ - \ \rho); \\ 0 & \text{, otherwise} \end{cases} \tag{3.1}$$

This represents the simplest form of guard unit specification.

In situations where the patterns present in the context of application are binary, as is the case of the experimental environment described in this thesis, they can be used directly as input to Equation 3.1. In the case of non binary patterns it is more appropriate that all training patterns are normalised to the interval $[0,1]$ (keeping the information about the amplitude of the vectors) before defining the guard unit representations. This is in order to avoid patterns with arbitrarily large components of being accepted as valid when given as input to Equation 3.1.

3.2.2 Euclidean guard units (EGUs)

In the case of the EGU, each guard unit defines a rejection area corresponding to a hypersphere in the input space. In a similar way to the case of a radial basis function [7] that computes how distant a pattern is from a centroid, the guard unit's output function (o_{pi}) calculates the simple Euclidean distance between the input pattern and the vector representing the amalgamated training patterns. This is given by equation 3.2, where n denotes the dimensionality of the input space and d is the minimum similarity considered for accepting the input pattern as valid.

$$o_{pi}(\vec{x}_p) = \begin{cases} 1 & \text{, if } \sqrt{\sum_{j=1}^{n}(x_j - w_{ij})^2} \geq d; \\ 0 & \text{, otherwise} \end{cases} \tag{3.2}$$

This offers an alternative mechanism for pattern rejection which is based on the commonly used and intuitively satisfying Euclidean distance metric.

3.2.3 The decision surfaces with the guard units

In order to explain and illustrate the function of the guard units more clearly, the classification problem involving two classes illustrated in Figure 3.2 can be considered. This shows the decision boundary created to separate the two classes determined by the backpropagation algorithm together with typical decision boundaries defined for the case of LGUs.

The introduction of guard units makes possible the definition of a rejection

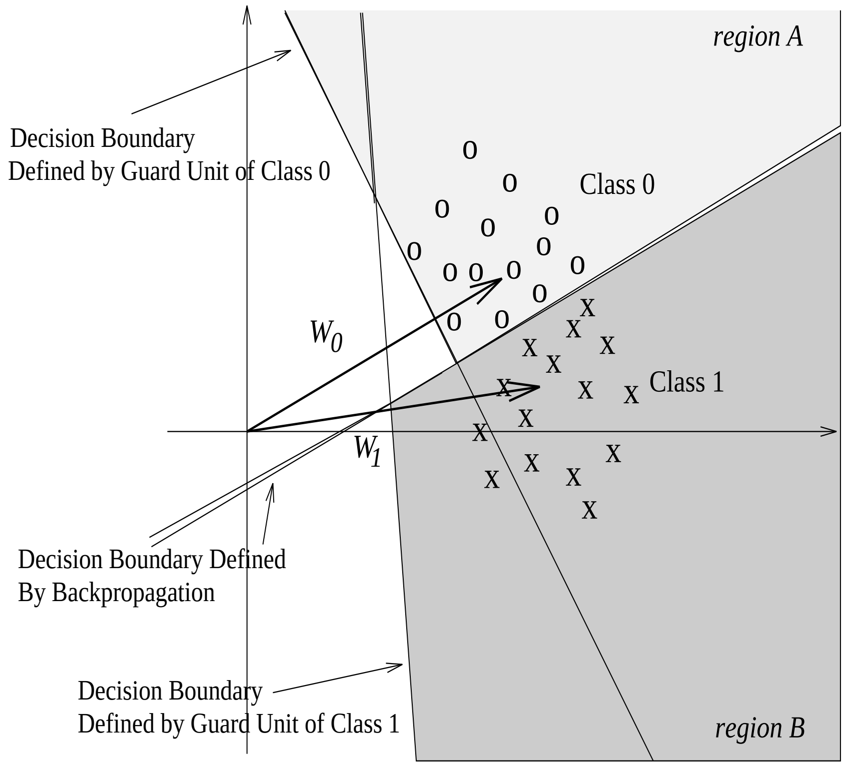


Figure 3.2: Input space separation with linear guard units

area in the input space in contrast to the case when the MLP network is used alone. The stored pattern in each guard unit corresponds in practice to a template pattern representing its class. Consequently, patterns which are very dissimilar from that template do not cause the guard unit to respond positively, while the criterion of required similarity can be controlled to avoid, or at least minimise, rejection of valid patterns. As can be seen in Figure 3.2, the discriminant function determined by each linear guard unit significantly reduces the area considered to belong to each training class. The entire region above the decision boundary defined by backpropagation is now confined to *region A*, when considering patterns belonging to class 0. Similarly, the region below the decision boundary defined by backpropagation is confined to *region B*, when considering patterns from class 1.

A similar situation occurs in the case of EGUs but, in this situation, the decision surface defined by each unit corresponds to a circular region (a hypersphere, in the n-dimensional case) as illustrated in Figure 3.3. The acceptance regions

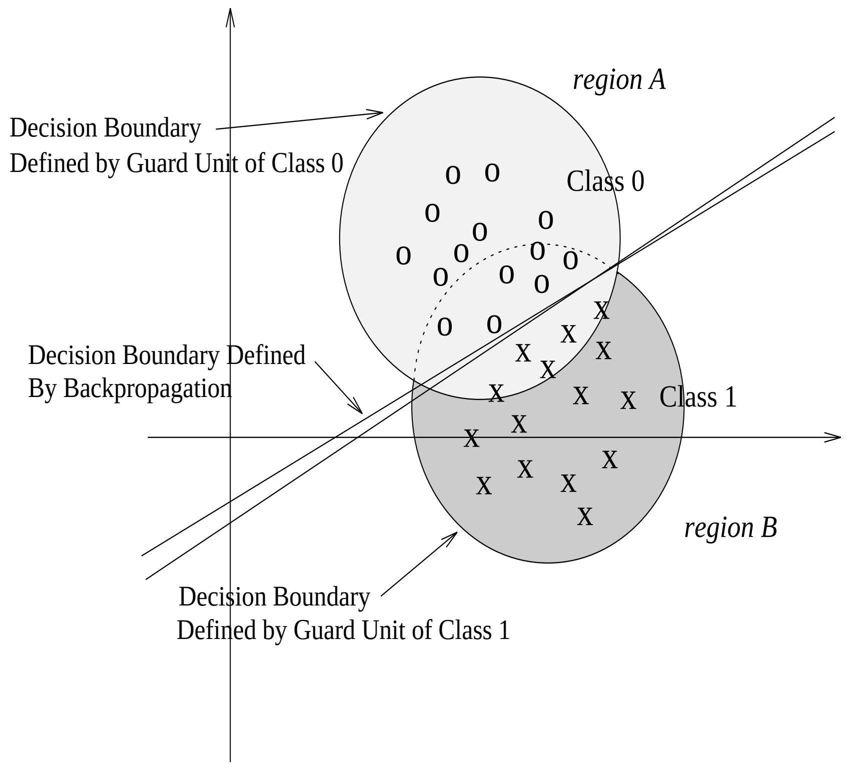


Figure 3.3: Input space separation with Euclidean guard units

considered as "valid" by the combined network are also significantly reduced in this case.

3.2.4 The inversion of MLP networks with guard units

The observations made in the previous section in the case of a simple 2-dimensional problem are important in order to understand the benefits gained with the approach presented. It is important to establish that these observations scale up to a higher dimensional class of problems, more appropriate to a practical application. This leads to the problem of how to visualise the results. One technique that can be used is to test the tolerance of the networks with respect to the visual appearance of the input patterns presented. Clearly, a network with limited tolerance to input pattern distortions will be good at rejecting spurious patterns. The method of network inversion for multilayer networks described in Chapter 2 provides a means of testing this tolerance.

In order to investigate this idea, the standard MLP and the LGU version of the guard unit network were trained on the problem of the recognition of characters, making use in one of the cases of a database composed of machine printed digits and in another case of a set of handwritten digits. The inversion of the networks were simulated several times after they had been trained with different values of the moderation parameter ρ and sets of patterns accepted as legitimate were obtained.

In all the simulations executed, there is a consistent observation that the modified network is much less tolerant to patterns with random characteristics. Figure 3.5 illustrates one of the trials for a network with guard units trained with machine printed digits, and the variable ρ set to 100. This value was chosen based on the maximum value of matching considered between the guard unit's vector and an input pattern, which is $(\vec{w}_i \cdot \vec{w}_i)$, equal to about 220 on average (value of ρ corresponding to 45 % of the original inner product). A substantial improvement in the digit shapes can be noted as compared to those obtained in the case of an MLP without guard units (Figure 3.4). An improvement is similarly noted for handwritten digits in this case with much less uniform characteristics. Figure 3.6 illustrates the inverted patterns for a LGU network with ρ equal to 60 (ρ equal to about 50 % of the inner product). Even if compared to the inverted patterns obtained for the network without guard units trained on the machine printed set (shown in Figure 3.4), whose patterns present much less variation than in the handwritten set, a clear improvement in the shapes of the digits is noted.

Another, perhaps more objective way of measuring the function of the guard units, is to observe the values of the inner product between the final inverted pattern for each class and the amalgamation of training patterns. Since an amalgamation of the patterns from a certain class is expected to represent relatively well the main characteristics of that class in the case where there is not much spread in the training data, this procedure can give a good idea about how far patterns being accepted as genuine by the networks are from the training classes. Therefore, the closer a pattern is to another pattern the greater should be the inner product between them.

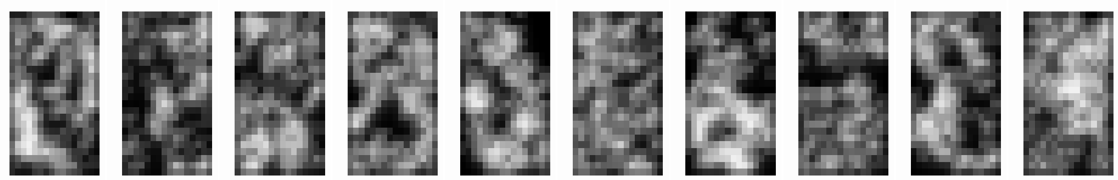


Figure 3.4: Inverted digits classified by an MLP trained with the machine printed set

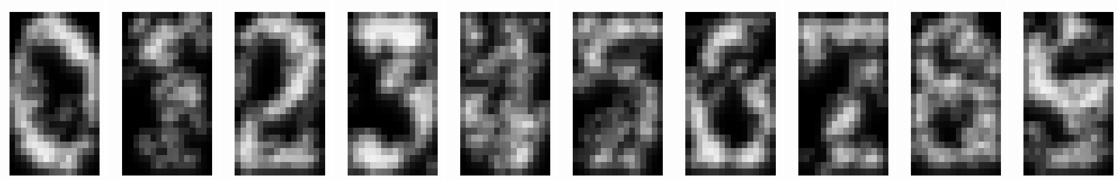


Figure 3.5: Inverted digits for the MLP with LGUs trained with the machine printed set

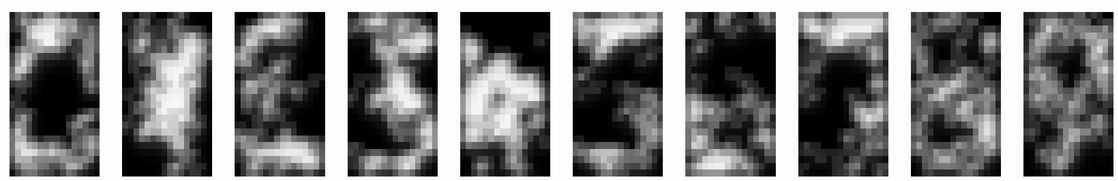


Figure 3.6: Inverted digits for the MLP with LGUs trained with the handwritten set

The comparative results for the standard MLP and the enhanced network with LGUs are shown in Table 3.1 for the case of machine printed digits and Table 3.2 for the case of handwritten digits, and they confirm that the inverted patterns accepted by the guard units are much closer to the training patterns than those accepted by the standard MLP. The *maximum* values presented in the tables correspond to the maximum value of the inner product between each guard unit's weight vector and itself. It can be seen that in certain cases (class 5 in

Table 3.1, for example) the approximation between the inverted pattern and the amalgamation of the patterns is very poor in the case of the standard MLP and this does not happen in the case of the LGU network (and also with the EGU network), since the criterion of a minimum similarity does not allow acceptance of random patterns.

Class	MLP Network	MLP with LGUs, $\rho = 100$	Maximum
0	7.48	186.60	249.57
1	56.10	84.77	170.23
2	80.07	170.19	245.28
3	68.85	171.61	250.05
4	24.99	98.16	184.09
5	1.65	127.22	213.67
6	33.72	162.92	223.78
7	91.00	174.31	253.26
8	23.25	133.49	231.23
9	79.61	112.91	198.74
Average =	46.67	142.22	221.99

Table 3.1: Inner products for networks trained with machine printed digits

3.2.5 Experimental results

In this section, a series of practical experiments in the application of the classification of handwritten characters is considered as an example to demonstrate the improvement in performance obtained with the introduction of the mechanism of guard units. First, a number of experiments are repeated with the standard MLP and, subsequently, results are shown with the networks using guard units.

The experiments deal with the same task described in Chapter 2 of training the network to recognise handwritten digits extracted from postcode information

Class	MLP Network	MLP with LGUs, $\rho = 60$	Maximum
0	61.03	107.87	159.99
1	0.82	82.22	97.11
2	34.64	88.27	146.73
3	6.62	112.53	157.37
4	18.78	88.69	137.88
5	16.62	103.73	134.51
6	59.97	101.17	146.55
7	45.05	108.65	181.53
8	29.75	62.04	104.53
9	36.57	89.90	139.85
Average =	30.98	94.50	124.86

Table 3.2: Inner products for networks trained with handwritten digits

on envelopes, and of testing its rejection performance with respect to alphabetic letters. As mentioned before, these particular data sets were adopted in order to carry out a strict test of rejection performance, since many letters and numerals share similar characteristics (for example 2/Z, 5/S, O/0, I/1, etc). The technique is general, however, and can be equally applied to any other data sets. The use of such "real" data is particularly important in evaluating the practical benefits offered by the approach described.

The simulations considered a single hidden layer network composed of 20 processing units. The entire database of digits contains, in this case, 2400 patterns separated into two disjoint subsets of 400 and 2000 patterns, respectively. The first subset was used to train the network with 40 patterns per class which was found, through experimentation, to be of an adequate size. By adequate size, it is meant here that the classification results observed are not far from other similar results reported in the literature on the same problem [38, 40, 47] and they are at a level sufficiently high to allow a fair comparison of the networks in terms of spurious pattern rejection capabilities. The second subset was employed as a test set for measuring correct digit recognition performance. The database of alphabetic letters, from the same source, is composed of 7800 patterns with 300 patterns per class.

In the experiments with the standard MLP, the best rate of digit recognition achieved for the database of 2000 digits was 87.5 % for a 0 % rejection rate of alphabetic characters. In other simulations, a level of confidence for acceptance of a classification response was introduced imposing the constraint that it should be accepted only when the response of the winning output unit exceeds those of the others by a certain margin. Otherwise, the input pattern is rejected. Figure 3.7 shows the results obtained using different values for the level of confidence of 0, 0.15, 0.25, 0.35. The best rate of letter rejection achieved was 36.8 % using the value of 0.35 for the level of confidence but, in this case, the digit recognition rate dropped to 77 %.

Experiments with networks integrated with either LGUs or EGUs showed an improvement over the rejection performance of the conventional architecture. Figures 3.8 and 3.9 illustrate the results observed for simulations carried out using different values of the relaxation parameter (ρ) for LGUs and different values of the d parameter for EGUs, respectively. These results were obtained with the level of confidence for the MLP part of the network set to 0. The previous results with the standard MLP are included in Figures 3.8 and 3.9, using the digit recognition rates as a point of reference (matching them over the rates obtained with the architectures with the guard units). Since the structure with linear guard units and the standard MLP generated approximately equivalent digit recognition rates (the enhanced architecture, in fact, performed marginally the better) a meaningful comparison of the respective letter rejection rates can be made, and it can be seen that a significant improvement can be obtained in terms of rejection capability.

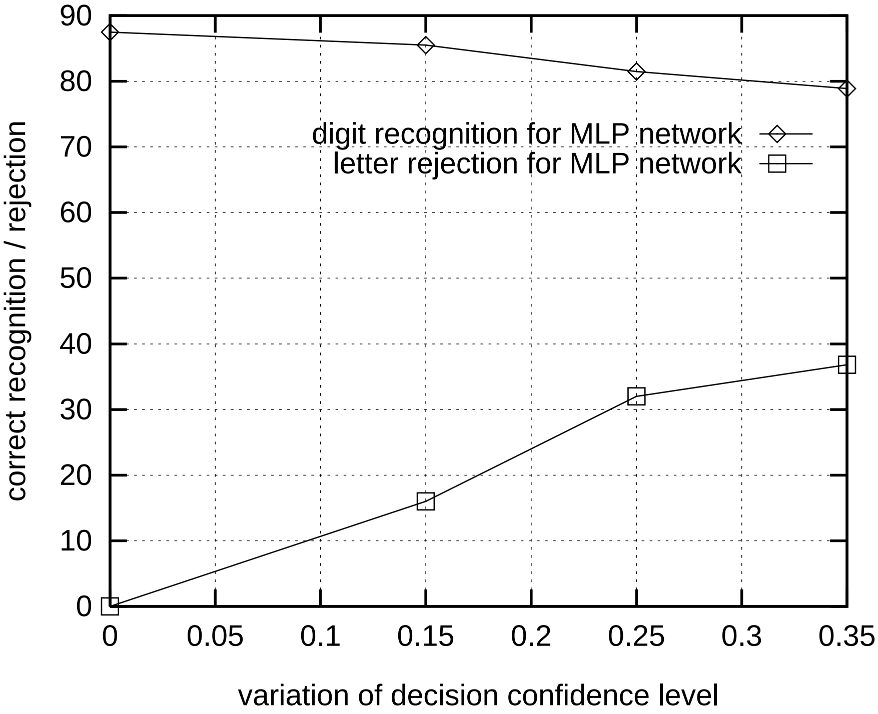


Figure 3.7: Digit recognition vs letter rejection for the standard MLP

The best digit recognition rate achieved was 88 % for a rejection rate of 22 % for non-digit characters, against 0 % for the rejection rate achieved previously. This suggests that an effective rejection mechanism can be implemented by a judicious change of the processing architecture which does not add significantly to its complexity. Modifying the ρ parameter to smaller values (implying more rigorous classification criteria) brings about an improvement in the rejection of non-digit characters, although this will also cause a degradation in the network's performance in the classification of the digits. The best rejection rate achieved was 55 % together with the worst recognition rate of 77 %. This represents an increase by a margin of 18 % in the letter rejection rate as compared to the standard MLP maintaining the same rate of 77 % for the recognition of digits.

As can be seen in Figure 3.9, the network using the Euclidean guard units also showed a stronger capacity to reject letters than the standard MLP architecture. The best rejection rate reached 62 %, although in this case the digit recognition

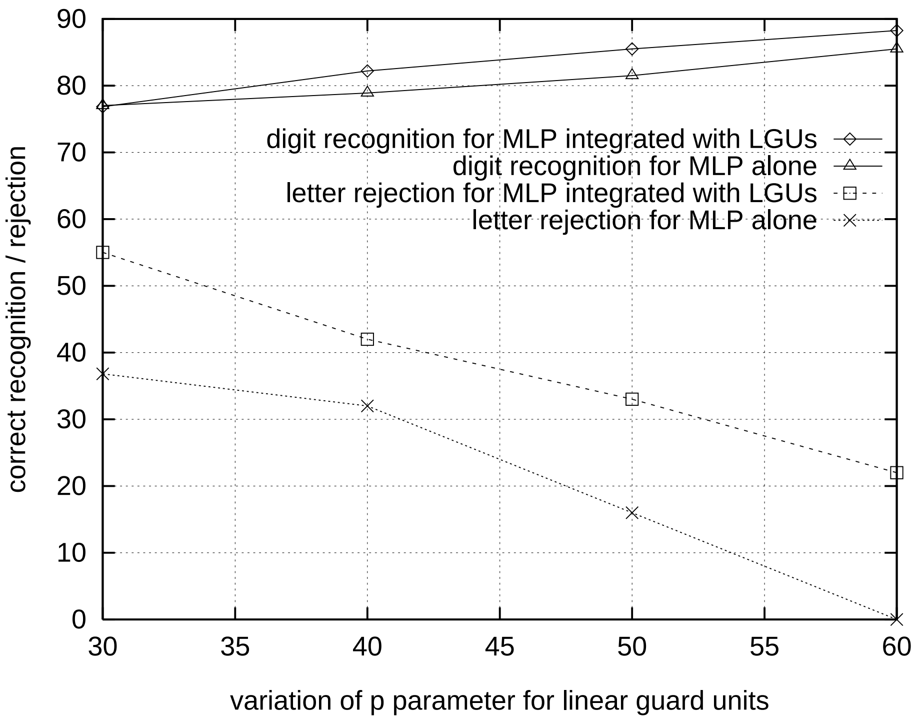


Figure 3.8: Digit recognition vs letter rejection for the MLP with LGUs

rate dropped to about 72 %. As the value of the d parameter is changed a trade off can be observed between the digit recognition rate and the letter rejection rate. This result, however, demonstrates a clearly improved performance over that attainable with the conventional MLP network alone.

3.3 Multiple Guard Units (MGUs)

Although the definition of LGUs and EGUs have shown overall improvements in terms of network rejection capability, some problems with the initial method can be raised with respect to the fact that the definition of single guard units do not take in consideration situations where the training data is not confined to a convex region of the pattern space and is, contrarily, very spread throughout the space. With the objective of considering this problem, an extension of the original concept is introduced in this section.

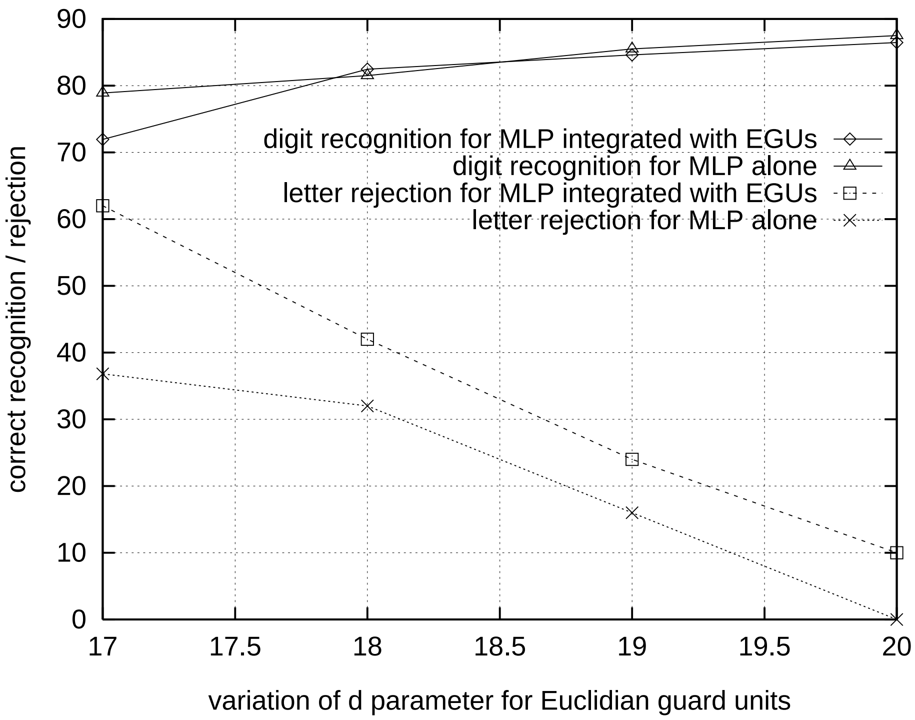


Figure 3.9: Digit recognition vs letter rejection for the MLP with EGUs

The modified procedure allows the number of guard units for each training class to grow dynamically during the training phase through the use of a clustering algorithm. This network is referred to as the multiple guard unit network (MGU).

3.3.1 Description of the method

The broad strategy adopted is basically the same as that used by the guard unit network to reject invalid patterns but, in this case, the procedure is used during the training phase in order to create new guard units when the distance between a new training pattern extracted from the database and the stored template in the corresponding guard unit exceeds a given threshold value D. In the experiments reported here, the Euclidean distance was used, once again, as the distance metric between the patterns but other metrics could also have been used.

The process begins with the adoption of a single guard unit for each class and then, by performing a single step over the entire training set, new guard units are added whenever the criterion of similarity exceeds the threshold D. The value of the D parameter is essentially defined ad-hoc but taking into consideration the important fact that it controls the number of clusters formed and consequently the number of guard units defined for each training class. The behaviour of the network in the recall phase is similar, with the introduction of a further threshold variable d. This is appropriate because although this variable has the same role as the variable D it has optimum values which might be different for each training class, depending on the similarity distribution of the patterns encountered in a particular application.

The use of multiple guard units (MGUs) per class allows a more refined representation of the areas in input space occupied by the training classes. In this case, combinations of hyperspheres define more complex decision surfaces than those obtained with LGUs and EGUs and, therefore, provide a better approximation of the training data distribution. Figure 3.10 illustrates the result of applying this modified version of the network in the definition of the valid acceptance areas for the two-class problem considered previously. As discussed at the beginning of this chapter, the only regions important in the diagram as far as the guard units are concerned is the decision surface which surrounds, as a whole, all the a priori defined training classes. This makes the job of guard units rather easier and more effective than if they had to create optimal decision boundaries separating the training classes. It is a common observation that the use of systems based on distance metrics for classification purposes is not usually as efficient as classifiers based, for example, on the use of hyperplanes as the discriminant function. For this reason, the part of the system that measures distance between patterns, the guard unit network, is used to encapsulate the training data and not to define classification regions to separate each training class from the others.

The definition of MGUs, however, introduces the question of how to estimate

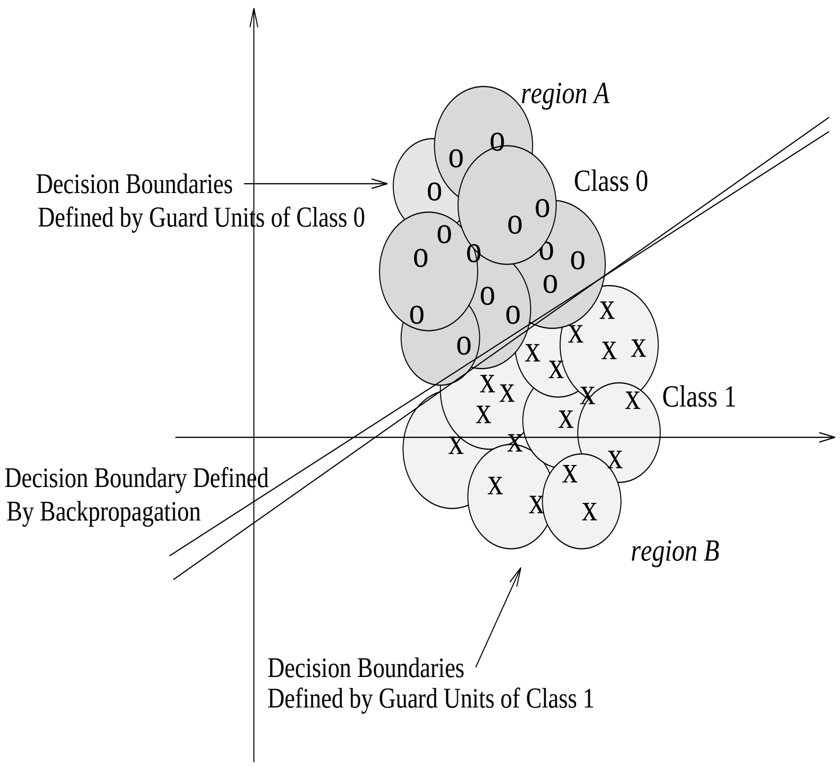


Figure 3.10: Input space separation with multiple guard units

efficiently the values of the threshold variable d for the many guard units in the different classes. In a real application, it would be appropriate to seek the development of an automatic system so that the manipulation of the values of this variable in order to obtain a good solution to the problem could be minimised. A simple way of solving this problem is to make use of a validation set for the estimation of the parameters. The idea consists of submitting the network to a second presentation of patterns belonging to the training classes and of modifying the parameters according to the their maximum (or mean) distances to the guard unit representations previously created. The original training set itself may be used as a validation set, as in the case of the experiments reported here.

Algorithm for creating MGUs

1. For each training class i do;

2. Take a training pattern $\vec{x}_p$ from class i;

3. If there is no guard unit for class i create one, assign the current input to its weights and go back to 2; otherwise go to 4;

4. Amalgamate the input pattern (the mean over the input and the guard unit's weights) with the closest guard unit representation if the distance between them is less than variable D, go back to 2; otherwise go to 5;

5. If none of the guard unit representations has an Euclidean distance to the input less than variable D create another guard unit assigning the input pattern to its weights;

6. Go back to 2;

Algorithm for estimating the thresholds

1. For each training class i do steps 2 to 4;

2. Take a validation pattern $\vec{x}_p$ from class i;

3. Carry out a competition between the guard units of class i and calculate the distance between pattern $\vec{x}_p$ and the representation of the winning unit;

4. Consider the calculated distance as the new estimated value of variable d for the particular winning guard unit if that distance is greater than the current value of d; otherwise do not change the value of d; go back to 2;

5. Finally, relax the values of each estimated variable d by a very small percentage rp ($d = d + rp/d \times 100$) in order to create more flexible thresholds.

3.3.2 Experimental results

Adopting this principle, further recognition experiments were carried out with the defined data set of handwritten characters, and the results demonstrated a further improvement over the previous architecture when LGUs and EGUs were used. These will now be described in more detail.

Experiment 1

Again, the experiments deal with the classification of handwritten characters. A set of 7800 letters is used to test the performance of the networks for rejecting spurious patterns. In this set of experiments with MGUs, a slightly bigger training digit set composed of 1000 patterns (100 per class) is used to train the network and the same set of 2000 digits employed before is used to test network recognition performance. The MLP part of the network contains the same number of 20 hidden units employed initially in the single guard unit structures. The experiments are repeated several times for different values of the confidence level imposed at the outputs of the MLP network for acceptance of a classification decision. The MGU network is also tested with different values of the relaxation parameter (rp) used to moderate the estimated values of variable d.

In a first attempt, no relaxation ($rp = 0\%$) was used and this case presented the best results in terms of letter rejection performance (for example, in one of the cases, a 41 % rejection rate was obtained as opposed to 0 % observed with the standard MLP). However, as might be expected, the degradation in the recognition of digits was also substantial (4.8 % on average). The full results achieved in this particular simulation are described in Table 3.3.

With the introduction of a small relaxation into the thresholds d, a very good trade-off can be brought about. For example, Table 3.4 summarizes the results

Relaxation Parameter (rp) = 0 %						
	Digit Recognition			Letter Rejection		
confidence	MLP	MGU	$\Delta =$	MLP	MGU	$\Delta =$
0	91.5	85.0	-6.5	0	41	+41
0.15	88.6	85.0	-3.6	21	49	+28
0.25	86.2	81.6	-4.6	31	56	+25
0.35	85.4	80.0	-5.4	40	57	+17
0.45	80.5	75.0	-5.5	48	66	+18
0.55	76.7	70.9	-5.8	58	71	+13
0.65	74.3	70.0	-4.3	65	75	+10

Table 3.3: Digit recognition rate vs letter rejection rate for the MGU, $rp = 0$

obtained for both the standard MLP and the architecture with MGUs with 2 % of relaxation. This particular case represented the best compromise in terms of performance, if "best" is seen as the situation where a consistent improvement in the rejection rate is obtained with only a very small degradation in the recognition of valid patterns. As shown in Table 3.4, for a small averaged degradation in the digit recognition performance (1.6 %) the increase in letter rejection is very substantial. Other values for rp were also tested and presented different compromises between spurious pattern rejection and valid pattern acceptance. Probably one of the best strategies for the definition of the relaxation value is to start with 0 % relaxation and observe any degradation with respect to the valid patterns. Then, the value of rp can be increased until any possible degradation has reached an acceptable level. It is important to note, however, that the term "acceptable level" used here is entirely dependent on the particular task environment and it can only be accurately determined by considering the objective context of the application.

Another important aspect, as indicated in Table 3.5 for the case of relaxation

Relaxation Parameter (rp) = 2 %						
	Digit Recognition			Letter Rejection		
confidence	MLP	MGU	$\Delta =$	MLP	MGU	$\Delta =$
0	91.5	89.4	-2.1	0	28	+28
0.15	88.6	86.7	-1.9	21	41	+20
0.25	86.2	84.7	-1.5	31	44	+13
0.35	85.4	83.4	-2.0	40	52	+12
0.45	80.5	79.2	-1.3	48	57	+9
0.55	76.7	75.8	-0.9	58	69	+11
0.65	74.3	72.9	-1.4	65	72	+7

Table 3.4: Digit recognition rate vs letter rejection rate for the MGU, $rp = 2$

parameter rp equal to 2 %, is the reduction of the digit error rates with the MGU network as a result of the more rigorous test of similarity between patterns. It can be concluded that patterns for which the network has low confidence about their identity are now being rejected instead of taking the risk of performing a misclassification. Indeed, it is important to pay attention to the fact that in all the experiments carried out with the different values of the parameter rp, the loss in recognition rate observed has not been transcribed into an increase in the error performance of the network but into an increase in its rejection performance. In other words, the overall error rates of the network has not gone up with the application of the MGU strategy.

The training of guard units created about 20 units per class on average (200 units in the total). Although this number of units can be considered high when compared to the original number of 20 units in the conventional MLP network, it can be argued, in fact, that if a full distance based classifier had been devised for the purpose of both classification of valid patterns and rejection of spurious

Relaxation Parameter (rp) = 2 %				
	Digit Error Rate		Digit Rejection Rate	
confidence	MLP	MGU	MLP	MGU
0	8.5	6.1	0	4.5
0.15	7.1	5.1	4.3	8.2
0.25	5.7	3.8	8.1	11.5
0.35	4.8	3.5	9.8	13.1
0.45	3.5	2.4	16.0	18.4
0.55	2.5	1.7	20.8	22.5
0.65	2.3	1.6	23.4	25.5

Table 3.5: Digit error rate vs digit rejection rate for the MGU, $rp = 2$

inputs, the number of required processing units would be much higher[1].

Experiment 2

An additional group of experiments carried out with the MGU network was developed with the purpose of demonstrating that the performance of this improved network should be even more striking when the input patterns are very different from the training classes. The experiments consist of the test of acceptance/rejection of 5000 randomly generated binary patterns presented to the network. Table 3.6 shows the complete results obtained where it can be seen that, even for very random patterns, the performance of the pure MLP can only slightly approximate the results achieved with the MGU architecture with a very high confidence level imposed at the network's output (0.65, 88 % spurious pattern rejection rate). For a far superior rejection performance (99 %), the integrated

[1]In chapter 5, for example, it will be seen that in order to maintain a similar level of valid pattern recognition performance a network such an RBF needs to have a considerably larger number (1000) of processing units.

network provides a much better valid pattern recognition rate (89.4 %) as opposed to 74.3 % obtained with the standard MLP. Again these results correspond to the value of 2 % for the parameter rp.

Relaxation Parameter (rp) = 2 %						
	Digit Recognition			Random Pattern Rejection		
confidence	MLP	MGU	$\Delta =$	MLP	MGU	$\Delta =$
0	91.5	89.4	-2.1	0	99	+99
0.15	88.6	86.7	-1.9	32	99	+67
0.25	86.2	84.7	-1.5	45	99	+54
0.35	85.4	83.4	-2.0	65	100	+35
0.45	80.5	79.2	-1.3	71	100	+29
0.55	76.7	75.8	-0.9	73	100	+27
0.65	74.3	72.9	-1.4	88	100	+12

Table 3.6: Random pattern rejection rates for the MGU

3.4 A Comparison with Other Models

The concept of guard units has a correspondence with that of the vigilance parameter in ART models [11] although the process of classification itself in these networks presents a completely different characteristic from that of the integrated MGU structure.

In the recall phase of operation of the ART network, when a pattern is fed into the input vector it is passed through the network from the input layer and matched against the representations stored in each node of the output layer. The winning node then sends its stored class pattern backwards to a comparison layer where its similarity with respect to the input pattern is tested. A reset circuit is responsible for the test by checking whether or not this degree of similarity is within the limit of a vigilance threshold. The test is a ratio count of the number of matched ones in both the input vector and comparison vector (in the ART-1 version) and subsequent verification of the result against the threshold.

It is in this respect that the guard unit approach is similar to the ART paradigm. As with the reset circuit in ART networks, each guard unit is responsible for performing a match operation between the input pattern and the composite representation of the training patterns. The fundamental difference, however, is in the fact that in ART networks the decision taken about the membership of an input pattern among the possible training classes is also a result of a template-match operation where the input is compared directly against the representations stored in the network. In the MGU architecture, the process of classifying an input pattern as belonging to one of the valid classes is executed by the MLP part of the network, which takes its decision based on the identification of certain features in the input pattern rather than on the distance between the input and the representations defined in the network. Only the guard units perform operations based on pattern matching and their role is to classify the input pattern as a "valid" pattern or not, without regard to which training class it might belong.

The operation of MGUs can also be related to other classifiers that use the Euclidean distance metric as a measure of similarity between patterns such as radial basis function networks (RBF). MGUs are different, however, again in the sense that they are not involved in the process of deciding to which training class an input pattern might belong. That is the function of the MLP part of the network. Consequently, it can be said that the job of the guard units is made easier because the objective in this case is that of creating boundaries surrounding all the defined training classes as if they corresponded to a single category. It can be argued that fewer processing units of the Euclidean distance based type should be expected to be necessary in the MGU network in order to maintain at an appropriate level the generalisation of genuine patterns than in a network such as an RBF, whose processing units have also to satisfy the additional imposed constraint that each training class has to be separated from the others.

It is also worth mentioning that, as a single clustering technique, the algorithm for the definition of MGUs presents some similarities with other known methods for cluster definition such as Learning Vector Quantisation (LVQ) and other general clustering procedures [70, 5].

3.5 Conclusion

This chapter has shown how a simple independent set of units working in parallel with a conventional MLP architecture enhances the reliability of the network with respect to the rejection of arbitrary patterns which do not share real class membership with the legitimate training classes. Moreover, this desirable property can be realized without a significant increase in the complexity of the network, and even the configuration which generates multiple guard units – the architecture which gave the best performance – uses only a simple clustering algorithm in order to build the required class representations which control rejection.

The experiments reported have shown that the standard MLP trained with backpropagation performs very badly when patterns different from the training classes are presented as input but, on the other hand, the network generalises very well over the training patterns. The guard units present an opposite behaviour with very bad generalisation performance and good ability to separate the training classes from the other regions of the pattern space. The objective of the integrated architecture is essentially to combine the good characteristics of MLPs with a more rigorous definition of the decision surfaces created by the guard units.

It has been shown that rejection of invalid patterns can be significantly improved without much degradation of performance with respect to the recognition of valid patterns. Considering the level of performance achieved with the experimental character data reported, where there can be a natural inter-class similarity between certain letter/numeral pairs (e.g 0/O, 8/B, 5/S, 2/Z, etc), it is to be expected that in many practical situations where the patterns to be rejected may share significantly less similarity with the training classes, the enhancement in performance can be even more striking.

Other advantages of this approach are that its applicability is independent of the classes present in a domain of application and that the auxiliary network embodying the guard units can also be used in conjunction with other kinds of networks which also lack the ability effectively to reject unknown patterns. The enhanced architectures described here would therefore appear to offer significant advantages over the basic MLP architecture since they improve the spurious pattern rejection performance of the model at low cost.

Chapter 4

Alternative MLP Network Configurations

4.1 Introduction

In the previous chapter, the problem of the detection of spurious patterns was considered from the perspective of the development of an additional mechanism integrated with the MLP network.

The objective of this chapter, in contrast, is to explore different alternative configurations for the standard MLP in order to transform the network itself into a structure more inherently able to detect invalid patterns. In this situation, the only factor taken in consideration for deciding about the identity of an input pattern is a confidence level imposed at the network's output for acceptance or rejection of a classification decision. The transformations applied to the standard network are developed with the main objective of establishing limits to the areas of the pattern space associated with each training class. However, since it is also of primary concern that the generalisation capacity of the network should not be significantly impaired, the boundaries defined surrounding the training classes should not be excessively rigorous. The algorithms described first construct the separation lines between the classes and then, as a consequence, boundaries are also created to separate the training points from the other parts of the pattern space. The idea is not to approach the problem from the perspective of a direct computation of the *density* of the training data, for example through the estimation of its probability density function, as in the case of some methods based on Bayesian theory [6, 64]. Instead, the aim is to modify the characteristics of the decision boundaries generated by the MLP network, through changes in the network structure and unit function, so that the decision regions defined not only separate the training data but also create bounded regions encapsulating individual classes.

The first part of this chapter describes the idea of mapping the original input space of an MLP into an extended space where it is possible to create closed classification surfaces. This network is referred to as the paraboloidal MLP. A second approach based on a modification to the original propagation rule of the standard MLP (the inner product) through its normalisation by the input vector is considered and simulations are used to outline the implications of this method with respect to the network operation. Then, two additional modified MLP configurations are investigated from the same point of view : the first consists of an MLP which uses additional direct connections (short-cut connections) from the input to the output layer of the network, and the second is an MLP which uses a Gaussian as its activation function instead of the usual sigmoid and which defines "semi-localised" receptive fields for the processing units in the network.

It is shown how the models investigated construct their decision regions in the pattern space, and the implications of these constructions for pattern rejection are discussed. The technique of network inversion is used to evaluate the various approaches, and practical experiments on the classification of handwritten characters are reported.

4.2 The Paraboloidal MLP Network (MLP^{par})

4.2.1 Extending the input pattern space

The first method considered in this chapter for modifying the MLP network comes from an idea in computational geometry theory, and is based on the observation that if a linear threshold unit that is operating initially in a space of n-dimensions (with inputs denoted by $[x_1, x_2, \ldots, x_n]$) now receives its input from an extended input space of one higher dimension where the $(n+1)$ coordinate corresponds to the sum of the squares of the other original n coordinates ($[x_1, x_2, \ldots, x_n, x_1^2 + x_2^2 + \ldots + x_n^2]$), then its receptive field becomes a sphere [55]. In this situation, the initial infinite half-space which would be normally covered by the threshold unit is mapped onto a limited region of the original space in the form of a localised receptive field (hypersphere). The usual backpropagation algorithm can be used to find the set of weights represented by $w_1, w_2, \ldots, w_n, w_{n+1}$ which solves the required input/output mappings, where w_{n+1} corresponds to the weight associated with the extra coordinate.

The geometrical representation of the transformation that takes place is that the original input space is mapped into a hyper-paraboloid of revolution in a space of higher dimension and the hyperplanes defined in this extended space during the process of network training intersect the paraboloid creating ellipsoids that form localised receptive fields, when projected onto the original input space. Figure 4.1 illustrates the construction of the paraboloid for the 2-dimensional case, where it is possible to see the result of the input space transformation. In this figure, x and y correspond to the coordinates in the original space and the addition of the coordinate $z = x^2 + y^2$ creates the paraboloid. The shaded area in the diagram represents the localised receptive field generated with the intersection of a hyperplane with the paraboloid.

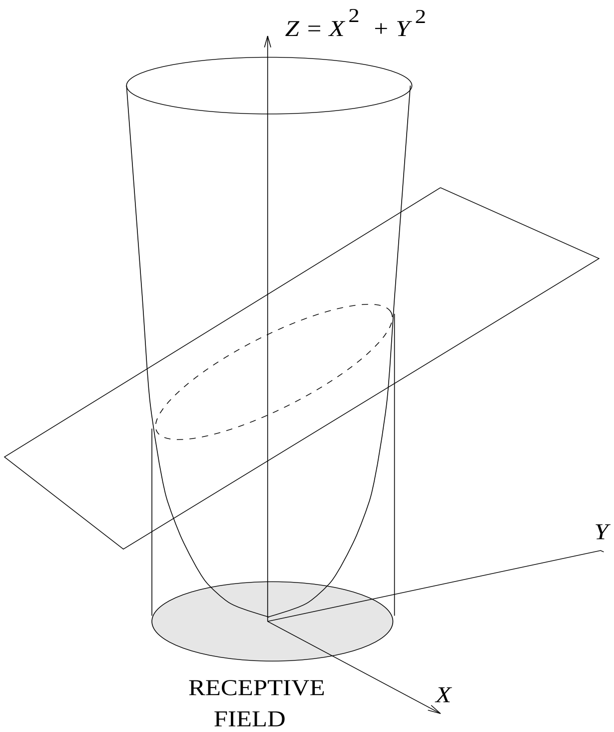


Figure 4.1: Construction of the paraboloid for the 2-dimensional case. Redrawn from [Omohundro(1990)]

The basis of this technique is the assumption that patterns belonging to the same class are expected to have similar values for the $(n+1)$ coordinate and that the more distant the patterns become from the training patterns the more different will be the values of the $(n+1)$ coordinate (as compared to the values obtained for the training patterns). Hence, the presentation of this additional information to the network enables it to define hyperplanes in the extended space which tend to separate the training points from the other parts of the input space through the creation of closed boundaries.

The concept of defining localised receptive fields for the MLP network makes it approximate another model of feedforward neural networks — the class of radial basis function classifiers [7, 52]. In fact, work reported by various researchers [48, 55] has demonstrated how one type of network can be mapped onto the other (for example, how the MLP can be mapped onto the RBF structure through the idea of the paraboloid) and Dorffner [18] has even shown how a unified framework can be devised to integrate both types of architecture. The idea presented in [18], described as conic section function networks (CSFN), is to define receptive fields for the processing units so that they can fold to a more closed receptive field or unfold to a completely open one depending on the particular characteristics of the data distribution. In this thesis, however, since the primary concern is to avoid spurious patterns being classified as valid by the network, only closed decision regions are of interest.

In this section, it is shown how the implementation of the idea of the paraboloid in the standard MLP modifies the decision regions generated by the network and, therefore, directly contributes to the enhancement of its rejection capabilities. This concept of extending the input space is not only a potential solution to the rejection problem but it has also been seen by some authors as an advantageous method for improving the convergence of the training process of some neural networks based on constructive algorithms [9, 22, 51]. From here on, this modification in the MLP network will be denoted as the paraboloidal MLP (MLP^{par}).

4.2.2 Visualisation of the decision regions

In order to understand the influence of the input space transformation in the definition of the decision regions in a classification process, the visualisation experiment in the 2-dimensional classification problem defined in Chapter 2 is repeated with the MLP^{par} structure, in a situation involving 9 classes. The result is shown in Figure 4.2.

In contrast with the result obtained with the standard MLP in the same problem, presented in Chapter 2 and illustrated again in Figure 4.3, the decision regions created in this case tend to correspond to closed regions with a much more confined area of the input space associated with each class. The hyperplanes are placed in this situation not only to separate the training classes but also to create

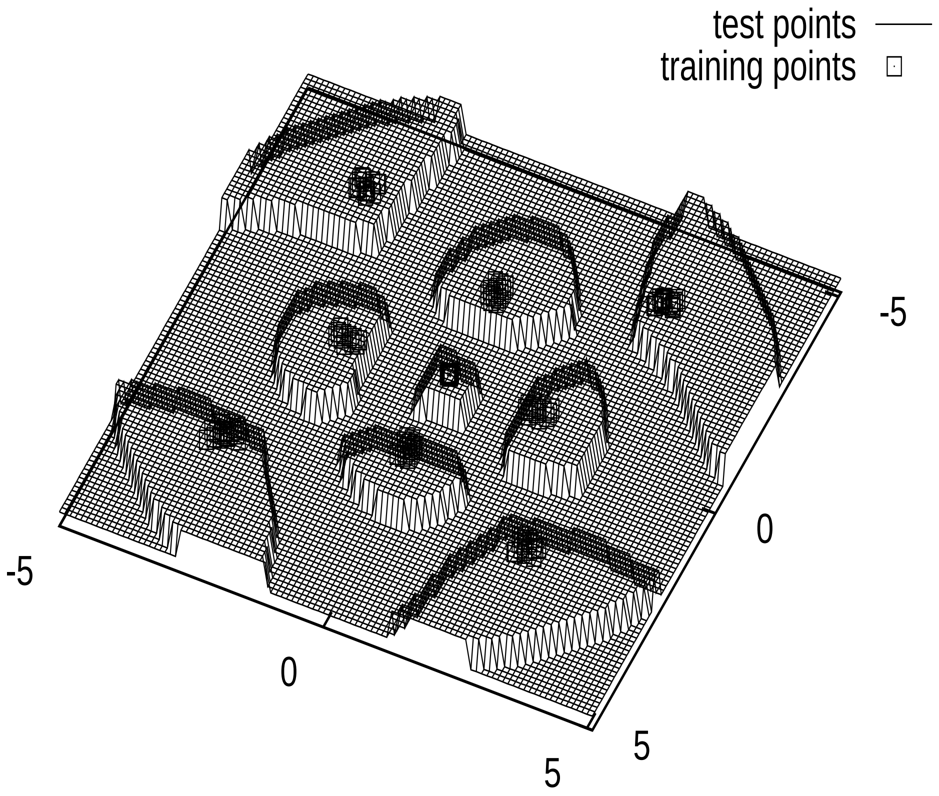


Figure 4.2: Decision regions for the paraboloidal MLP (9 classes)

flexible boundaries surrounding the training points. This situation is more reliable for rejecting spurious patterns than that obtained with the standard MLP since the rejection areas in the pattern space for the case of the standard MLP only correspond to the regions in between the training classes, and patterns far away from the training points are still very likely to be classified as valid patterns.

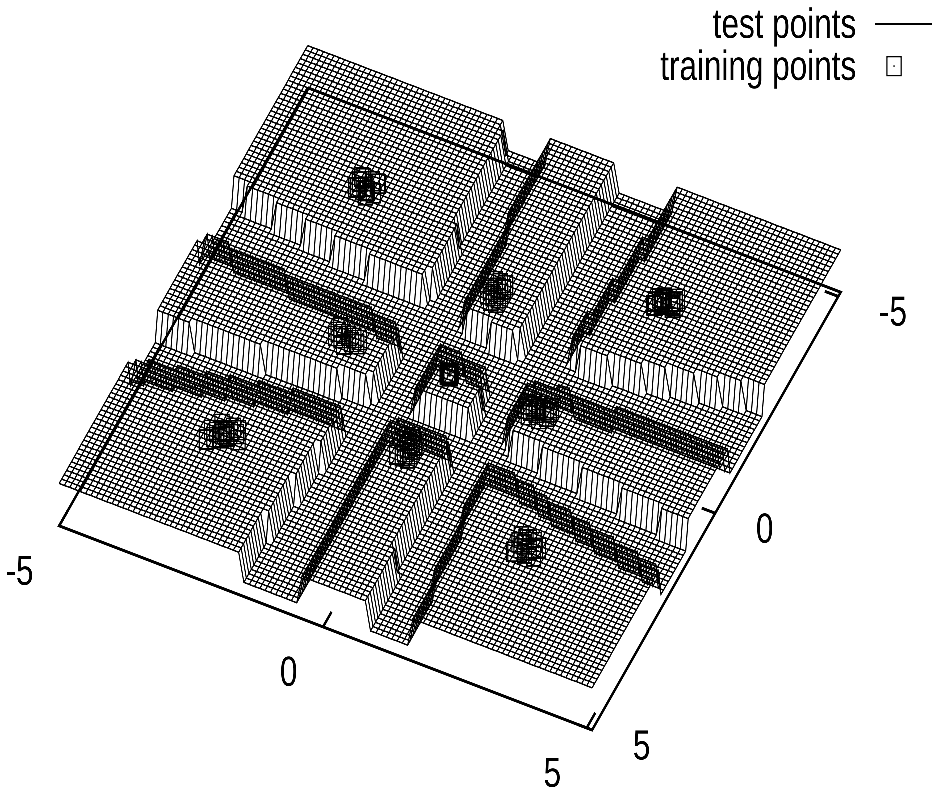


Figure 4.3: Decision regions for the standard MLP (9 classes)

4.2.3 A further evaluation of the method

An examination of the propagation rule defined for the processing units in the modified structure can provide some useful information. The first thing to note is that the propagation rule corresponding to the inner product (net_j, for each unit j) given by :

$$net_j = \sum_{i=1}^{n+1} x_i \cdot w_{ij} \tag{4.1}$$

can be re-written in a different way [18] to provide :

$$net_j = \sum_{i=1}^{n} x_i \cdot w_{ij} + w_{n+1} \sum_{i=1}^{n} x_i^2.$$

It is important to observe in this equation that the role of the weight w_{n+1} is different from that of the others in the sense that it directly determines the influence that the paraboloid has on the definition of each unit's receptive field.

Originally, the weight w_{n+1} associated with the $n+1$ input is defined in the same way as all the other coordinates during the normal process of training and, therefore, it is expected that a different value for w_{n+1} should result for each different processing unit. If the value of w_{n+1} defined is small then the term $\sum_{i=1}^{n} x_i^2$ will have a small effect on the shape of the decision region created by the processing unit. Conversely, if w_{n+1} is large then the paraboloid part will have a strong influence on the receptive field, creating decision regions of more limited size. The determination of which hidden units will have small receptive fields and which will have large receptive fields is not known a priori in this situation and is a result of how these units have their weights adapted by the learning algorithm to represent the training data. For example, some units that are adapted to represent certain features may only require small receptive fields, which are sufficient for their differentiation from other features. While in some other units that evolve to represent different features it may be the case that only larger receptive fields are sufficient to fit the training data in a way that it separates the different classes.

Another way of approaching the situation, however, is to consider the definition of weight w_{n+1} as a constant prior to the training phase ($B = w_{n+1}$). This constant can be manipulated accordingly so as to obtain a receptive field of a different size each time but common to all processing units [18]. For example, if $B = 0$ the $n+1$ input will have no effect in the definition of the decisions regions and the units will see the input space exactly as if they were operating in the original n-dimensional space, instead of in the extended space. In other words, the discriminant function of the units will correspond to hyperplanes and unlimited decision regions will be created by the network. If $B > 0$, on the other hand, then the hypersphere part is taken into consideration, creating bounded decision regions with variable size according to its value.

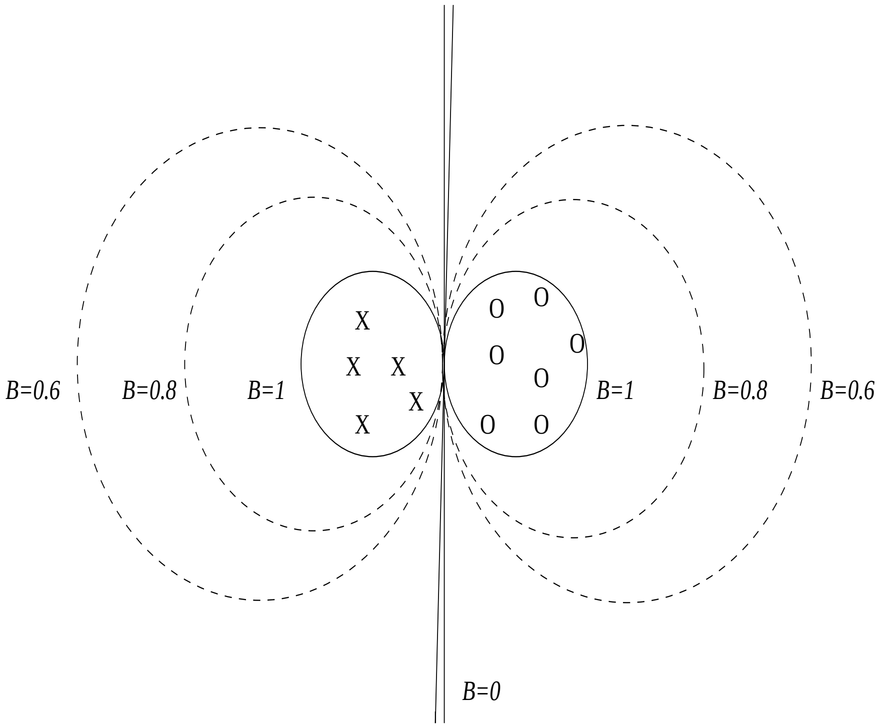
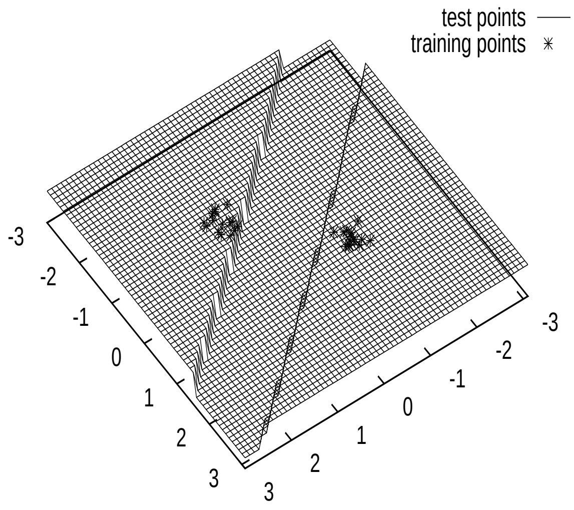


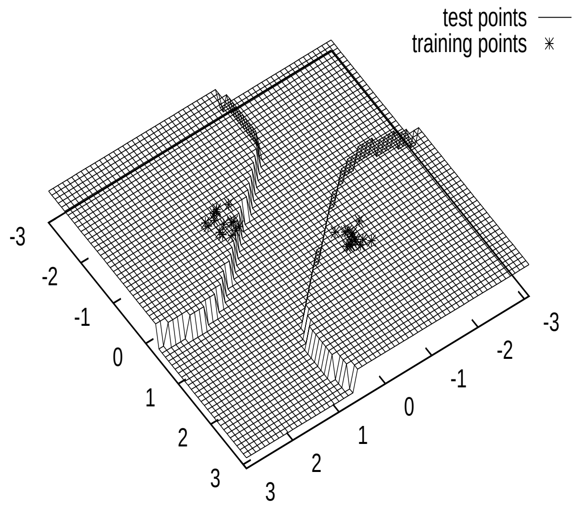
Figure 4.4: Decision regions for different values of the B variable. Redrawn from Dorffner (1994)

Figure 4.4 illustrates the effect of modifications in the value of B. In this example, two classes with a small number of patterns from each class are represented in the diagram and two hypothetical hidden units each responding to patterns from a different class are shown to encapsulate the training data. As explained, the straight line corresponds to the case where $B = 0$.

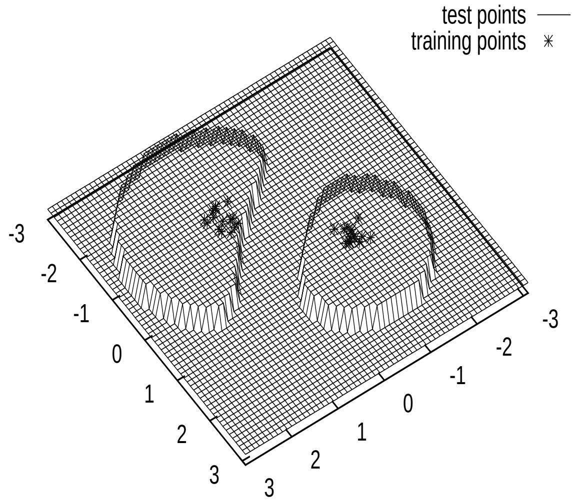
The actual simulation of a single-hidden layer MLP^{par} network and the visualisation of the decision regions generated in the 2-class problem shown in Figure 4.5 further illustrate the variable influence that the hypersphere part can have in the shapes of the classification regions. Figure 4.5 shows the results obtained with different values for B, illustrating a gradual evolution of the decision regions created from completely open regions (when $B = 0$) to bounded regions (when $B = 1$).



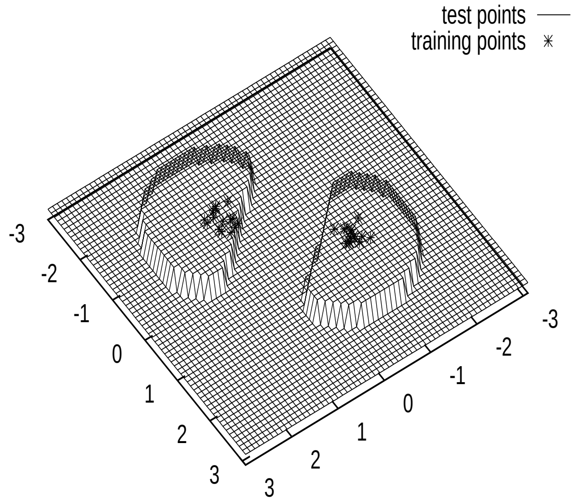
(a)



(b)



(c)



(d)

Figure 4.5: 2-class problem : MLP^{par} with different values for the B parameter (a) $B = 0$, (b) $B = 0.4$, (c) $B = 0.8$, (d) $B = 1.0$.

One important and interesting aspect to mention about the MLP^{par} but which also relates to some other networks described in this chapter is that with the usual process of network training with backpropagation, where the initialisation of the weights is performed randomly, there is an imminent risk, albeit unlikely to happen in practice, of the occurrence of a certain configuration for the shapes of the decision regions which is completely undesirable with respect to the spurious pattern problem.

This situation refers to the case where for a given class, among all the possible classes present in the problem, all the hidden units that respond to patterns from that class are defined in the training process as "negative" detectors of the other classes in the problem. This situation is easier to understand if the example shown in Figure 4.4 is considered. For the separation of the two classes in this problem it is clearly seen that only one processing unit is necessary and sufficient since this unit can easily generate a decision boundary between the two classes. Consider, however, that more than one hidden unit is present in the network. Since during the original training process the only constraint to be satisfied is the separation of the classes, with no additional imposition that the two classes should have boundary lines surrounding each one of them, it is possible that all the existing processing units are adapted to become "positive" detectors of the presence of patterns from only one of the classes. All the patterns are then classified as belonging to the first class if they lie in a limited region of the pattern space corresponding to the receptive fields of the units but any other input will also be considered as genuine as those that effectively belong to the second class. Another interpretation, one which is more appropriate to describe the more general case of m classes, is that the processing units have become in fact "negative" detectors of the presence of patterns from the first class because they have their weights defined so that any arbitrary input not belonging to this class is assigned as a member of the second class.

However, fortunately, there are easy ways of circumventing this kind of problem and one of them is to make sure that there is a distribution of processing units as positive detectors of the different classes. Since the introduction of the paraboloid makes MLPs similar to RBF networks, one natural way of doing this is to initialise the hidden units with weights in the vicinity of the regions where patterns from the different classes are concentrated, causing them to respond positively to these regions. Then, during the training process, there is always a way of preventing these units from becoming negative detectors of the other classes. Another similar strategy is to initialise the network through the process known as a *Voronoi tessellation* of the input data which is an idea inspired on the design of MLP networks using principles from nearest neighbour systems [76]. The idea is to place the hyperplanes directly between the data points from the different classes as an initialisation process, where the data points are represented by approximate cluster centers (for example, the mean over the patterns). In the particular case described here, the utilisation of this technique would be slightly modified so that when a hyperplane is placed between two training classes, another version of the same hyperplane pointing in the opposite direction is also defined in order to make sure that the classes are not only being separated but are also being encapsulated by confined decision regions.

The effectiveness of the MLP^{par} approach should be expected, however, to vary in different situations according to the particular problem. This is because the introduction of only one extra coordinate in the extended space which carries information about all the other coordinates should have its effect diminished in proportion to the number of dimensions involved in the particular task. Considering the context of practical applications, where there can be a natural tendency for variations in the input data, it is logical to think that the application of this method in spaces of lower dimensions is likely to be more successful since the information in the extra coordinate is much more representative of the training classes. In spaces of higher dimensions, the occurrence of variations in the input data within each individual class is much more likely to affect the extra coordinate and this may lead to similar values for the extra coordinate being obtained for different classes. In consequence, the separation of the training classes based on the extra input becomes much less effective.

In problems involving higher dimensions, a possible modification in the approach is to consider the extension of the original input space in m-dimensions instead of only one, and of associating the calculation of each one of the m coordinate values with subdivisions of the input space instead of the whole space. This would allow the division of the responsibility for carrying information about the original coordinates between the different extra coordinates, where each of them is responsible for covering only a specific area of the input space.

One possible drawback of the MLP^{par} method is in problems involving binary patterns and this will become evident in the next section with the experimental results reported.

4.2.4 Experimental results

The MLP^{par} network was tested in the application of the classification of handwritten characters following the same scheme of the digit recognition and letter rejection experiments described in the previous chapters. A network with 80 hidden units was simulated to compare with the equivalent standard MLP configuration of Chapter 2, and the complete set of results is presented. The experiments are repeated with different values for the classification confidence level imposed at the network's output. A pattern is classified as belonging to one of the classes if the output unit representing that class exceeds those of the others by the given confidence level. Otherwise, the pattern is rejected.

Table 4.1 outlines the digit recognition rates obtained with different values defined for the constant B. Table 4.2 describes the letter rejection rates also associated with each B value. The digit recognition results show a gradual decrease in the rates as B increases towards 1, although this degradation is not accentuated up to $B = 0.5$. This result is not surprising since that the use of units with a more local receptive field is expected to bring about some loss in terms of generalisation ability.

	Digit Recognition Rates (%)						
confidence	B=0	B=0.1	B=0.2	B=0.3	B=0.4	B=0.5	B=1.0
0	92.0	90.8	89.3	89.9	90.4	90.3	84.8
0.15	89.4	88.9	88.3	88.9	87.1	88.3	85.0
0.25	88.2	87.5	87.5	86.1	86.0	86.5	82.6
0.35	84.4	87.1	85.2	84.6	83.6	86.5	79.6
0.45	80.8	84.9	85.6	81.5	84.0	80.6	79.3
0.55	81.3	84.2	81.5	82.7	80.9	80.5	80.2
0.65	72.9	81.6	80.5	79.6	79.8	79.5	75.0

Table 4.1: Digit recognition rates for the MLP^{par}

The more interesting and to some extent intriguing result, however, is with respect to the letter rejection rates obtained. Table 4.2 shows that for small values of B (0.1 and 0.2) there was actually a decrease in the rejection performance. Then, for $B = 0.3$, the rejection rate was improved when compared to the standard MLP ($B = 0$) for most cases and for $B > 0.3$ it started again to decrease. This is rather surprising because, in principle, it would be expected that larger values of B would generate more strict and closed decision regions. Therefore, there should be an increase in the rate of spurious patterns being rejected by the network. The results also suggest that there is actually an optimum value (0.3) for setting the B variable in this specific situation examined. Indeed, when compared to the standard MLP, the MLP with the paraboloid presented in this particular case offered an improved, although not very significantly, rejection performance (see Table 4.2).

confidence	Letter Rejection Rates (%)						
	B=0	B=0.1	B=0.2	B=0.3	B=0.4	B=0.5	B=1.0
0	0	0	0	0	0	0	0
0.15	17	14	14	25	23	19	19
0.25	25	22	31	37	31	26	24
0.35	32	30	41	42	37	32	30
0.45	46	32	45	48	47	46	35
0.55	48	44	55	53	52	54	37
0.65	64	53	57	58	57	57	45

Table 4.2: Letter rejection rates for the MLP^{par}

The disappointing performance of the MLP^{par} in the experiments can be explained by two factors. The first concerns the already mentioned problem of the number of dimensions involved in the task. There are 384 inputs in the original space in addition to the extra 385^{th} coordinate. This can be considered a relatively high dimension and it can raise the question of the relevance of the extra coordinate. The second factor, and one which is more relevant in this case, refers to the fact that the task in hand deals with the classification of binary patterns. A simple examination of the equation for the $n+1$ coordinate, given by :

$$x_{n+1} = x_1^2 + x_2^2 + \dots x_n^2. \tag{4.2}$$

shows that in the case of binary patterns much information is reduced.

Consider initially the case where the pixels that compose the input matrix are represented by 1s and 0s, since extending the explanations given below to other representations is, as will be seen, a straightforward matter. Given that the only values in the input matrix are either 1s or 0s and the square of a unity value remains unchanged, equation 4.2 can be simply re-written with the removal of the squared factor :

$$x_{n+1} = x_1 + x_2 + \ \dots \ x_n.$$

Now, one possible interpretation for this equation is that the only information represented in it about an input pattern is the *area* (the number of pixels with value 1) occupied by the pattern in the input matrix. The equation is in effect taking a crude count of the number of 'on' pixels present in the input matrix. It is therefore clear that this information is not sufficiently relevant for discriminating between different classes in a practical application, especially when the task involves high dimensional data and the particular experiments concern the discrimination between patterns which share many characteristics such as letters and digits. To back up this argument, the simple example illustrated in Figure 4.6 can be considered. This Figure shows two patterns representing the digits 5 and 2, respectively, which are obviously different but cover exactly the same area in the input matrix (each composed of 90 coordinates). In spaces of higher dimensions the possibility of finding patterns from different classes with similar area values is even more likely to happen.

Taking into consideration the case where other values are employed to represent the binary patterns, such as arbitrary constants b and c, the situation is not very different. Equation 4.2 can be simply re-written as :

$$x_{n+1} = b_1^2 + b_2^2 + \ \dots \ + b_m^2 + \ \dots \ + c_1^2 + c_2^2 + \ \dots \ c_n^2.$$

where m is the number of pixels with value b in the image and n is the number of pixels with value c. A simple transformation in this equation produces :

$$x_{n+1} = ((b^2 \times m) + (c^2 \times n)) \times 1.$$

Interpreting variable m as the area occupied by the pattern in the input matrix and variable n as the rest of the input matrix (the area *not occupied* by the pattern), and considering that parameters b and c are constants defined a priori and are the same for all input patterns irrespective of their class, it is seen that the extra coordinate simply calculates the sum of multiples of theses areas and, therefore, does not provide much more relevant information to distinguish between different patterns than the calculation of the simple area would produce.

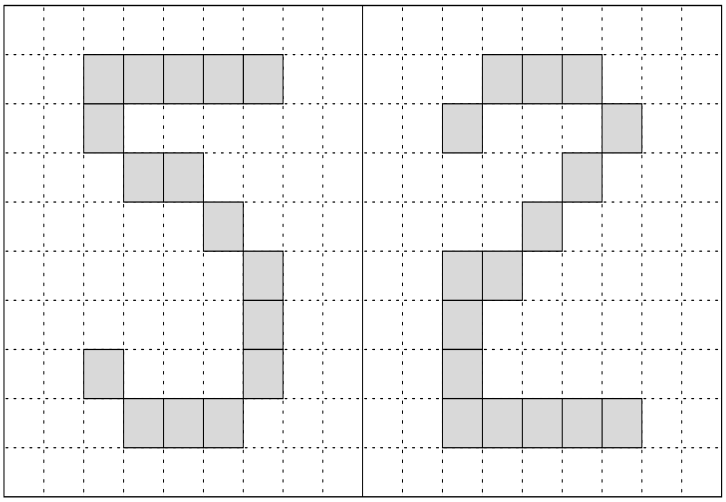


Figure 4.6: A simple example of patterns that cover the same area in the input matrix.

An examination of the character recognition experiments with respect to the variation of the $n + 1$ coordinate for the different digit classes and its comparison with the values observed for the test classes confirms the comments made above. Table 4.3 presents the mean, the minimum and the maximum value obtained for the $n + 1$ input calculated for the training set of 100 patterns per digit class. This table also shows the same values calculated for the test set of 300 letters per class. For the reader's convenience, only the results corresponding to the letter classes 'A' to 'J' are shown in Table 4.3 since these are sufficient to illustrate the point. This shows a complete overlap between the range of values found for both the digit classes and the letter classes implying that the information contained in the extra coordinate can have an opposite effect to that aimed at in the first place, contributing to the training and test classes being considered more similar to each other instead of for the differentiation between them. As can be seen, it is virtually impossible to find a digit class which does not overlap with any of the alphabetic classes.

	Digit Classes				Letter Classes		
Class	Min	Mean	Max	Class	Min	Mean	Max
0	17.0	43.7	88.6	A	14.6	40.9	84.3
1	14.4	83.9	154.9	B	13.3	42.3	87.2
2	9.2	34.6	88.7	C	5.6	32.0	64.0
3	9.1	34.6	93.2	D	13.8	39.7	104.1
4	7.7	29.8	87.4	E	10.3	37.7	89.9
5	12.3	33.1	87.9	F	7.9	33.9	77.2
6	5.5	38.1	95.1	G	12.2	34.5	94.5
7	8.6	28.5	58.1	H	7.7	38.7	79.0
8	9.2	40.9	93.9	I	14.7	65.3	170.4
9	8.4	35.8	68.9	J	6.5	28.7	67.1

Table 4.3: Overlap between training and test classes with respect to the $(n+1)$ coordinate

The final comment about this method, however, is that its ineffective application in the case of the experiments described does not invalidate completely its practical use. Many practical applications with different characteristics may still be found where the approach can be applied. The simple concept of creating a paraboloid in an extended space based on information from the original space is an appealing idea for the creation of bounded decision regions in the MLP architecture.

Extensions of the original idea can be easily thought of, such as the use of more relevant information represented in the extra coordinate. It may be possible, for example, to consider the extraction of some features from the original input image such as moments (or of other possible variables) and it is important that these features are used as input to a paraboloidal function to determine an output value to serve as the extra input. In this case, of course, other aspects about the computational complexity of the method involved has also to be taken into consideration.

In the context of the experiments carried out, of the classification of binary patterns, the MLP^{par} fails for the reasons already explained. However, the results obtained with the visualisation experiment in the 2-dimensional case showing the creation of bounded decision regions gives evidence to expect that in environments of real valued inputs, and especially in problems of lower dimensions, the method may still represent a potential enhancement in the network's rejection performance.

4.3 The Normalised MLP Network (MLP^{nor})

In this section, a modification in the original propagation rule employed in the standard MLP, the *inner product*, is considered with the objective of modifying the way that similarity is seen by each component unit in the network. This consists of taking the normalisation of the original inner product ($net_j = \sum_i x_i \cdot w_{ij}$) for each unit by its input vector before presenting it as input to the network's sigmoid activation function ($(1 + exp(-net_j))^{-1}$). The idea is to consider only the angle between the input vector and the weight vector as the most important aspect in deciding about their degree of similarity. With the normalisation of the propagation rule, instead of the region of the input space which a processing unit responds to being determined by a simple hyperplane, two separation lines form a receptive field defined by an angle with the unit's weight vector.

Equation 4.3 defines the new propagation rule net_j for each unit u_j in this normalised MLP network (MLP^{nor}), with ϕ being the angle between the input vector ($\vec{x}_p$) and the weight vector ($\vec{w}_j$) and θ the usual bias value. The normalisation of the propagation rule by the input vectors transforms the original pattern space into a hyper-sphere of unity size and any pattern is mapped into this sphere.

$$net_j \quad = \quad \frac{\mid \vec{w}_j \mid \; \cdot \; \mid \vec{x}_p \mid}{\mid\mid \vec{x}_p \mid\mid} \, cos \; \phi \; - \; \theta \tag{4.3}$$

The idea with this modification in the propagation rule is that, in theory, it is possible to obtain an input space separation for the training classes where only a limited region of the space is attributed to each class. Consider, for example, the hypothetical classification problem involving two classes illustrated in Figure 4.7. The weight vectors $\vec{w}_0$ and $\vec{w}_1$ representing the output units of classes 0 and 1, respectively, are defined such that they point in the direction where the training patterns of each class are located, and they form decision regions (regions A and B) according to an angle between each vector and two left and right boundary lines, as shown in the picture. In this case, despite the fact that, because the magnitude of the inputs are not considered, many patterns with different magnitudes (but that point at the same direction in the pattern space) can be mapped to similar points on the surface of the unity sphere, the regions representing the training classes (shaded areas in the diagram) are much more confined to the neighbourhood of the training points than for the case of the standard MLP network. In contrast, the part of the diagram corresponding to rejection regions (in white) is consequently much increased.

Theoretically speaking, the introduction of this modification in the MLP network requires the re-definition of the original equations for training the network

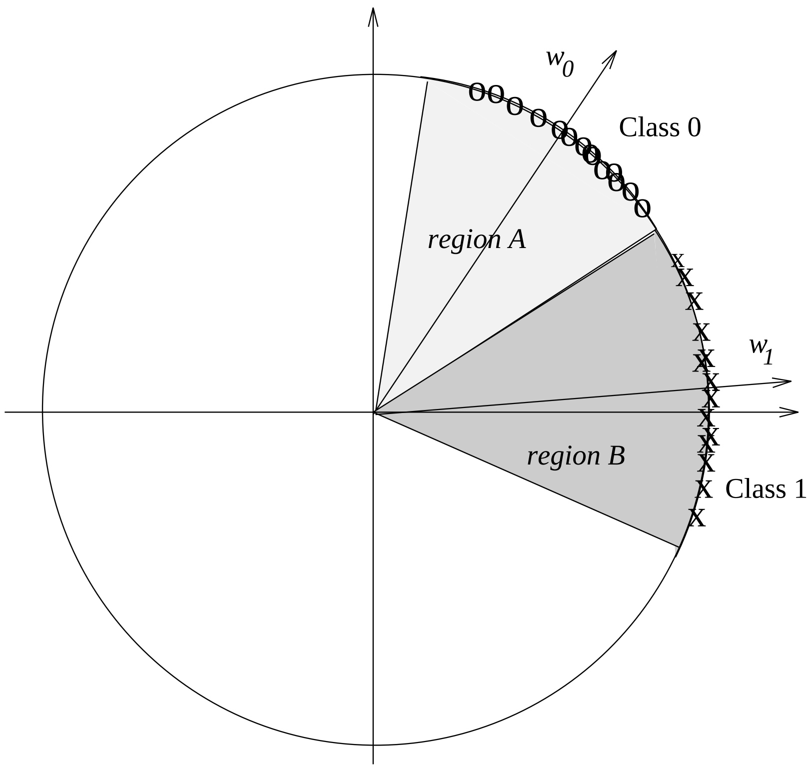


Figure 4.7: Ideal input space separation for the normalised MLP

with the backpropagation algorithm. This is necessary in order to guarantee that the process of network training still converges to a global (or local) minimum solution in the error space in a finite period of time. The complete re-definition of the backpropagation equations are described in Appendix A.

4.3.1 Unsatisfactory practical performance

In contrast with the original expectation, however, it was observed that the application of the backpropagation algorithm to the MLP^{nor} network in the same handwritten character classification problem described in the beginning of the chapter does not provide a much improved solution over the standard network configuration. The repetition of the experiments basically did not show any substantial difference in the classification rates obtained. One interesting aspect observed in the experiments was that the training process converged normally using the original generalised delta rule and since this form of the learning rule saves computation time it was preferred to the re-defined rule described in Appendix A.

Table 4.4 shows the digit recognition and letter rejection results for the MLP^{nor} and standard MLP networks. While the recognition rate of digits does not really change, the rejection of letters is only marginally improved in some of the cases and, therefore, cannot be considered as an overall improvement in performance.

	Digit Recognition Rate		Letter Rejection Rate	
confidence	MLP	MLP^{nor}	MLP	MLP^{nor}
0	92.0	91.9	0	0
0.15	89.0	90.0	16	21
0.25	87.9	88.3	32	32
0.35	86.4	84.7	37	43
0.45	83.0	82.3	51	51
0.55	80.9	79.8	57	58
0.65	75.1	77.1	68	66

Table 4.4: Digit recognition rate vs letter rejection rate for the MLP^{nor}

In order to investigate the reasons for the unsatisfactory performance of the MLP^{nor} network in practice, a hyperplane animator program [57] was used to monitor the evolution of the decision regions generated by the units in the network in the 2-class illustrative problem described above. The original animator program displays the gradual movement of the weight vectors of the processing units in the network throughout the pattern space. A modification was then appropriately made to show the different type of decision boundary defined in the MLP^{nor} network (corresponding to the two separation lines that form an angle with the weight vectors).

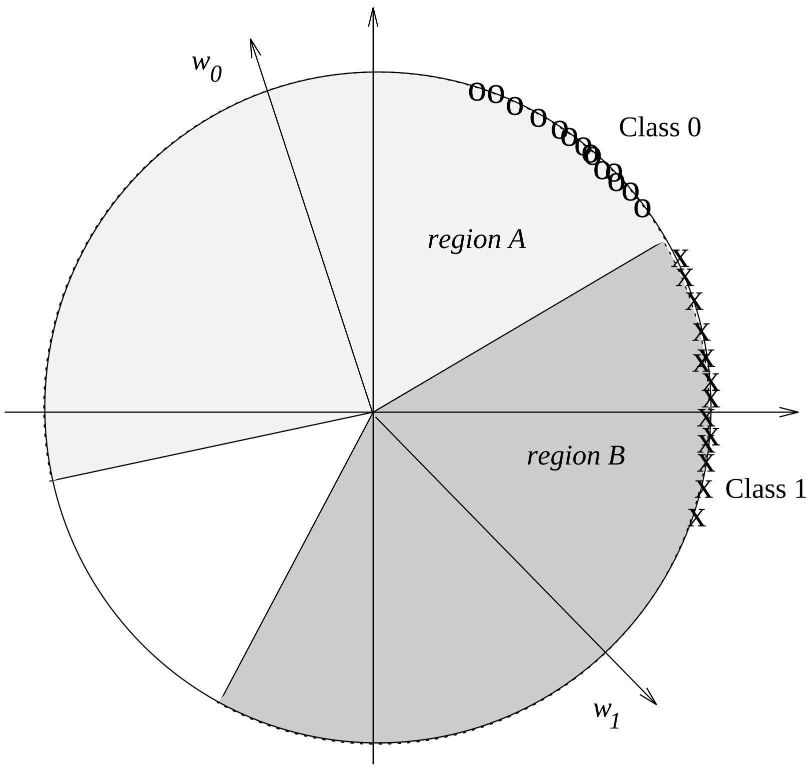


Figure 4.8: Actual input space separation for the normalised MLP

The simulation of the experiment explains that, most often, the decision regions generated by the network lead to solutions where very open areas of the space are attributed to each class, such as the one presented in Figure 4.8. Although theoretically, a unit's receptive field can be generated so as to be confined to a limited area of the space because there is a flexible movement of the separation lines that form the receptive field, it can be seen that this may not be necessarily the case after the training process is completed. In Figure 4.8, the regions representing the training classes are not as confined to the neighbourhood of the training points as might be expected, and even points very distant from the training patterns are considered as belonging to the trained classes.

The reason for this is, unfortunately, the same as that which explains the definition of open decision regions in the pattern space with the standard MLP network : the fact that because no information is available to the learning algorithm about points in other regions of the input space and the only constraint imposed in this context is the separation of the training data, the angle between the weight vectors and the separation lines can grow arbitrarily large[1], which defines a region that not only encapsulates the training patterns but also other regions in the input space.

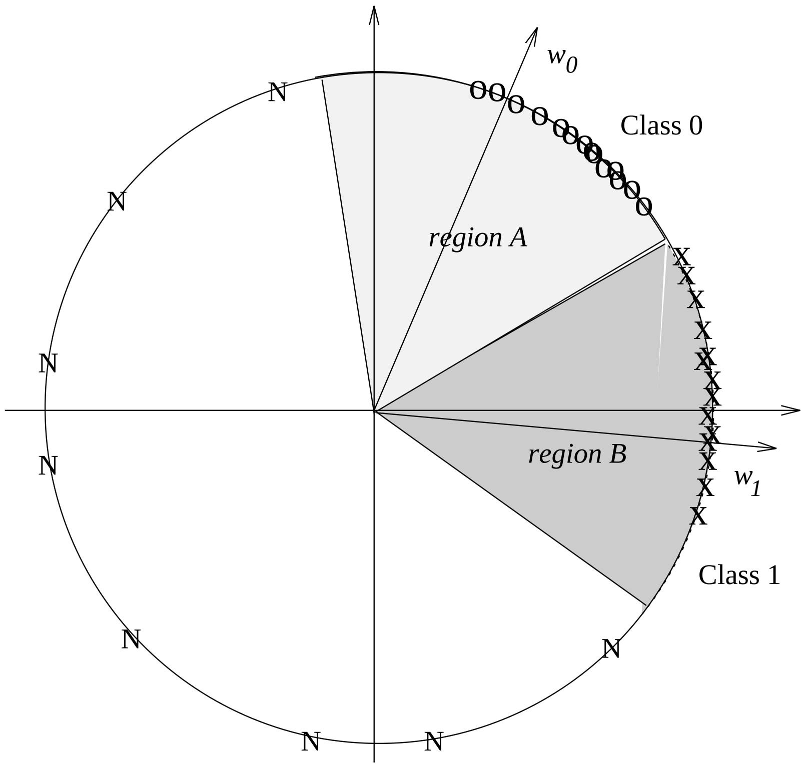


Figure 4.9: Input space separation with additional training data

This explanation can be further supported with a similar experiment to that described in Chapter 2 which makes use of extra training data points surrounding the original training patterns. These negative examples are designated by 'N' in Figure 4.9. In this case, the extra information imposes an additional constraint to be satisfied which requires the separation of the patterns 'N' from the other patterns in the training classes. As a result, the solution achieved by the learning algorithm approaches much more the "ideal" situation shown in Figure 4.7. It is important to note that, within the self imposed constraints of the work carried out in this thesis, this example can only serve as an illustration since information

[1]A variation of the original idea might be possible where the angles are prevented from growing arbitrarily. However, this further idea has not being investigated in this thesis.

about the patterns to be rejected is not available.

Although the MLP^{nor} has not presented the enhancement in performance desired, this network can be used, as will be seen in the following section, in conjunction with other different configurations.

4.4 MLPs with Direct Connections (DMLP)

Another possible idea for achieving an improvement in the MLP rejection performance is through structural modifications by the addition, to the existing interlayer structure of connections, of direct connections from the input to the output layer of the network.

Apart from the theoretical work of Sontag [73, 74] considering the recognition power of this variation of MLP networks in the particular case of threshold units, no work has been reported which investigates the implications, especially in practical applications, of using such direct connections. This section explores this novel network configuration (DMLP) with respect to its reliability as a pattern classifier.

The motivation for this is that the representations stored in the direct connections tend to correspond more closely to template-like representations of the training classes, instead of the normal feature based representations obtained in the hidden layer(s) of the network. As a result, this introduces an additional constraint to be satisfied by the backpropagation algorithm when looking for a solution to a problem, since the response of the network's output units to the input patterns is more directly dependent on the similarity of the patterns to a template representation of the training examples.

4.4.1 Some theoretical consideration

Consider the case of a single hidden layer network[2] with direct connections. The function computed by each output unit u_j for a given input pattern $\vec{x}_p$, $F_j(\vec{x}_p)$, can be defined as :

$$F_j(\vec{x}_p) \;=\; f(\theta \;+\; \vec{v_0} \cdot \vec{x}_p \;+\; \sum_{i=1}^{k} w_i f(net_j)), \tag{4.4}$$

where f is the sigmoid activation function, θ is the bias of unit u_j, $\vec{v_0}$ is the weight vector associated with the direct connections, k is the number of hidden units in the network and net_j is the inner product $net_j \;=\; \sum_{i=1}^{n} x_i \cdot v_{ij} - t$.

An examination of the above equation can give some insight into why the addition of direct connections can help the network to be more rigorous with respect to spurious inputs. In Chapter 1 it was seen that the weights of an MLP trained with backpropagation are modified according to the following rule :

$$\Delta_p w_{ji} \;=\; \eta \delta_{pj} o_{pi}, \tag{4.5}$$

where in the case of an output unit u_j, the value of δ_{pj} is calculated by :

$$\delta_{pj} = (t_{pj} - o_{pj}) f'(net_{pj}), \tag{4.6}$$

and in the case of a hidden unit u_k, it is given by :

$$\delta_{pk} = f'(net_{pk}) \sum_j \delta_{pj} \; w_{kj}.$$

[2]Although the observations made here reflect the case of an MLP with one hidden layer they also apply to networks with more hidden layers.

Consider initially the case of a standard MLP without any direct connections. This corresponds to having the term $\vec{v_0} \cdot \vec{x}_p$ removed from Equation 4.4. In this case, the equation for δ_{pk} for the hidden units can be further developed by replacing δ_{pj} by the corresponding values measured at the output layer, so that :

$$\delta_{pk} = f'(net_{pk}) \sum_j [f'_j(net_{pj})(t_{pj} - o_{pj})w_{kj}].$$

which leads to :

$$\begin{aligned}\delta_{pk} = f'(net_{pk}) \ [f'_1(net_{p1})(t_{p1} - o_{p1})w_{k1}] +\\ f'(net_{pk}) \ [f'_2(net_{p2})(t_{p2} - o_{p2})w_{k2}] + \ldots +\\ f'(net_{pk}) \ [f'_m(net_{pm})(t_{pm} - o_{pm})w_{km}].\end{aligned}$$

where m is the index into the output nodes.

It is seen from this equation, that the modifications in the weights from the input layer to the hidden layer are given by a combination (with weights w_{kj}) of a proportion of the errors measured at the output units representing all the different classes. Therefore, many possible different sets of weights $(w_{k1}, w_{k2}, \ldots, w_{km})$ that satisfy the equation can result from the training process. Since this is the case, the representations of the training classes that are stored in the network's hidden layer are commonly features that the learning algorithm defines to be useful for distinguishing between the classes.

In the case of the network with the direct connections, however, a different kind of representation is also obtained in the network. The modifications in the weight vector $\vec{v_0}$ are made according to Equation 4.6 and in this situation the changes in the weights for a given output unit are directly proportional to the input pattern and the error computed at that output unit only $e = (t_{pj} - o_{pj})$. In effect, what happens during training is that when the input pattern $\vec{x}_p$ is a representative of class j, the value of e is positive and the weight vector is modified towards the direction of the input. When pattern $\vec{x}_p$ is from another class, e is negative and the weight vector is again adjusted so that it moves in a direction opposite to that of the input. As a result, the kind of patterns stored in the direct connections taken on template like representations of the training classes.

An illustration of the characteristics of the representations stored in both hidden layer connections and direct connections can be obtained from the example of a DMLP simulation applied to the problem of the classification of machine printed digits. Figure 4.10 shows the weight values represented in grey-levels of a network with 20 hidden units. The first block in the diagram (the first four rows of 5 patterns each) correspond to the hidden unit representations whereas the second block (the last two rows) correspond to what each digit class' output unit sees from the input matrix. These last representations are obtained by taking the mean over the hidden layer representations weighted by the corresponding weight value from the hidden to the output layer. While the shapes of the digit classes 0 to 9 can be identified in these last images (from left to right, top to bottom), in the hidden layer images only pieces of shapes that form partial representations can be seen.

The display of the direct connection weights, on the other hand (Figure 4.11), illustrates the creation of representations that actually resemble the shapes of the training patterns. Hence, in addition to the feature based representations derived in the hidden layer, template-like patterns are also defined.

A further modification that can be made to the network is based on the observation from Equation 4.5 that the changes in the direct connections for an output unit tend to move those connections in the direction of the training patterns from that class. Since this is the case, a small change in Equation 4.5 can be thought

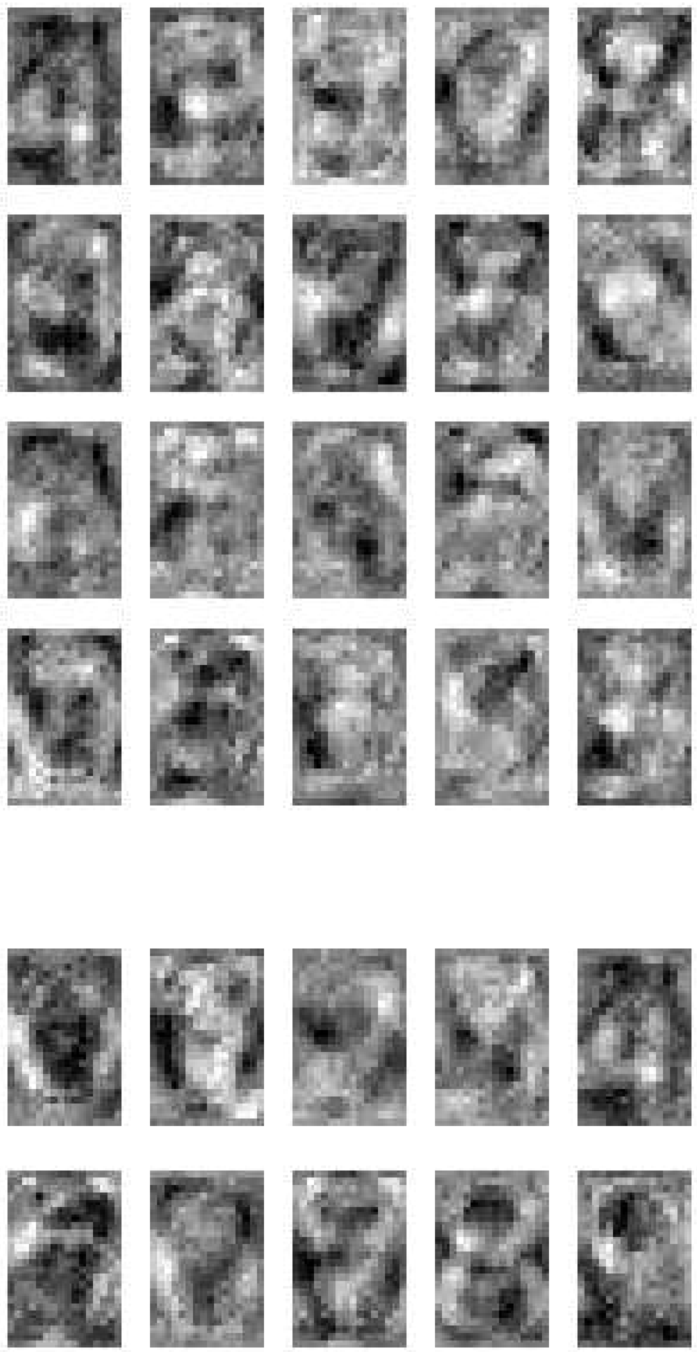


Figure 4.10: Representations stored in an MLP network. First block corresponds to hidden layer representations and second block to the representations seen from the output layer

of which only affects the acceleration of the direct connections training. This consists of using a process of incremental learning for moving the weights a fraction further towards the input, according to :

$$\Delta_p w_{ji} \;=\; \eta \delta_{pj}(o_{pi} - w_{ji}). \tag{4.7}$$

The result of this modification is that patterns produced in the direct connections correspond to a slightly smoother representation of the digits, as depicted in Figure 4.12.

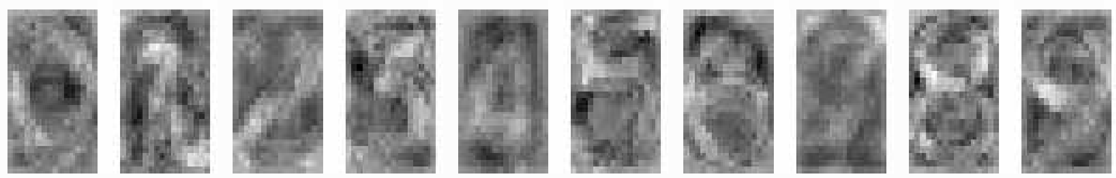


Figure 4.11: Representations stored in the direct connections

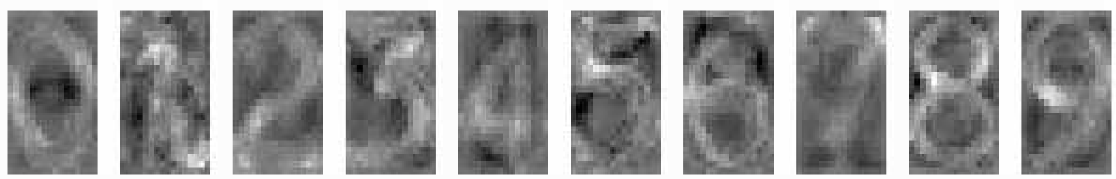


Figure 4.12: Another example of representations in the direct connections

4.4.2 Visualisation of the decision regions

The introduction of direct connections is a general idea that can be used in combination with many network variations. In what follows, it is shown how this technique can affect the decision regions of both the standard MLP and the MLP^{nor} examined in the previous section.

Figure 4.13 shows the visualisation of the decision regions generated by the DMLP in the 9 class problem. Here, it can be seen that boundaries start to appear in the pattern space surrounding the training points. Similarly to the case of the MLP^{par} network, it can be argued that this DMLP configuration should provide a more reliable structure than the MLP for rejecting invalid patterns.

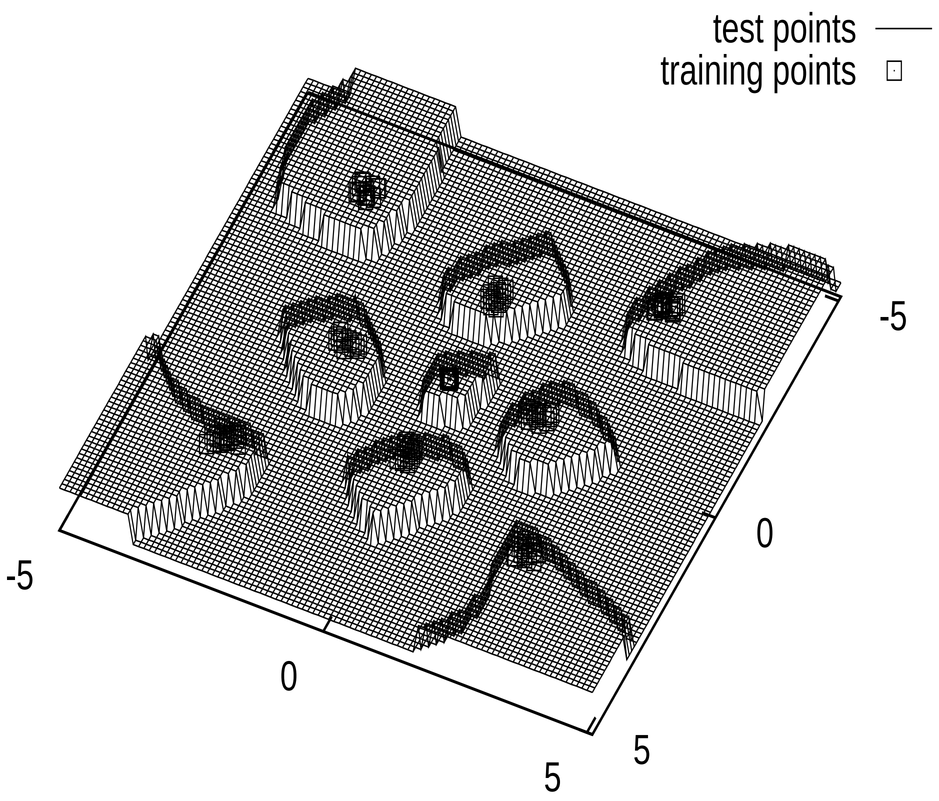


Figure 4.13: Decision regions for the DMLP (9 classes)

In Figure 4.14 the same experiment is repeated but this time for the normalised MLP network with direct connections (DMLP^{nor}). This result shows an even better definition of limited boundaries when compared to the above case.

One important aspect to observe about the inclusion of the direct connections is that the units in the network still use the same type of decision surface (hyperplanes in the case of the DMLP) to separate the classes, nevertheless, classification regions which tend to enclose the training data are seen to emerge naturally from the process of network training.

This behaviour can be explained by the fact that the responses of the output units are dependent on the direct connection representations and the errors committed by the network during training as a result of these responses not only influence the adjustment of these connections but also indirectly affect the placement of the hyperplanes in the hidden layer(s).

The output units operate in a space whose dimension is determined by the number of units in the input layer together with the number of units in the last hidden layer. The hyperplanes in this space defined from the combination of the hidden layer representations and the direct connections have to organise themselves so as to satisfy the constraint that they not only separate the training classes but also, and more importantly in this case, cause the output units to respond positively only for input patterns similar enough to the template representations in the direct connections. The result of this additional constraint is that the training patterns are much more surrounded by hyperplanes than in the case of the standard MLP. The same argument applies to the DMLP^{nor} and the fact that the decision surface computed by the units is not a hyperplane (it is a region defined by an angle between two lines) seems to create classification regions even more appropriate in terms of pattern rejection.

4.4.3 Experimental results

It is now important to establish the link between the performance observations discussed above and the rejection performance of the networks in practical pattern

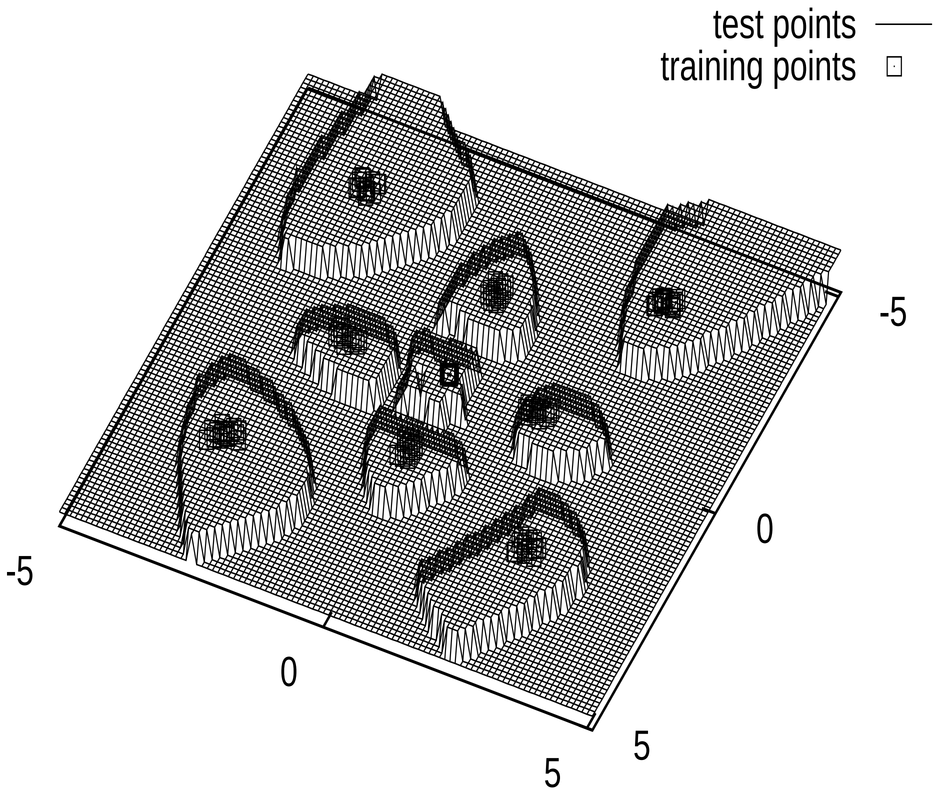


Figure 4.14: Decision regions for the DMLP^{nor} (9 classes)

recognition applications. The set of experiments in the classification of handwritten characters were conducted on the DMLP networks. In all the simulations carried out it was noted that the normalised version of the network produced a more stable and faster convergence of the training process and, therefore, was used in these extensive experiments. Two variations of the network were evaluated, the first corresponding to the normal DMLP^{nor} network and a second, denoted by DMLP^{t}, which uses the incremental learning scheme of Equation 4.7.

Networks with 10, 20 and 40 hidden units were initially tried and since the configuration with 40 units presented a slightly better performance this was the version further explored. Table 4.5 presents the correct recognition rates for the digits and also the proportion of letters rejected, obtained for networks with 40 units.

	Digit Recognition Rate			Letter Rejection Rate		
confidence	MLP	$DMLP^{nor}$	$DMLP^{t}$	MLP	$DMLP^{nor}$	$DMLP^{t}$
0	92.0	90.6	91.6	0	0	0
0.15	89.1	85.4	88.1	16	30	29
0.25	87.9	83.1	85.6	32	45	44
0.35	86.4	79.9	82.1	37	53	52
0.45	83.0	76.6	78.3	51	61	61
0.55	80.9	69.9	73.0	57	71	71
0.65	75.1	66.2	66.5	68	76	78

Table 4.5: Digit recognition rate vs letter rejection rate for the DMLPs

Considering the results obtained with the networks, with each value of the confidence level, both DMLP configurations presented a consistent increase in the letter rejection rates when compared to the standard MLP. This refers to the general ability of the networks to detect patterns different from the training classes with a fixed imposed confidence. For the case of the $DMLP^{nor}$, the rejection rates are increased by 14.3 % on average for small values of the confidence level (0.15, 0.25, 0.35) and for the $DMLP^{t}$ the improvement is by 13.3 %. For larger confidence level values (0.45, 0.55, 0.65) the rejection ability is enhanced by 10.7 % on average in the case of the $DMLP^{nor}$ and by 11.3 % in the case of the $DMLP^{t}$.

As might be expected, there is also some degradation in the digit recognition performance as a result of the more strict definition of the classification surfaces. In this respect, however, the $DMLP^{t}$ presented a much improved performance when compared to the $DMLP^{nor}$, while maintaining the same enhancement in rejection performance. While for the $DMLP^{t}$ the reduction in the digit recognition rate for small values and for larger values of the confidence level is smaller, around

2 % and 7 %, respectively, for the DMLP^{nor} it is 4.1 % and 8.7 %, respectively.

Another way of looking at the results is to observe the letter rejection rates obtained for each network with respect to a particular digit recognition performance; although this kind of evaluation is more related to the general classification capacity of the networks (achieved with respect to a particular fixed learning set) than to the their natural ability to distinguish between different patterns with a required level of confidence. From this point of view, while there are cases of clear improvement in terms of general classification capabilities with the DMLP^t, in some other cases, the enhancement in performance is not very evident. For example, for a digit recognition rate of 89.1 % and rejection rate of 16 % with the MLP, a similar recognition rate of 88.1 % with the DMLP^t corresponded to a letter rejection rate of 29 %. In another situation, for 80.9 % of digit recognition, the MLP provided 57 % of letter rejection while the DMLP^t provided 78.3 % of digit recognition and 61 % of letter rejection.

One very important point to make about the experiments carried out is that, for a fixed network configuration, while recognition performance can be boosted by the use of a larger training set, therefore approximating even more the performance of the DMLP^t to that of the MLP, rejection capacity cannot be increased by the same procedure so as to make the performance of the MLP approximate that achieved by the DMLP^t. Improvement in rejection performance is only possible with the introduction of structural modifications, such as the use of the direct connections. The recognition performance of the DMLP^t can also be improved with a larger network, as evidenced below.

Additional trials were performed with the DMLP^t configuration and with the standard MLP, this time employing 80 hidden units, to see if any improvement was observed when compared to the configuration with 40 units. A DMLP^t configuration with 120 units was also simulated and the complete set of results obtained is summarised in Table 4.6. These results illustrate a general improved compromise

	Digit Recognition Rate			Letter Rejection Rate		
confidence	MLP	$DMLP^t_{80}$	$DMLP^t_{120}$	MLP	$DMLP^t_{80}$	$DMLP^t_{120}$
0	92.0	92.2	92.7	0	0	0
0.15	89.4	88.3	89.5	17	29	26
0.25	88.2	85.7	87.4	25	42	39
0.35	84.4	82.9	83.6	32	53	50
0.45	80.8	78.4	82.1	46	62	60
0.55	81.3	73.6	76.8	48	71	69
0.65	72.9	66.5	71.9	64	79	74

Table 4.6: Another set of results for the $DMLP^t$

between the letter rejection and the digit recognition rates when compared to the MLP.

The classification rates obtained with the MLP with 80 units represented a reduction in rejection performance in comparison with the configuration with 40 units. With respect to the $DMLP^t$ with 80 units ($DMLP^t_{80}$), the enhancement in the rejection rates at each value of the confidence level is by 17.3 % on average (in comparison with the MLP with the same number of units) whereas the digit recognition rates drops by 3.1 %. The $DMLP^t$ with 120 units ($DMLP^t_{120}$) also showed a consistent increase in rejection performance (by 14.2 % on average) but with the important characteristic of practically maintaining (a decrease by only 0.7 %) the same digit recognition rates obtained with the standard MLP.

A further examination of the values in Table 4.6 also shows that, even in terms of general classification capacity, the performance of the DMLPs exceeds that of the standard MLP. At similar recognition rates obtained for both the MLP and the DMLPs, the rejection rates of the DMLPs are superior. Consider, for example, the case of the $DMLP^t_{120}$. For a digit recognition rate of 89.4 % and 89.5 % with the MLP and the DMLP^t_{120}, respectively, the letter rejection rates are 17 % and 26 %, respectively. In another situation, for 84.4 % of digit recognition with the MLP and 83.6 % with the DMLP^t_{120}, the rejection rates are 32 % and 50 %, respectively. Other comparisons can be obtained from Table 4.6 and a similar evaluation can be made for the case of the DMLP^t_{80}.

Another important information extracted from the experiments refers to the digit error rates and digit rejection rates achieved by the different networks. These results, illustrated in Table 4.7, indicate an overall reduction in the digit error rates in the case of the DMLPs. A general comparison between the DMLPs and the MLP also shows an advantageous trade-off in favour of the DMLPs, in terms of recognition and rejection performance obtained with a fixed error rate. For example, comparing the DMLP^t_{120} with the MLP for an error rate of 1.7 % (i.e. the confidence level is set to 0.65 in the MLP), the digit recognition rates obtained are 72.9 % for the MLP and 78.4 % for the DMLP^t_{120}, with similar letter rejection rates of 64 % for the MLP and 60 % for the DMLP^t_{120}. In another case, for an error rate of 2.6 %, the digit recognition rate for the MLP is 80.8 % and the letter rejection rate is 46 %, while the DMLP^t_{120} presents 83.6 % of digit recognition and 50 % of letter rejection.

From the results obtained with the experimental work, it can be concluded that the introduction of direct connections represents a real enhancement in the rejection capabilities of MLP networks.

4.5 The Gaussian MLP Network (GMLP)

The unreliability of standard MLPs with respect to the rejection problem, caused by the creation of open decision regions in the input space, can be partially attributed to the type of activation function used in the network (i.e. the use of an

confidence	Digit Error Rate			Digit Rejection Rate		
	MLP	$DMLP^t_{80}$	$DMLP^t_{120}$	MLP	$DMLP^t_{80}$	$DMLP^t_{120}$
0	8.0	7.8	7.3	0	0	0
0.15	5.3	5.1	5.0	5.3	6.6	5.6
0.25	5.3	3.4	3.7	6.5	10.9	8.9
0.35	3.7	2.2	2.5	11.9	14.9	13.9
0.45	2.6	1.6	1.8	16.6	20.0	16.1
0.55	3.0	1.1	1.1	15.7	25.3	22.1
0.65	1.7	0.8	0.9	25.4	32.7	27.2

Table 4.7: Digit error rate vs digit rejection rate for the $DMLP^t$

inherently "open" function). This implies that the use of a different activation function might profitably be considered. A multilayer network which employs a Gaussian activation function (GMLP) instead of a sigmoid function in combination with an inner product as the propagation rule has recently received some attention [17, 21]. In this network, a typical activation function is :

$$a_j \;=\; f(net_j) = exp(-net_j^2/c_j^2), \tag{4.8}$$

where the parameter c_j controls the width of the function and net_j is the inner product :

$$net_j \;=\; \sum_{i=1}^{n} x_i \;\cdot\; w_{ij} \;-\; \theta,$$

where θ is the bias term.

The idea of using a non-monotonic function like the Gaussian as the activation function of a feedfoward network is not completely new and dates back to the work of Ballard in 1986 [2]. What is new, however, is the recent observed fact, contrary to previous assertions [66] about the need for using only monotonic activation functions in the MLP trained with backpropagation, that a network of Gaussian units of the type described can be trained using the generalised delta rule (backpropagation).

It has been shown that this modified architecture is capable of approximating the same sort of functions as MLPs and RBF networks [21] and it is claimed to provide faster training times and to reduce the number of units necessary for particular problems [17]. This network is examined in this chapter with respect to the problem of spurious pattern detection and it is shown how the substitution of the sigmoid by the Gaussian function contributes to the enhancement of its rejection capabilities.

With the substitution of the sigmoid by a Gaussian function, the receptive field of each network unit is transformed into a kind of hyper-hill in the pattern space which forces the unit to generate strong responses only if its input falls within a certain range. This is the result of the combination of the inner product as the propagation rule and the Gaussian as the activation function, which creates a semi-local receptive field such as that illustrated in Figure 4.15. As can be seen, points falling along the discriminant line perpendicular to the unit's weight vector produce the same unit output and as they move in parallel to the weight vector the unit response becomes weaker. When projected onto the plane, the decision region defined by the unit is bounded by two parallel hyperplanes.

4.5.1 Training GMLP networks

The motivation for using a Gaussian function in the MLP is combined with the idea of applying the generalised delta rule as the learning procedure for the network. As described in Chapter 1, the generalised delta rule uses the difference

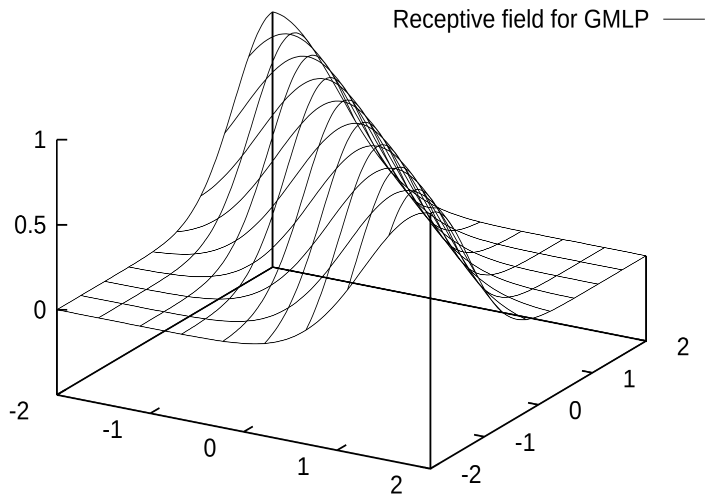


Figure 4.15: Combination of the inner product with the Gaussian activation function

between the actual output of the network and a given desired output as the basis for changing the network's weights. The error at the output layer of the network is given by :

$$E_p \; = \; \frac{1}{2}\sum_{j=1}^{n}(t_{pj} \; - \; o_{pj})^2. \tag{4.9}$$

where o_{pj} is the actual output of unit u_j and t_{pj} is the desired output.

It was also seen that for reducing the network errors two rules have to be formulated which define, respectively, how the connection weights leading directly to the output layer are modified as a function of the errors measured at the output layer, and how the connection weights leading to each existing hidden layer can be changed recursively in terms of the error signals of the individual units of the layer to which it directly connects and the weights of those connections. Once the errors at the output layer are defined, they can be backpropagated through the preceding layer and when the errors at the preceding layer are defined they can be backpropagated again to the next layer. This process goes on until all the weights in the network are adequately modified. Consider again the equation for changing the weights, defined as :

$$\Delta_p w_{ji} \;=\; \eta \delta_{pj} o_{pi}, \tag{4.10}$$

where δ_{pj} is calculated in the case of an output unit by :

$$\delta_{pj} = (t_{pj} - o_{pj}) f'(net_{pj}),$$

and in the case of a hidden unit by :

$$\delta_{pj} = f'(net_{pj}) \sum_k \delta_{pk}\, w_{kj}.$$

Now, for training the GMLP with backpropagation, the derivative $f'(net_{pj})$ in the above equations need to be properly redefined to account for the change in the activation function. This is given by :

$$f'(net_{pj}) \;=\; -2\; net_{pj}\; a_j.$$

The training process can then go on as before with the introduction of some small modifications. Dawson and Schopflocher [17] propose that, for avoiding the problem of easy local minima being reached by the learning algorithm, an augmented error function of the form presented below should be used instead of the plain version of the function (Equation 4.9).

$$E_p \;=\; \frac{1}{2}\sum_{j=1}^{n}(t_{pj} \;-\; o_{pj})^2 \;+\; \frac{1}{2}\sum_{j=1}^{n} t_{pj} \cdot (net_{pj})^2.$$

The reason for this is the observed fact that for an activation function like the Gaussian ($f(net_j) = exp(-net_j^2/c_j^2)$, the derivative rapidly goes to zero as the net_j input reaches extreme values. The addition of term $t_{pj} \cdot (net_{pj})^2$ elevates the derivative of the error with respect to the net input such that in situations where the derivative tends towards zero (making learning difficult), the extra term takes a strong role in the expression preventing it from become null and allowing learning to proceed. In fact, as noted by Flake [21], this modification in the error function is simply equivalent to the addition of a small constant to the derivative function $f'(net_{pj})$, so that :

$$f'(net_{pj}) \;\simeq\; -2\; net_{pj}\; (a_j + \epsilon).$$

The difference, however, is that the augmentation of the first derivative by the constant is applied to each unit in the network whereas the modified error function has an effect only on output units. In the latter case, the units in the hidden layers can still suffer from the same problem of the zero derivative, but both constructions are potential solutions to the problem. The simulations carried out in this chapter all use the first formulation.

4.5.2 Visualisation of the decision regions

The result of the 2-D visualisation experiment for the 2 class problem is illustrated in Figure 4.16. In this simulation, the training patterns from one of the classes are confined to the region limited by the two parallel lines in the middle of the diagram and the other points are considered to belong to the second class. The decision regions identified in the diagram show, however, that in this particular case one of the classes still accepts patterns distant from the training points as genuine members of the class. This is because, as explained in subsection 4.2.3, the hidden units in the network that are actually responding to the patterns from that class are doing so by acting as negative detectors of the second class. Although this situation can still be considered more reliable for rejecting spurious patterns than that obtained with the standard MLP, because the rejection areas are indeed increased, there are ways of controlling the distribution of the responses of the hidden units so that there are positive detectors of both classes.

The result of a simulation considering the distribution of positive detectors of both classes is illustrated in Figure 4.17, where it can now be seen that limited acceptance regions for the two classes have been created. It is important to mention the fact that in all the practical experiments reported in the next section it was actually observed that an undesirable situation like this is unlikely to happen, especially when there are an increasing number of training classes present in the problem.

Repeating the visualisation experiment for the GMLP, this time for a problem with 9 classes, generates the characteristics shown in Figure 4.18. In this case, the decision regions created by the network tend to correspond to closed regions with a much more limited area of the input space associated with each class, when compared to the case of the standard MLP (see Figure 4.3). The problem observed with the standard MLP is minimised with the GMLP since the rejection areas are substantially increased.

4.5.3 Experimental results

The handwritten character classification experiments are now executed with the GMLP network. The previous results obtained with the standard MLP, the DMLP^t and with the negative training approach (MLP^{neg}) described in Chapter 2 are also re-presented in order to allow a final comparison between the various approaches.

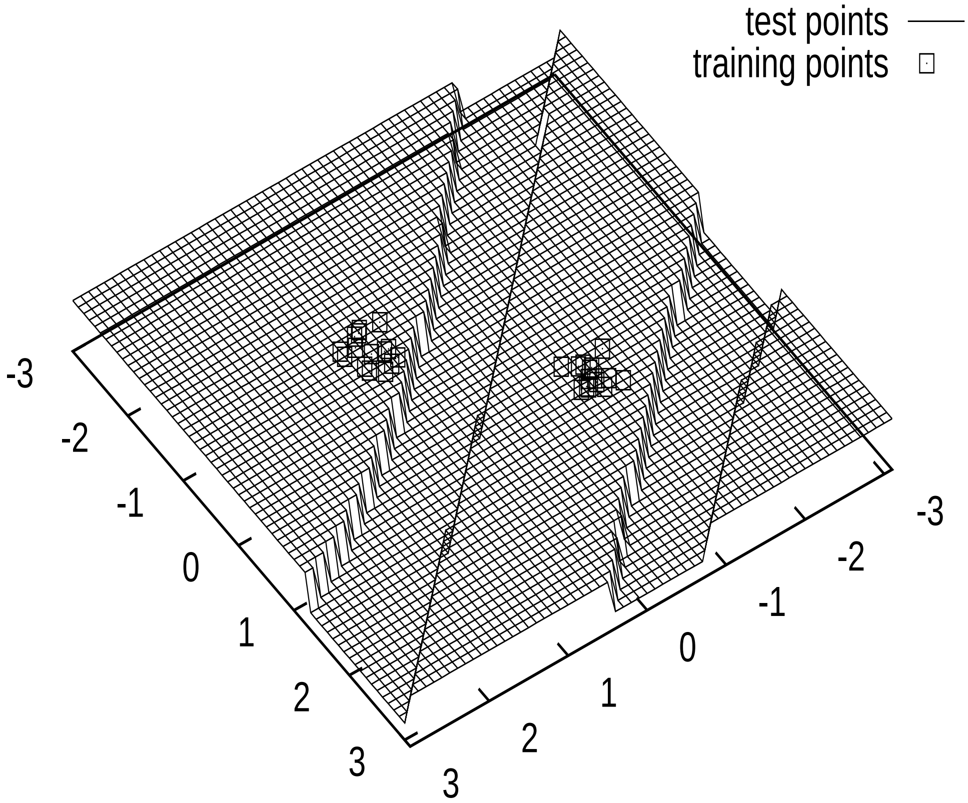


Figure 4.16: Decision regions for the GMLP (2 classes)

GMLP networks were also simulated with a different number of hidden units but the networks with 80 units presented the best results. Tables 4.8 and 4.9 present the classification results obtained for these networks. Table 4.8 shows the correct recognition rates for the digits and also the proportion of letters rejected. Table 4.9 illustrates the percentage of digits erroneously classified together with the percentage of digits rejected by the network.

In the case of both the GMLP and the DMLP^t, a consistent increase in the letter rejection rate is observed when compared to the standard MLP as well as in comparison to the MLP^{neg} network (whose performance is very irregular). For the case of the GMLP, the rejection rates are increased by 20 % on average for small

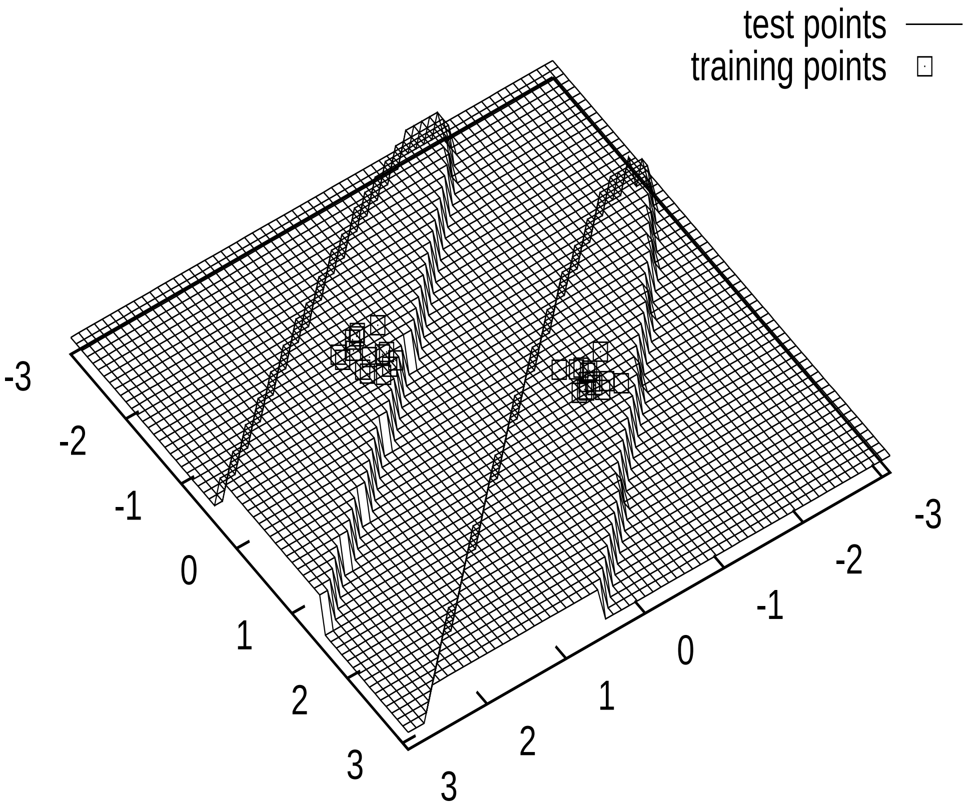


Figure 4.17: Another example of decision regions for the GMLP (2 classes)

values of the confidence level (0.15, 0.25, 0.35) when compared to the standard MLP, whereas for larger confidence level values (0.45, 0.55, 0.65) the rejection ability is enhanced by 21 % on average. There is also some degradation in the digit recognition performance. The reduction in the digit recognition rate for small values of the confidence level is smaller, around 1.4 %, whereas for larger values the degradation in digit recognition is greater (around 6 %). For the case of the $DMLP^t$ network, the increase in letter rejection for small values of the confidence level is by 17 % with a decrease in the digit recognition by 1.3 %. For larger values of the confidence level, the rejection rate is enhanced by 18 % whereas the digit recognition drops by 5.5 %.

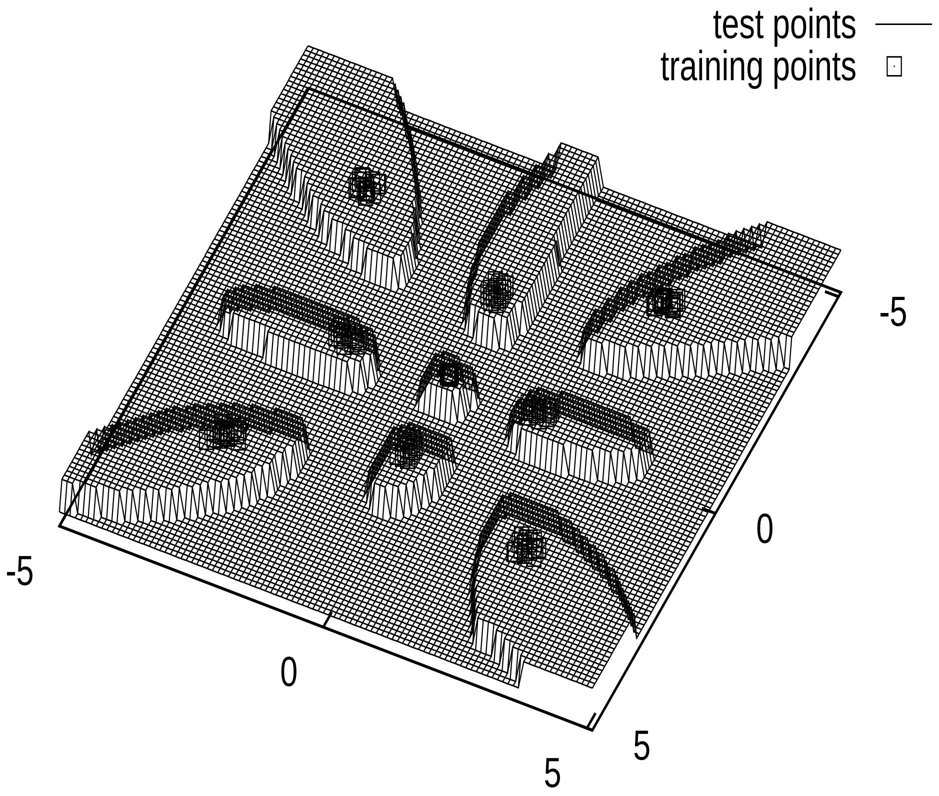


Figure 4.18: Decision regions for the GMLP (9 classes)

One very relevant aspect to consider regarding the GMLP and the DMLP^t, shown in Table 4.9, is that there is a reduction in the rate of misclassified patterns for all values of the confidence level introduced, especially in the case of the GMLP (except where there is no confidence level imposed on the network, value = 0). For instance, while the digit error rate is 5.3 % for the standard MLP, with the value of 0.15 for the confidence level, it is 3.8 % for the GMLP. This is a result of the fact that some patterns, for which the network has no confidence about their identity, are being rejected by the GMLP instead of being erroneously classified.

With respect to the general classification capacity of the GMLP, in comparison with the MLP in terms of the letter rejection rates obtained at similar rates

	Digit Recognition Rate				Letter Rejection Rate			
conf.	MLP	MLP^{neg}	GMLP	$DMLP^{t}$	MLP	MLP^{neg}	GMLP	$DMLP^{t}$
0	92.0	91.5	91.0	92.2	0	0	0	0
0.15	89.4	87.2	87.5	88.3	17	28	36	29
0.25	88.2	86.2	86.7	85.7	25	36	44	42
0.35	84.4	80.3	83.5	82.9	32	48	55	53
0.45	80.8	79.4	77.6	78.4	46	56	65	62
0.55	81.3	80.2	71.4	73.6	48	53	76	71
0.65	72.9	78.4	67.5	66.5	64	60	80	79

Table 4.8: Digit recognition rate vs letter rejection rate for the GMLP

of recognition performance, the GMLP presented a clear improvement over the standard MLP network. The following associations can be used as examples. For a 88.2 % of digit recognition rate obtained with the MLP and 87.5 % obtained with the GMLP, letter rejection rates of 25 % and 36 % are produced by the two networks, respectively. In another situation, a letter rejection rate of 55 % is provided by the GMLP with a recognition rate of 83.5 %, as opposed to a much inferior rejection rate of 32 % and similar recognition rate of 84.4 % with the MLP. Another important fact is that, in both examples, the error rates observed with the GMLP are smaller than those produced by the MLP. In the first case, while the GMLP presented an error rate of 3.8 %, the error rate with the MLP was of 5.3 %. In the second case, an error rate of 0.9 % was obtained for the GMLP in contrast with a 1.7 % rate for the MLP.

It becomes readily apparent that GMLPs and $DMLP^{t}$s provide a more reliable structure for rejecting invalid patterns than the standard architecture, with the GMLP presenting the best overall performance among all the networks tested. However, as might be expected, considering the results observed at fixed confidence levels, there is a small trade-off between enhancing the rejection capabilities

conf.	Digit Error Rate				Digit Rejection Rate			
	MLP	MLP^{neg}	GMLP	DMLP^t	MLP	MLP^{neg}	GMLP	DMLP^t
0	8.0	9.5	9.0	7.9	0	0	0	0
0.15	5.3	5.7	3.8	5.1	5.3	7.1	8.7	6.6
0.25	5.3	4.3	2.9	3.4	6.5	9.5	10.4	10.9
0.35	3.7	3.5	2.0	2.2	11.9	16.2	14.5	14.9
0.45	2.6	2.2	1.4	1.6	16.6	18.4	21.0	20.0
0.55	3.0	2.6	0.9	1.1	15.7	17.2	27.7	25.3
0.65	1.7	1.7	0.6	0.8	25.4	19.9	31.9	32.7

Table 4.9: Digit error rate vs digit rejection rate for the GMLP

and losing some performance in the recognition rates of valid patterns (although the error rates are also accordingly reduced). The choice of which network to use should be made depending on the parameters imposed by a specific application. Standard MLPs tend to maximise generalisation capabilities with respect to a given training set but can be very unreliable in use, making them unsuitable for problems which demand a high degree of reliability.

Another aspect to take into account, as mentioned before with respect to a fixed architecture, is that while the use of a larger training set can further enhance the recognition performance of the GMLP and the DMLP^t, making them even more attractive in terms of their general classification performance when compared to the MLP, it cannot have the same positive effect on increasing the rejection capabilities of the standard MLP.

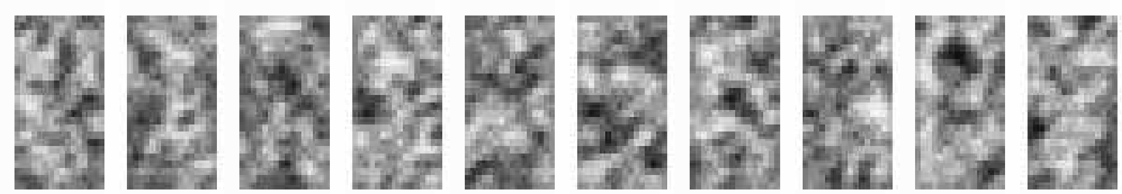
4.6 Inversion of the Different Configurations

In Chapter 2 and Chapter 3, it was described how the use of the technique of network inversion could assist in the observation of the tolerance of the networks with respect to the appearance of the patterns accepted by it. In this section, the inversion technique is used again to compare the visual characteristics of the inverted patterns obtained with the two networks that presented the best performance in the character classification experiments, the $DMLP^t$ and the GMLP network.

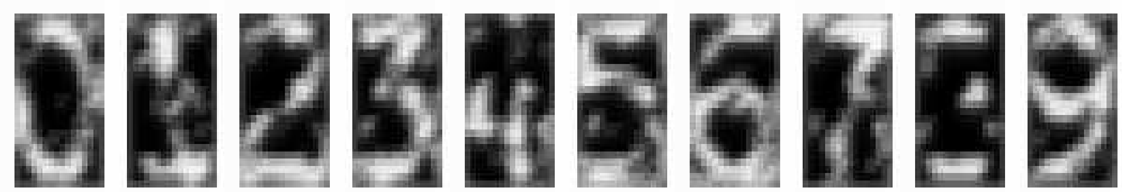
First, the inversion of the $DMLP^t$ architecture was implemented following the same scheme for network inversion presented in Chapter 2 and the result of several runs showed a substantial improvement in the appearance of the patterns accepted by the network. Figure 4.19(b) illustrates the result of one of the trials and it can be seen that the inverted patterns approximate much more the usual shapes of digits, indicating that the network is much less tolerant to deformed or unknown patterns. In Figure 4.19(a), the inversion of the standard MLP is also given, allowing a better comparison between the results.

For the case of the GMLP network, the result of several trials also demonstrated a clear enhancement in the shapes of the inverted patterns (Figure 4.19(c)), when compared to the standard MLP, although less apparent than that attained with the $DMLP^t$. Such an analysis suggests that the observations made on the simple 2D-problem scale up to higher dimensions and it also corroborates the results of the practical experiments reported in the previous section.

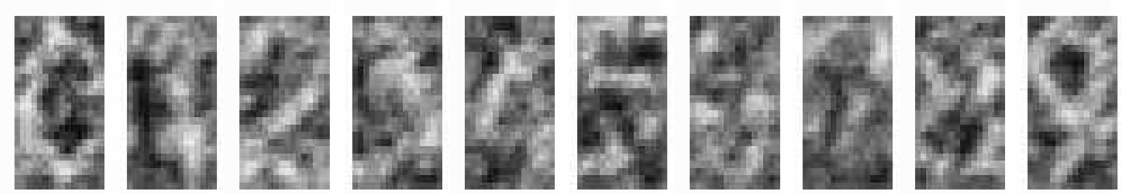
The experiments of network inversion provide in conjunction with the results obtained in the problem of the classification of handwritten characters a more complete examination of the rejection capabilities of the different MLP configurations. These two sets of experiments complement each other. While the classification



(a)



(b)



(c)

Figure 4.19: Inverted patterns : (a) MLP, (b) DMLP^t, (c) GMLP

of handwritten characters examines the performance of the networks in a difficult situation where the objective is to check their ability to reject patterns that share many similarities with the training classes, but still are known to belong to different classes, the network inversion experiments allow the observation of the general characteristics of the patterns accepted as valid by the network and, therefore, provide a means of checking the tolerance of the networks with respect to patterns very different from the training classes.

4.7 Conclusion

The work reported in this chapter has investigated the problem of the detection of spurious inputs from the perspective of enhancing network capability based on the built-in properties of the network itself. In this case, the imposition of a threshold for measuring the confidence level of the network's outputs is the criterion used to accept or reject an input pattern.

It has been observed that the type of similarity measure computed by the processing units in the network plays an important role in its ability to reject spurious patterns. Units that compute global mappings of the input space, as in the case of standard MLPs, tend to classify into one single class extensive areas of the input space. Here, the simple application of a high confidence level for acceptance of a decision response is not sufficient, since in these conditions even patterns with very random characteristics are still confidently classified by the network.

Basically, four potential strategies for improving the MLP performance were examined. The first, based on the idea of transforming the receptive field of the processing units into hyper-spheres (MLP^{par}), although it has not generated the expected performance in the test application considered (the classification of handwritten characters) is still a potential solution to the problem that can be employed in other applications. The second strategy, using a network that employs the normalisation of the propagation rule also did not show the improvement expected but proved to be very useful when applied in conjunction with the MLP network with direct connections.

Finally, the last two strategies examined produced the best practical results and confirmed the theoretical observations. The GMLP network investigated represents a possible way of overcoming or, at least, minimising the spurious pattern problem, where the combination of the inner product as the network's propagation rule and the Gaussian activation function creates a semi-localised receptive field in the pattern space. This network can offer the benefits of a much more enhanced structure for rejecting spurious data when compared to the standard MLP with only a small degradation in the recognition of valid patterns.

The other alternative configuration, the DMLP network, enhances the network's rejection ability through the imposition of additional constraints on the learning algorithm for the definition of the decision regions during the training process. The introduction of the direct connections allows for the generation of "template-like" representations in these connections which tend to force the learning algorithm to create boundaries surrounding the training examples.

Next chapter will look at another type of feedforward network, the model of radial basis function networks (RBFs), and practical experiments will be used to compare this model with some of the MLP configurations explored in this chapter.

Chapter 5

Radial Basis Function Networks

5.1 Introduction

Radial basis function networks (RBFs) are together with the multilayer perceptron among the most widely used neural networks in practical applications. Many researchers have contributed to the theory and design of radial basis function models [7, 52, 54, 59] as well as in its application to many real world problems such as speech recognition [54, 61], medical diagnosis [45], radar point-source location [82] and handwritten character recognition [40], to mention just a few.

This chapter examines the model of radial basis function networks with respect to the rejection problem. RBF networks are compared to both the standard MLP and to the GMLP and DMLP networks described in the previous chapter. The motivation for this comparison is the fact that RBF networks appear naturally as very strong candidate structures for the reliable detection of spurious inputs, due to the characteristics of the processing units in the network which compute localised functions of the pattern space.

The classification problem in the simple 2-dimensional space is used first to visualise the shape of the decision regions generated by RBF networks and, subsequently, some practical results are presented with RBFs in the same context of the classification of handwritten alphanumeric characters described in the previous chapters. First, however, the mathematical background for the design of RBF networks is presented.

5.2 Theoretical Background

The design of a feedforward network based on the approach of radial basis functions presents a very different characteristic from that of a multilayer perceptron network. While the definition of an MLP through a learning algorithm can be viewed as a statistical optimisation process, the design of an RBF network is made from the perspective of a function approximation problem where a surface in a multidimensional space (corresponding to a function f) is defined by a linear combination of basis functions (ϕ) that provide a fit to the training data.

In the case of the standard MLP, the classification decisions created by the network to partition the pattern space are built from hyperplanes defined by the inner product ($\sum_i w_i \cdot x_i$) between the input and the weight of the processing unit which are then used as the input to a non-linear activation function (e.g the sigmoid). In a RBF network, in contrast, the classification regions correspond to *hyperellipsoids* which are the result of functions of the form $\phi(|\vec{x}_p - \vec{c}|)$, which compute a distance measure between the input pattern $\vec{x}_p$ and a given *center* $\vec{c}$.

Given n pairs of input/output associations $(\vec{x_p}, y_p)$, the function f computed by an RBF network can be defined as :

$$f(\vec{x_p}) = \sum_{j=1}^{k} w_j \cdot \phi(|\vec{x_p} - \vec{c_j}|) \tag{5.1}$$

where ϕ is a basis function, $\vec{c_j}$ are the centers that have to be defined according to the training data and w_j are the coefficients (weights) of the basis functions.

The function ϕ can take various forms but, usually, it is defined to be a Gaussian function, i.e.

$$\phi(r) = exp(\frac{-r^2}{d^2}) \tag{5.2}$$

where d controls the width of the function.

And the function $|\ldots|$, corresponding to the distance measure, is usually chosen to be the Euclidean distance :

$$|\vec{x} - \vec{c}| = \sqrt{\sum_i (x_i - c_i)^2} \tag{5.3}$$

The representation of this description in a feedforward network can be illustrated in Figure 5.1. The network is composed fundamentally of three layers : an input layer, a hidden layer and an output layer. The elements in the input layer do not compute any function and, as in the case of an MLP, serve only the purpose of passing the input pattern to the network. The k processing units in the hidden layer compute the radial functions ϕ as defined above. The outputs of these units are then combined linearly in the output layer through the connection weights w_j. The output unit computes therefore a function f which is an approximation of the n input/output pairs $(\vec{x_p}, y_p)$.

In a typical classification problem, the units in the output layer implement, in fact, a logistic activation function (e.g the sigmoid) instead of the linear function, so that the outputs of the units are limited to the range [0,1]. The other extension is that there can be as many units in the output layer as are necessary to compute different functions f, each one representing a training class (in a classification problem, for example).

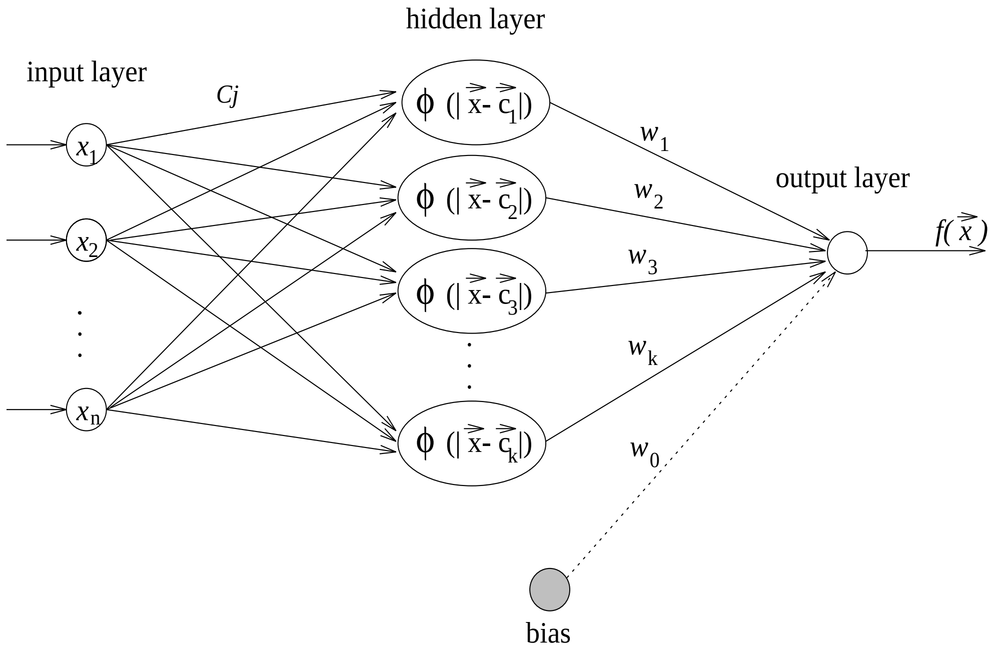


Figure 5.1: RBF network

5.3 Training RBF Networks

There are many possible strategies for training an RBF network especially because there are three different sets of parameters that have to be estimated : the widths d of the basis functions, the centers $\vec{c_j}$ and the weights w_j from the hidden to the output layer. The widths d can be set to a constant value prior to the training process, can be defined through a gradient descent technique or through a heuristic process. This third situation, usually referred to as the nearest neighbour heuristic, has been used very frequently and consists basically of defining each width to be proportional to the distance to its nearest neighbour.

For the definition of the centers $\vec{c_j}$, one possibility is to evenly or randomly spread the basis centers throughout the input space in order to cover it almost completely. This strategy is only feasible when the number of dimensions in the problem is small. Another more realistic technique is to choose randomly a sample of patterns from the training classes and use them as the centers of the functions. This strategy has been used in many applications and has been shown to work effectively. The third method corresponds to training techniques based on competitive learning, feature map techniques and genetic algorithms [27]. The objective in this case is to find gradually a distribution for the centers that approximates in the end the distribution of the input patterns.

After the widths and the centers are defined, all that is left to be estimated are the weights w_j in the output layer of the network. It is here that one of the great advantages of RBF networks becomes evident. Since these are parameters of a linear function, the problem of finding them can be solved by a linear optimisation technique where a solution of global minimum is, in contrast with the MLP, guaranteed to be obtained. The training of the weights can be achieved either by an error-correcting scheme : a gradient descent technique such as the least-mean-square algorithm (LMS) [27] or a one-dimensional Newton-like method [21] or by a straightforward procedure of matrix manipulation known as the *pseudo-inverse method* [7, 27]. The latter approach is the one described below since this is the method employed in the practical experiments reported later in the chapter.

The problem of training an RBF network to define a function f that approximates n associations $(\vec{x_p}, y_p)$ is viewed as the problem of minimising the error in the output layer of the network, which can be defined as :

$$E[f] = \sum_{p}^{n}(y_p - f(\vec{x_p}))^2 + \lambda\|Pf\|^2 \tag{5.4}$$

where the first part of $E[f]$ $(\sum_p^n(y_p - f(\vec{x_p}))^2)$ accounts for the condition for the minimisation of the error of the approximation and the second part $(\lambda\|Pf\|^2)$ corresponds to a stabilisation factor which controls the smoothness of the approximation.

The minimisation of Equation 5.4 with respect to the weights w_j in this equation (remember that $f(\vec{x_p}) = \sum_{j=1}^{k} w_j \cdot \phi(|\vec{x_p} - \vec{c_j}|)$) is shown to be calculated [27] by the following matrix operation :

$$\vec{w} = (G^T \cdot G + \lambda G_0)^{-1} \cdot G^T \cdot \vec{y} \tag{5.5}$$

where G is the matrix :

$$G = \begin{bmatrix} \phi(|\vec{x_1} - \vec{c_1}|) & \phi(|\vec{x_1} - \vec{c_2}|) & \dots & \phi(|\vec{x_1} - \vec{c_k}|) \\ \phi(|\vec{x_2} - \vec{c_1}|) & \phi(|\vec{x_2} - \vec{c_2}|) & \dots & \phi(|\vec{x_2} - \vec{c_k}|) \\ \vdots & \vdots & \vdots & \vdots \\ \phi(|\vec{x_n} - \vec{c_1}|) & \phi(|\vec{x_n} - \vec{c_2}|) & \dots & \phi(|\vec{x_n} - \vec{c_k}|) \end{bmatrix}$$

and G_0 is defined as :

$$G_0 = \begin{bmatrix} \phi(|\vec{c_1} - \vec{c_1}|) & \phi(|\vec{c_1} - \vec{c_2}|) & \dots & \phi(|\vec{c_1} - \vec{c_k}|) \\ \phi(|\vec{c_2} - \vec{c_1}|) & \phi(|\vec{c_2} - \vec{c_2}|) & \dots & \phi(|\vec{c_2} - \vec{c_k}|) \\ \vdots & \vdots & \vdots & \vdots \\ \phi(|\vec{c_n} - \vec{c_1}|) & \phi(|\vec{c_n} - \vec{c_2}|) & \dots & \phi(|\vec{c_n} - \vec{c_k}|) \end{bmatrix}$$

The parameter λ can be set to 0 [27] and in this case Equation 5.5 becomes equivalent to the computation of the Moore-Penrose inverse matrix. This matrix represents the best solution of an undetermined system composed of linear equations.

The advantage of RBF networks is exactly the fact that the linear weights w_j can be computed directly as indicated above. However, the success of the matrix procedure is dependent upon a good choice of the other network parameters, especially of the centers (weights) of the hidden layer. For this reason it is frequently recommended after the definition of the weights w_j with the pseudo-inverse method the use of a gradient descent procedure (for example, the LMS algorithm) to append a few training cycles to the network, so that all the parameters can be fine tuned to better approximate the input/output mappings.

5.4 A Comparison Between RBFs and MLPs

Although superficially, RBF and MLP networks can present some similarities, which make even possible the development of a general framework that integrates both architectures [18], there are in fact fundamental differences that can be easily spotted by looking at the two networks. The first important characteristic is the fact that an RBF network is usually composed of a fixed architecture of three layers, each one of them with a very different purpose. The first layer is made up of source units (that only pass the inputs to the network) as is in the case of the MLP. The hidden layer implements the basis functions that actually fit the training data and the output layer only combines linearly the outputs of the hidden nodes. In the standard MLP, there is no fixed number of layers. This depends essentially on the characteristics of the problem being investigated. Although in theory an MLP with only one hidden layer is sufficient to define any form of complex decision region [44], it is observed that in some applications the use of more hidden layers can be appropriate. Apart from the input layer, regardless of the number of layers used in the network, there are no fundamental differences between the intermediate layers themselves nor between them and the output layer, in terms of the purpose served by each one of the layers.

The fact that the output layer of an RBF computes a linear function is also a crucial difference between the two networks. Training in RBFs can be extremely fast because after the hidden layer weights are defined (which can also be done quickly) the output layer weights can be computed by the direct pseudo-inverse method. This is only possible because the hidden processing units implement local functions. In effect, what happens is that the radial functions expand the inputs into a space of higher dimension where the patterns are now linearly separable. It is precisely because of this, however, that it is usually necessary to use many more processing units in the hidden layer of an RBF network than is required in a MLP network for the same problem. Since the volume of a space increases exponentially according to its dimension, the number of radial functions necessary in the hidden layer to maintain the same level of approximation capacity observed in lower dimensions must expand at the same rate. For example, in some practical applications using RBFs thousands of units need to be employed to maintain at the same generalisation level what would be achieved by a hundred units in an MLP (see the experiments described later in this chapter).

A further fundamental difference between RBF and MLP networks is the main object of study in this chapter. This is that the local functions used in RBFs can bring a practical benefit to the network with respect to the spurious pattern problem. This property is clarified with the experiments described in the next sections.

5.5 Implications for the Rejection Problem

From Equations 5.2 and 5.3 it is seen that, in contrast with the global function used in the units of a standard MLP, each node in the hidden layer of an RBF network computes a localised mapping of the input space defined by the combination of a distance metric as its propagation rule and a Gaussian (usually) as its activation function. The effect of this combination, as illustrated in Figure 5.2, is the creation of a receptive field which forces the unit to respond positively to only a restricted region of the input space close to a center stored in its weights. The output of the unit is maximum when the input pattern $\vec{x_p}$ has a zero distance to the center $\vec{c_j}$ and decreases rapidly as the input becomes more distant from it.

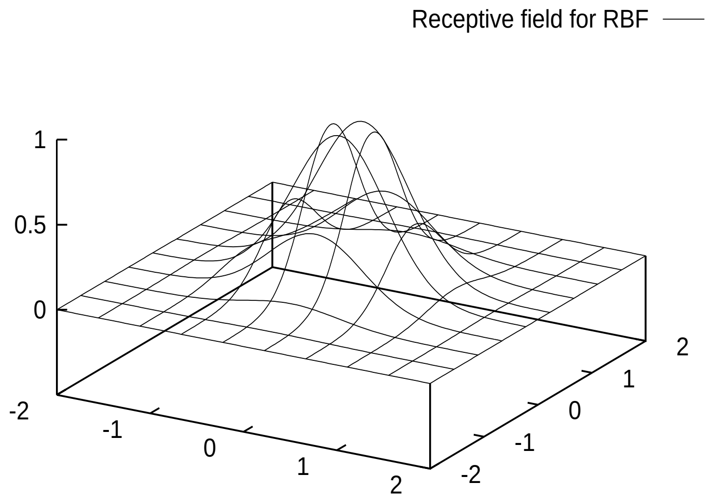


Figure 5.2: Combination of the Euclidean distance with the Gaussian activation function

Regardless of the procedure used to estimate the centers $\vec{c_j}$ in the hidden layer, the basis functions will always form a convex set that encapsulates the training data. The centers $\vec{c_j}$ are defined to be template-like representations of the training patterns (usually, for example, by the nearest-neighbour heuristic) and, therefore, any input which is significantly different from the training patterns receives a low output from the hidden units in the network. This is the basis of the argument to believe that RBF networks can present a very natural behaviour for rejecting patterns that differ significantly from the training classes, when the criterion of the confidence level for acceptance of a classification decision is imposed on the network.

Figure 5.3 illustrates the result of the visualisation procedure with an RBF network in the 2-class 2-dimensional problem. The decision surfaces defined by the network show, as originally expected, the generation of completely bounded regions surrounding the patterns from the training classes. The widths of the basis function are defined according to the nearest-neighbour rule and the classification decision taken by the network uses the confidence level (here defined as 0.65) for the acceptance/rejection of the input patterns.

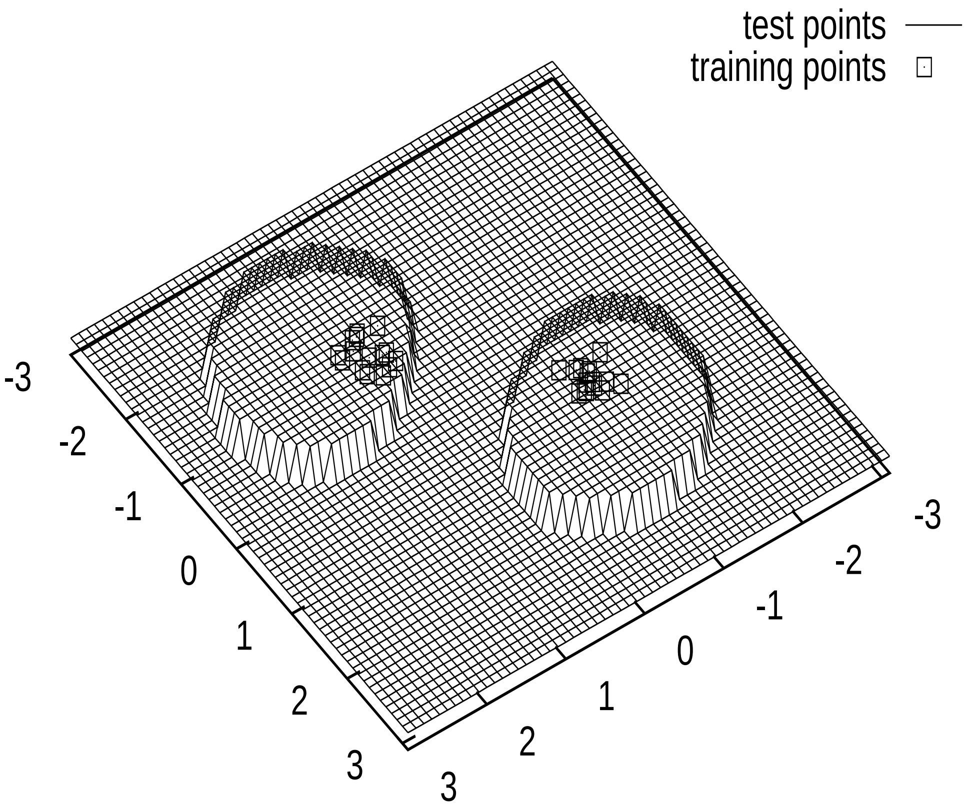


Figure 5.3: Decision regions for the RBF (2 classes)

In the 9-class problem the situation is not different and, as depicted in Figure 5.4, closed boundaries are again naturally created surrounding the training classes. It is clearly seen that the rejection areas are hugely increased in the case of the RBF, not only when compared to the standard MLP but also to the situations where the other MLP configurations considered in the previous chapters were employed.

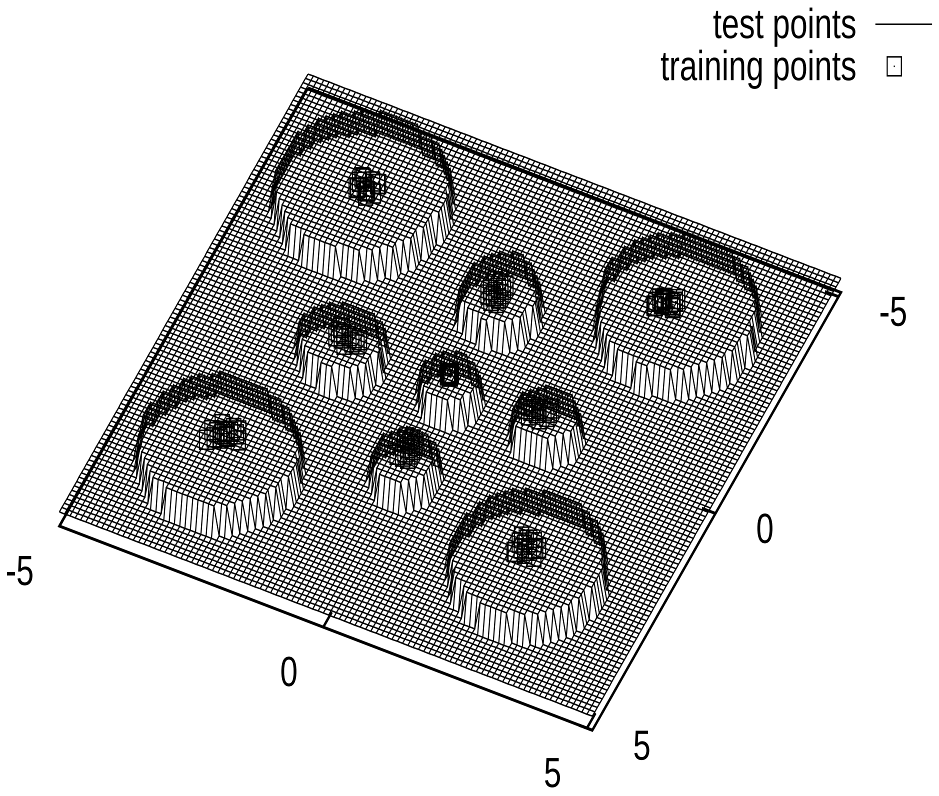


Figure 5.4: Decision regions for the RBF (9 classes)

5.6 Experimental Results

Practical experiments in the classification of alphanumeric characters have been carried out with RBF networks and a comparison with the standard MLP and with the most successful networks described in Chapter 4, the GMLP and the DMLP^t, is presented here. The strategy used to define the centers of the basis functions correspond to the nearest-neighbour heuristic procedure whereas the method employed for estimating the output layer weights is that of the pseudo-inverse matrix. The experiments were implemented in part using the Rochester simulator [24] and in part using the mathematical tool Matlab.

Although RBF networks with a different number of hidden units were initially attempted (100, 500 and 1000) only the networks with 1000 units were further explored. The reason for this is that this particular configuration was the only one to present results comparable to the other networks in terms of generalisation performance, and the number 1000 corresponds to the use of the same set of 100 patterns per class employed in the experiments with the other networks (100 patterns per class and 10 classes which gives a total of 1000 units to be defined in the hidden layer of the network)[1].

Table 5.1 presents the digit recognition and letter rejection results achieved by the network together with the previous results obtained with the MLP, GMLP and DMLP^t networks. It is observed that even for a small value of the confidence level (0.15) the rejection rate obtained with the RBF (44 %) is almost three times better than the equivalent with the standard MLP (17 %). For bigger values of the confidence level, the improvement in rejection performance is also very evident when compared to the MLP.

There is, however, a degradation in the recognition of valid digits critically proportional to the increase in the required confidence level. For example, it can be seen that with the use of the value of 0.55 for the confidence level, the digit recognition rate drops by up to 25 %. This is a result of the fact that many input patterns that receive low confidence outputs from the hidden units in the network are being rejected, in order to reduce the risk of a misclassification. The evidence for this argument can be illustrated by the results outlined in Table 5.2 which presents a comparison between the proportion of digits rejected by the network and the percentage of digits erroneously classified. This shows a decreasingly small error rate with the direct variation of the confidence level and, conversely,

[1]This confirms the initial expectation that usually many more units are necessary to be used in a RBF than in an MLP.

confidence	Digit Recognition Rate				Letter Rejection Rate			
	MLP	GMLP	DMLP^t	RBF	MLP	GMLP	DMLP^t	RBF
0	92.0	91.0	92.2	91.1	0	0	0	0
0.15	89.4	87.5	88.3	85.7	17	36	29	44
0.25	88.2	86.7	85.7	80.3	25	44	42	63
0.35	84.4	83.5	82.9	74.9	32	55	53	74
0.45	80.8	77.6	78.4	66.7	46	65	62	82
0.55	81.3	71.4	73.6	55.8	48	76	71	89

Table 5.1: Digit recognition rate vs letter rejection rate for RBF networks

an increase in the rate of valid patterns rejected by the network. For example, for a confidence level of 0.55 the error rate generated by the RBF network is around 0.2 % whereas for the standard MLP it is 3 %. On the other hand, the rate of digit rejection is around 44 % for the RBF and 15 % for the MLP network. However, it is important to note that if the detection of spurious inputs is one of the most important aspects to be considered in a given problem then the results obtained with the RBF network offer a better solution, even in terms of the trade-off between valid pattern recognition and spurious pattern rejection. For example, if it is required to establish a maximum error of around 3 % for the valid patterns, the RBF is able to recognise correctly 85.7 % of the digits with the small confidence level of 0.15 imposed on the network, as opposed to 81.3 % achieved with the MLP with the high value of 0.55 for the confidence level (see Table 5.1). In this case, the rejection of letters achieved with the MLP network is around 48 % and with the RBF network it is around 44 %. Another example can be given when a minimum recognition rate of 80 % is established. In this example, the rejection of letters achieved by the RBF is 63 % for a digit error rate of 1.4 % and the rate obtained with the MLP is 46 % for a digit error rate of 2.6 %. It is clear that a better balance between valid pattern acceptance and spurious inputs detection is obtained with the RBF network.

confidence	Digit Error Rate				Digit Rejection Rate			
	MLP	GMLP	$DMLP^t$	RBF	MLP	GMLP	$DMLP^t$	RBF
0	8.0	9.0	7.9	8.8	0	0	0	0
0.15	5.3	3.8	5.1	3.4	5.3	8.7	6.6	10.9
0.25	5.3	2.9	3.4	1.4	6.5	10.4	10.9	18.3
0.35	3.7	2.0	2.2	0.5	11.9	14.5	14.9	24.6
0.45	2.6	1.4	1.6	0.4	16.6	21.0	20.0	32.9
0.55	3.0	0.9	1.1	0.2	15.7	27.7	25.3	44.0

Table 5.2: Digit error rate vs digit rejection rate for RBF networks

Comparing the results obtained with the $DMLP^t$ and, especially, the GMLP network presented in Chapter 4, the improvement shown with RBF networks in terms of letter rejection is not as marked as that observed when compared to the standard MLP. The increase in the letter rejection rate at fixed confidence levels is by about 15 % on average with respect to the GMLP and by 19 % when compared to the $DMLP^t$. However, the same comparisons made above with the MLP when applied to these networks show that, in various cases, the compromise between valid pattern recognition and invalid pattern rejection offered by them can be similar or, in some cases even superior, to that attained with the RBF network.

Consider, for example, the following correlation between the results obtained with RBFs and the results obtained with GMLPs. For an error rate of 3.4 % observed with the RBF network, 85.7 % of the digits are correctly classified with 44 % of letters rejected by the network. An even smaller error rate of 2.9 %, obtained with the GMLP, provides 86.7 % of digit recognition and the same 44 % of letter rejection rate. In another case, for a digit recognition rate of 85.7 % with the RBF, the digit error rate is 3.4 % and the letter rejection rate is about

44 %. For a recognition rate of 83.5 % obtained with the GMLP and a digit error rate of 2.0 %, the rejection of letters is 55 %.

A similar level of evaluation can be made with the DMLP^t network although, in this case, its performance is slightly inferior to that achieved by the GMLP network. For example, for a digit recognition rate of 85.7 % with the RBF, 44 % of letters are rejected and the digit recognition rate is of 3.4 %. Almost exactly the same rates of 85.7 %, 42 % and 3.4 are obtained with the DMLP^t with the confidence level set to 0.25. In another situation, while the RBF network provides 80.3 % of digit recognition, 63 % of letter rejection and 1.4 % of digit error rate, the DMLP^t provides similar rates of 78.4 % of digit recognition, 62 % of letter rejection and 1.6 % of digit error rate.

The results illustrated here indeed indicate that RBF networks are strong candidates for the reliable rejection of spurious patterns. However, the results also indicate that GMLPs and DMLP^ts can, in various situations, represent a better alternative in terms of the trade-off between valid pattern recognition and spurious pattern rejection.

One very important aspect to note concerns the relative computational and implementational complexities associated with RBFs and with the GMLP and the DMLP^t. Although RBF networks can be trained very fast, they require a much larger number of processing units than the GMLP and the DMLP^t to provide a similar classification performance. The results reported above has shown situations of similar classification rates obtained with the RBF and the DMLP^t with the important difference that the DMLP^t used only the same number of 80 hidden units employed originally in the MLP, whereas the RBF network was composed of 1000 units. With respect to the GMLP, the comparison is even more favourable to this latter network since the same number 80 of units provided a better overall classification performance in various situations.

5.7 Addressing a Potential Problem

This section examines a specific situation which, in the event of occurring, can affect the rejection capabilities of RBF networks. A simple example network is used to illustrate this situation and possible solutions to the problem are considered.

It has been seen in this chapter that the hidden units of an RBF network provide a way of constructing a convex set that fits the training data in a way that any point falling outside the limit of the closed boundaries tend to receive low outputs from the units. It was also seen that this is precisely the reason why RBFs present an inherently strong rejection capability.

However, since the process of network training uses only the outputs of the hidden units with respect to the training patterns to define the representations stored in the output layer weights and hyperplanes are used at the level of the output layer to partition the pattern space, it is possible that the situation where all the hidden units provide low outputs (a situation which does not happen during the training process and does not represent any of the training classes) is mapped exactly to the set of responses given by the network to one of the trained classes. If this happens, any input presented to the network that receives low outputs from the hidden units will be accepted if as belonging to the referred class.

Figure 5.5 shows the example of a network in a 2-class problem, carefully designed to exhibit such behaviour. Two hidden units are used to represent two cluster centers ($c_{11} = -0.99, c_{12} = 0.99$) and ($c_{21} = 0.67, c_{22} = -0.65$), and the output layer weights w_{ji} and biases w_{j0} are defined in a way that the decision regions are bounded for the output unit representing class 2 ($w_{21} = -3.63, w_{22} = 3.57, w_{20} = -0.96$), but are open for the unit representing class 1 ($w_{11} = 3.85, w_{12} = -3.96, w_{10} = 1.13$). In the latter case, the situation of low hidden unit outputs are mapped as belonging to this second class. The calculation of class 2's output unit activation when the outputs of the hidden units are set to *zero* is the result of the inner product ($net_2 = -3.63 \times 0 \; + \; 3.57 \times 0 \; + \; (-0.96)$) applied as input to the sigmoid function $a_2 = 1/(1 \; + \; exp(0.96))$ which provides $a_2 = 0.27$. This is a level of activation considered to be small and the imposition of the confidence level on the network can easily reject the input associated with it. However, the calculation of class 1's output unit activation, the inner product ($net_1 = 3.85 \times 0 \; + \; (-3.96) \times 0 \; + \; 1.13$) given as input to the sigmoid $a_1 = 1/(1 \; + \; exp(-1.13))$ provides $a_1 = 0.76$. This output can be considered as high as any other response given to a slightly distorted valid pattern presented to the network.

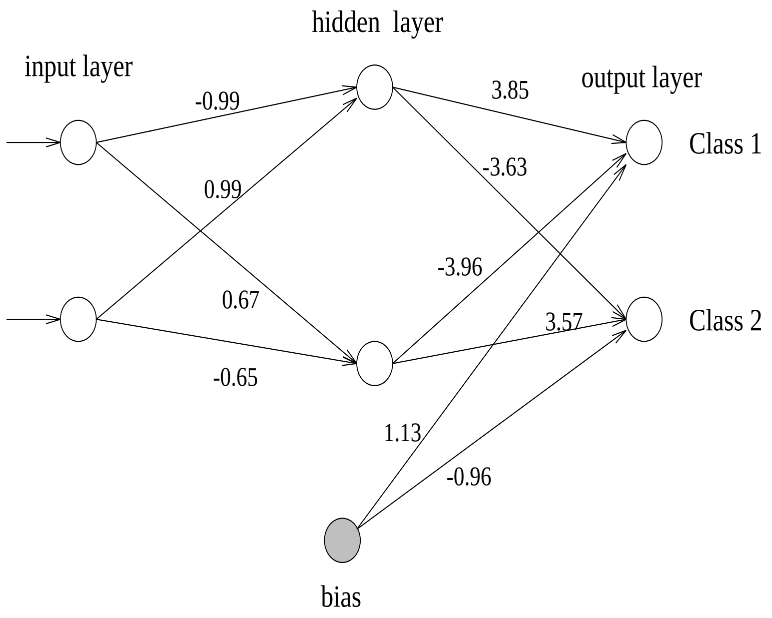


Figure 5.5: An example of RBF network

The actual implementation of the above network and the test of its response to a large matrix of points covering the input space results in the diagram illustrated in Figure 5.6. This experiment illustrates the occurrence of the problem. For class 2, only the points in the area surrounding the training patterns of this class are classified as belonging to it. However, any other input that is not classified to belong to class 2 is accepted as belonging to class 1.

Although this undesirable situation can be considered unlikely to happen in practice (for example, in all the experiments reported with RBF networks this was never observed), it is important that the problem is understood and that possible mechanisms for dealing with it are considered, and used if necessary.

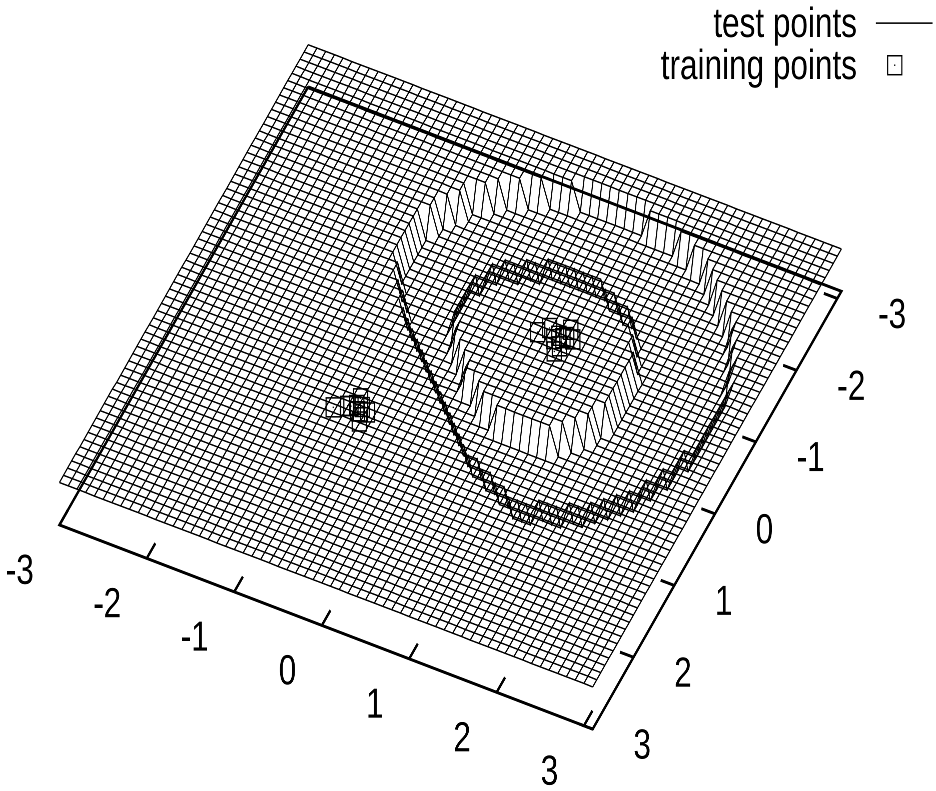


Figure 5.6: Decision regions for the RBF network of Figure 5.5

There are different ways of handling this problem. One straightforward idea can be the further test of the network after it has been trained on a particular task, where the output unit responses are checked for the case of low hidden unit outputs. If all the output nodes are providing low outputs to low hidden unit activations then nothing needs to be done. Any pattern very different from the training classes should be rejected by the criterion of the confidence level. If, however, one of the trained classes is detected to be responding positively to low hidden unit outputs then a minimum level of hidden unit activation can be established as a condition to accept the input pattern, based on activations previously obtained for patterns from the training classes. A validation set can be used to estimate the threshold for the level of activation below which any input would be rejected by the system. The level of activation would correspond to the linear combination of the hidden layer outputs $\phi_j(|\vec{x_p} - \vec{c_j}|)$ according to weights λ_j. Since no assumption is made about the kind of distribution that the training data has, the values of λ_j can be defined considering that they contribute equally to measuring the level of activation in the hidden layer. Another possible procedure is to choose the values of λ_j based on the log likelihood of the training data with respect to the variable A_p [6]. Hence, the level of activation for a pattern $\vec{x_p}$ can be defined as :

$$A_p = \sum_{j=1}^{k} \lambda_j \cdot \phi_k(|\vec{x_p} - \vec{c_j}|)$$

where k is the number of hidden units, $\sum_{j=1}^{k} \lambda_j = 1$ and $0 \leq \lambda_j \leq 1$. Since all λ_j's are defined to be equal each $\lambda_j = \frac{1}{k}$.

The presentation of the validation set is then used to estimate the threshold value for A_p. For example, the values of A_p are calculated and the threshold is set so that most of the patterns in the validation set are classified as valid by the network. Any input that does not produce a level of activation above the threshold can be readily rejected by the system. This would be seen as an additional mechanism integrated with the network which examines the responses of the hidden layer before taking a final classification decision.

Another possible strategy can be the inclusion of an additional node in the output layer of the network to represent the class of spurious patterns, and the previous training of the network with this further class. This strategy is similar to that of negative training described in Chapter 2 with the very important difference that there is no need for the presentation of a massive number of random patterns as members of the spurious class. Instead, the unit representing this class has to account only for the case where all the hidden units provide low outputs. The training of the additional class is carried out by the presentation of low hidden outputs and the explicit definition of the hidden to output layer weights, leading to the unit representing the spurious class, so as to recognise those patterns as genuine members of the spurious class.

All the procedures applicable originally (least-mean squares, pseudo-inverse matrix, Newton-like method) can still be used with this modification. In order to maximise the spurious pattern rejection rate, it is necessary to define for each application appropriate values for the hidden unit outputs representing the spurious class. This can be done by considering, for example, the level of degradation in the network's recognition performance with respect to the training classes measured at each different trial. The hidden outputs representing the spurious class can be initiated with a close to zero response and can be further elevated gradually until they reach a point just before the situation where the degradation in recognition performance for the application starts to be considered substantial. This would represent the boundary on the activations of the hidden units for which a pattern should be classified as genuine or not.

The definition of the hidden to output layer weights through a linear optimisation technique (like the pseudo-inverse matrix) defines a boundary line to separate the spurious class from the valid classes. Since the higher the activation of the hidden layer units the more likely it is that the input pattern belongs to the training classes and the lower the activations are the more likely the pattern is to belong to the spurious pattern class, the decision surface that results from the training process should be such that any input that receives hidden unit outputs below the thresholds should tend to be classified as more similar to the spurious class rather than as to the valid classes.

5.8 Conclusion

This chapter has investigated the model of radial basis function with respect to the rejection problem and an experimental comparison between RBFs and some of the other networks described in the thesis was presented. In general, it can be said that networks in which the processing units compute a localised function of the input space should naturally provide more reliable structures for rejecting spurious inputs. One such class of networks correspond to the radial basis function architecture. In this case, because each processing unit in the network creates a localised receptive field in the input space, there is a very low probability of spurious patterns being accepted by the network.

Indeed, the results shown in this chapter confirm that RBF networks are powerful candidates for applications that demand high reliability when one of the primary concerns is the correct identification of spurious input patterns. However, an example illustrating the practical application of the classification of handwritten characters has shown that some "intermediate" network configurations between MLPs and RBF models such as the DMLP, and especially the GMLP, may offer a better compromise between the recognition of valid patterns and rejection of spurious patterns. The other advantageous feature of the GMLP and the DMLP when compared to RBFs is that they can improve the rejection abilities over the MLP network with a much smaller number of free parameters required in the network. Whereas the RBF network used 1000 hidden units for achieving the results reported in this chapter, the GMLP and the DMLP^t each employed only 80 units as used in the standard MLP network. On the other hand, RBF networks offer the benefits of a very fast training process.

It is clear, therefore, that the choice of which network to use in a practical application has to take into account the particular parameters demanded in the specific problem. This chapter has explored and identified some of these parameters of choice.

Chapter 6

A Bootstrap-like Rejection Mechanism

6.1 Introduction

All the approaches described in this thesis so far refer to solutions where the problem of the detection of spurious inputs is dealt with during the process of network training. In this chapter the subject is investigated, in contrast, from a very different perspective.

A mechanism is proposed based on the ideas of bootstrapping [46] that can be incorporated into the standard MLP architecture, as a back-end mechanism, with the objective of providing the network with the ability to continuously modifying its responses across the input space during its usage phase. The mechanism makes use of the outputs provided by the network itself during its recall phase, in order to improve performance in the rejection of spurious inputs, through a scheme of reinforcement of the classification decisions taken by the network.

The *bootstrap* algorithm is a powerful mechanism that provides a way of adapting the weights of the processing units in a neural network when the desired output for each training input pattern is not known in an application environment. The concept of the bootstrap has a long history, going back to the work of Lucky [46] in 1966 and Widrow [85] in 1973 and it is one of the basic principles of the classical paradigm of reinforcement learning [3, 85]. The algorithm has been successfully applied in practical applications [86] and, more recently, Hinton & Nowlan have demonstrated how it can be viewed as an unsupervised clustering procedure [30].

In this chapter, it is shown how an adaptation mechanism based on the concept of bootstrapping can be incorporated into an MLP network to improve its knowledge of the pattern space in classification problems. It is shown how the decision regions generated by the network pre-trained on a set of classes through supervised learning can be gradually modified, during the network's recall phase, to improve its ability to detect spurious patterns. The procedure makes use of the classification decision taken by the network itself as a bootstrap signal for readapting the weights of the network without the use of any supervised training.

An example of the operation of the mechanism is given in the practical application of the classification of handwritten characters and some practical consideration is given for its use in a real-world environment. It is also shown how the performance of the different MLP network configurations considered in the previous chapters, such as the GMLP and the DMLP networks, can be further enhanced with respect to the detection of spurious patterns through their integration with the bootstrap procedure.

6.2 The Bootstrapping Algorithm

The idea of the bootstrap algorithm was conceived taking into consideration situations where an adaptive element (a linear unit) operating in a particular pattern environment is presented with a series of inputs for which there is no a priori information about what the desired output should be for each given input pattern. In such situations, one approach is to consider the actual output of the linear unit itself thresholded at zero as the desired output, assuming that the decision taken by the unit is in fact correct. Figure 6.1 illustrates the general form of operation of the mechanism (redrawn from [85]). When the output of the unit is above zero, +1 is taken as the desired output (d) and the weights of the unit are adapted accordingly. A similar action takes place in the case of an actual output below zero but, in this case, the desired output becomes -1. This form of bootstrap is referred to as *positive bootstrap or learning by reward*. An alternative way of defining the desired outputs is by inverting the signal of the thresholded output and training the unit to modify its response in a direction away from the actual output. In this case, the bootstrap is known as *negative bootstrap or learning by punishment*.

In typical situations of the application of reinforcement learning, an external element, a *critic*, is responsible for deciding what form of adaptation should be carried out, whether reward or punishment, based on a series of observations of the behaviour of the system. If successes are achieved as a result of a sequence of decisions taken by the system, positive bootstrapping takes place; otherwise, the system is punished for its failures.

In the next section, it is shown how the principle of bootstrapping can be used to modify the outputs of an MLP network to the input pattern space, using as a basis for re-adaptation the classification decisions taken by the network itself.

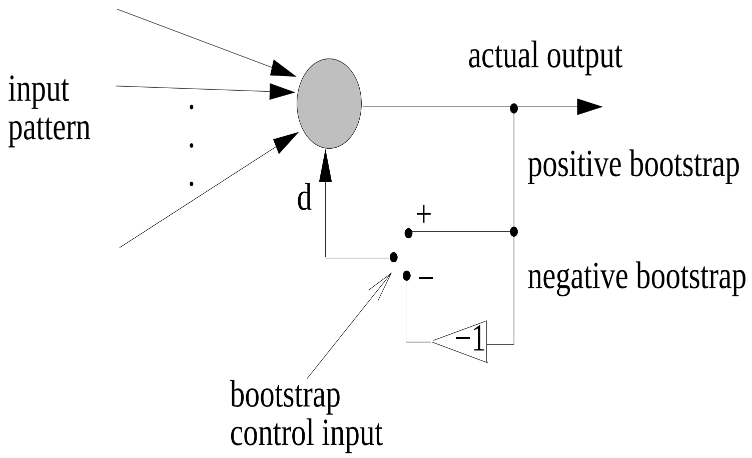


Figure 6.1: Bootstrap adaptation

6.3 A Bootstrap-like Rejection Mechanism

6.3.1 Combining negative training with bootstrapping

In chapter 2, it was described how one of the first techniques proposed to overcome the problem of the rejection of spurious patterns was based on the idea of previously training the network with random "negative" examples. Unfortunately, it was also demonstrated that the solution obtained with this method does not guarantee that the decision regions generated by the network to separate unknown regions of the pattern space from the training classes are uniform. This problem becomes especially critical in situations involving higher dimensions, as is frequently the case in many real world applications.

However, the original concept of using negative training to provide a network with information about rejection areas of the input space can be further considered in the recall phase of the network. Since, in the network's training phase, there is little (or no) a priori information about the kinds of patterns that should be rejected in a certain environment, one possibility is to use the classification decision taken by the network itself in the recall phase to dynamically change its knowledge about the structure of the pattern space. The technique of bootstrapping is a method that allows such a form of continuous re-adaptation to the properties of the current pattern environment.

The scheme for a bootstrap-like rejection procedure can be described as follows : when a given pattern is rejected by the network as a result of "considerable doubt" about its identity, it is "assumed" that the decision taken by the network was in fact correct and the entire network is re-trained to reinforce its decision. The whole idea is based on the assumption that if none of the output units is responding positively to the input pattern with a minimum level of activation then it is very reasonable to assume that the input should correspond to an invalid pattern and, accordingly, this recent acquired information can now be used to enhance the network's rejection response to the current pattern. The network enters a process of re-learning by applying the same gradient descent process used in the training phase making use of a new training set created on-line, composed of the rejected pattern and a sample of other patterns from the original training set, included in order to avoid the problem of catastrophic forgetting of their previous learning [23, 50]. This modification in the MLP network operation results in the architecture shown in Figure 6.2.

When a pattern is presented to the network, the responses of the output units are evaluated and compared against each other to determine if the output of the winning unit exceeds those of the others by more than a given *confidence level*. If so, the pattern is accepted as belonging to the corresponding trained class. So far, this description of network behaviour is exactly the same as that presented previously. A different operation takes place, however, when a rejection decision occurs. In this situation, another component of the system, a bootstrap control mechanism (see Figure 6.2) is responsible for checking the degree of "uncertainty" about the identity of the input pattern present in the rejection response of the network. In a typical classification decision, it is observed that a rejection decision occurs either when the responses of all the output units are very low (close to '0') or

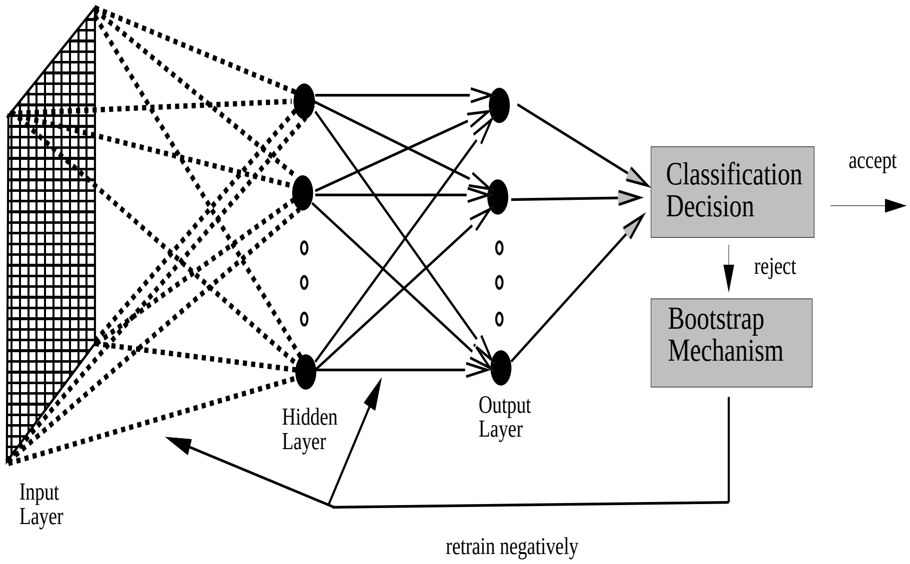


Figure 6.2: Multilayer perceptron with bootstrapping

some (or most) of the outputs are high (close to '1'). In either case, if the rejection decision is taken because the difference between the largest output and the second largest output did not reach a minimum defined *certainty level* then it is assumed that the input pattern is very different from the training classes and can be used to re-train the network negatively to reinforce its rejection (with the vector of desired outputs defined as [0,0,...,0]). The certainty level implements, therefore, the threshold for the state of doubt from the network. If one of the output units reaches a large enough level of activation to exceed those of the others by this threshold then the rejected pattern is not considered for negative re-training. Otherwise, there is "considerable doubt" about the input pattern membership, enough to consider it as an invalid pattern and this is used to reinforce the rejection decision.

The confidence level, as described in all the approaches in the previous chapters, can be varied to make the acceptance of an input more or less rigorous according to the needs of the particular problem, whereas the certainty level should be small in order to reduce the risk of valid patterns being re-trained negatively. It is important to note that the use of a sample set of patterns from the original valid training set is absolutely crucial; otherwise, the network will tend to forget the previous input/output associations learned in the training phase.

The idea behind this approach is based on the expectation that other patterns similar to the current rejected pattern, which were still being erroneously classified, also come to be rejected as a result of the modification in the decision regions generated by the network. Consider the simple 2-class classification problem illustrated in the example of Figure 6.3. Figure 6.3(a) shows a pattern (denoted R) which is an example of a pattern normally rejected by the network (it falls in the rejection region represented by the shaded area in the diagram). Similarly, another pattern (designated S) represents a pattern considered to be similar to R which is still erroneously classified as belonging to class X. In Figure 6.3(b), a hypothetical result of the network re-training to reject pattern R is depicted, where the classification boundaries are altered in such a way that pattern S now becomes a rejected pattern.

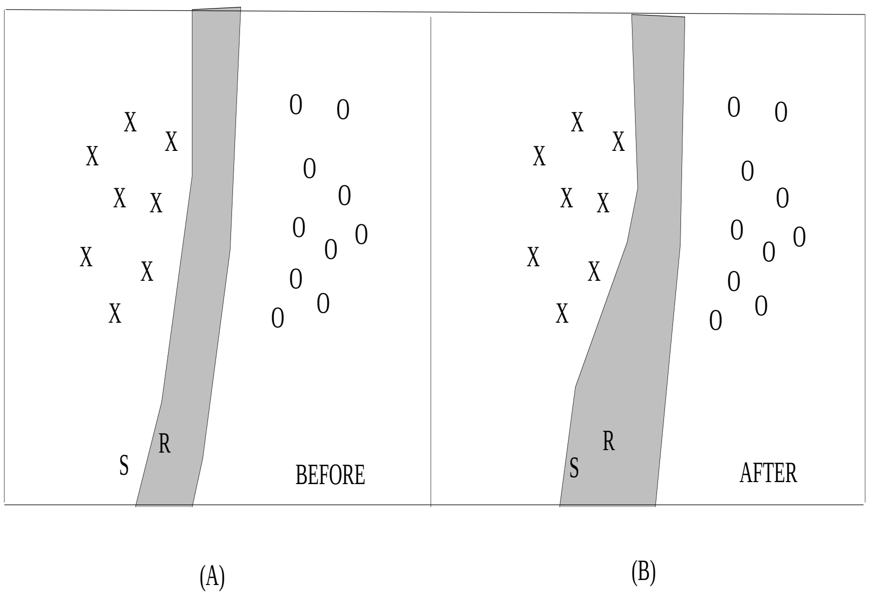


Figure 6.3: Input space separation for the MLP with bootstrapping

The mechanism of reinforcement of a network's response described here can be seen as an optimisation process where the network's parameters are adjusted to include the new information obtained with a rejection decision. The problem at hand is different from that encountered in a situation employing the classical paradigm of reinforcement learning in the sense that there is no "critic" involved in the process of determining the desired output of the network based on the successes or failures of the system.

In the context of the application of the bootstrap rejection mechanism, there is no specific concept of success or failure because there is no way of knowing what the correct output should be for any given input pattern. It is considered that the decision taken by the network is correct by "assuming" that the network has been properly trained previously during the training phase and a control signal is issued for readapting the network such as in the case of a positive bootstrap adaptation (reward) [85]. If the input pattern is accepted as belonging to one of the training classes then no process of retraining takes place[1].

It is also useful to note that the rejection mechanism differs from the original concept of the bootstrapping algorithm in the sense that the actual output of an individual processing unit is not thresholded at zero to determine the desired output for that unit. Instead, the rejection mechanism determines the desired output of the entire network through what can be called a transition from a state of "doubt" about the identity of the input pattern (the original rejection decision) to a state of "certainty" that the input pattern is indeed a spurious pattern and should, therefore, be rejected (represented by the reinforcement of its rejection decision by defining the desired responses of all the outputs units to be [0,0,. . .,0]). However, as with the original procedure, the basic principle of reinforcement of a response is still the active ingredient of the system.

[1] A similar reinforcement strategy can be thought of when an input pattern is confidently accepted by the network, but the effectiveness of this further modification is not examined here.

6.4 Visualisation of the Decision Regions

The experimental visualisation procedure which considers the classification problem in a 2-dimensional input space is applied again in this section to illustrate how the mechanism of bootstrapping can dynamically modify the decision surfaces generated by the network.

An MLP network (with one or more hidden layers) is trained through back-propagation learning with a sample of patterns representative of the training classes, and a test input matrix large enough to encompass an extensive area of the pattern space surrounding the trained classes is presented to the network. The responses of the output units for each point in the input matrix are compared and a pattern is classified as belonging to one of the training classes if the output (in the range [0,1]) for that class exceeds those of the others by a given confidence level; otherwise, the pattern is rejected.

The experiments are carried out with a very high value of the confidence level (0.65) and the results are shown in Figure 6.4 for the case of a 2-class problem, where the elevated surfaces correspond to the points confidently accepted by the network while the rest of the points represent the rejected patterns. Figure 6.4(a) corresponds to the simulation of the standard MLP where the bootstrapping mechanism is not yet integrated with the network. As previously illustrated in chapter 2, almost all the points in the input matrix are either classified as belonging to one of the classes or to the other, and even those most distant from the training patterns are still accepted by the MLP network. Only the points falling in between the training data points are rejected.

In Figures 6.4(b),6.4(c) and 6.4(d), the effect of applying the bootstrap rejection mechanism in the 2-D problem is illustrated. These show the gradual modification in the shape of the decision regions after 1, 3 and 5 test epochs of presentation, respectively, of the whole set of matrix points covering the input space. As can be seen, the weights of the network are modified towards a more reliable structure for detecting spurious patterns through the creation of boundaries surrounding the training data; and, more importantly, these modifications take place without the action of an external teacher for dictating the desired output for each input pattern.

Now, in addition to the original constraint imposed for the separation of the trained classes, the process of bootstrapping slowly displaces the hyperplanes towards a distribution that limits the areas of the pattern space considered to belong to each class, as would be most appropriate for the purposes of spurious pattern rejection.

A repetition of the above experiment now on the classification problem involving 9 classes ratifies the initial observations. Figures 6.5(a),6.5(b) and 6.5(c) show the evolution of the decision surfaces with the bootstrap mechanism, where, again, a gradual transformation of the class boundaries is seen from open regions to closed regions.

It is important to emphasise that the main objective of this visualisation procedure is to demonstrate that the inclusion of the bootstrap mechanism makes possible the alteration of the responses of a pre-trained network as a result of the occurrence of rejection decisions taken by the network itself during its usage phase. It is clear, however, that in real environments it is not possible to have the presentation of a whole matrix of points covering the pattern space so that the original decision boundaries can be modified. On the other hand, the experiments reported are very useful in understanding the beneficial gradual transformations (in terms of spurious pattern rejection) of the decision regions which can result from the application of the mechanism.

Further simulations were used to observe the behaviour of the network in a slightly more complex situation in the 2-class problem where the data consists of

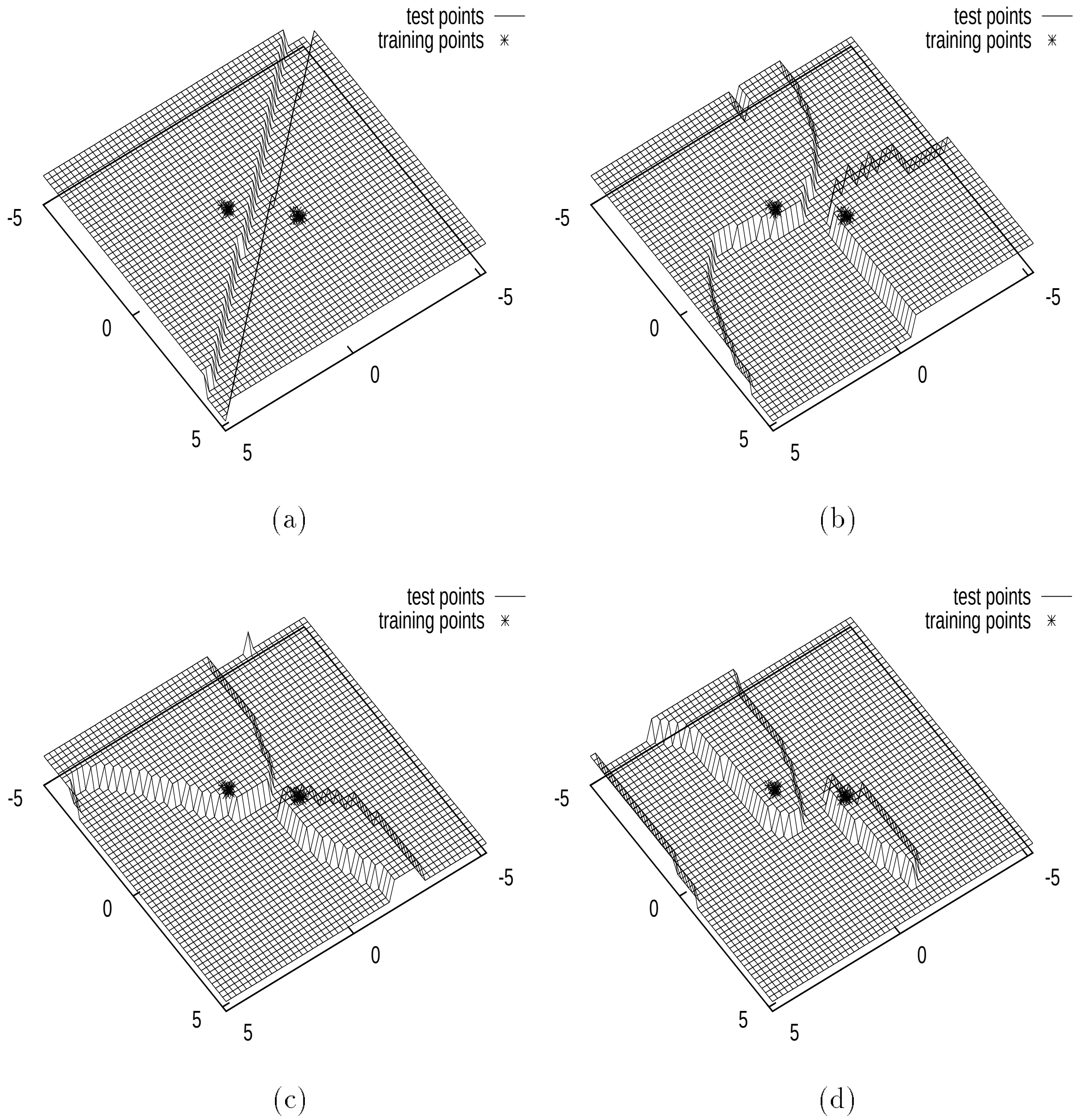
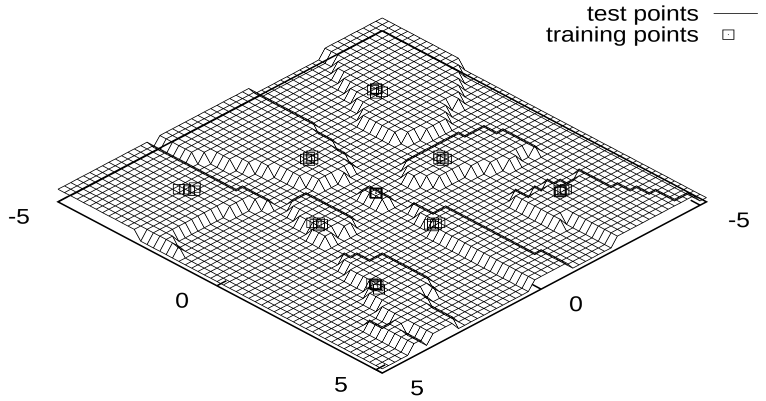


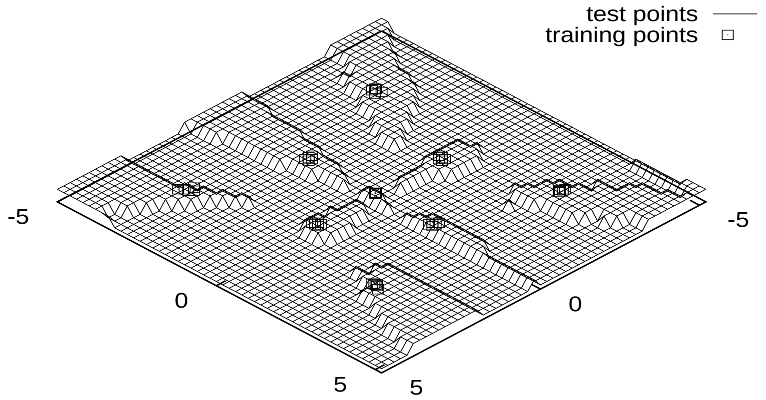
Figure 6.4: 2-class problem (a) standard MLP, 1 test epoch; MLP with the bootstrap (b) 1 test epoch, (c) 3 test epochs, (d) 5 test epochs.

clusters of larger sizes that form a 90 degree-like shape between them (in contrast with the simple clusters seen in Figure 6.4). Various trials were attempted and the results of two of these are presented in Figures 6.6 and 6.7. In Figure 6.6, the diagram shows again an evolution from completely open decision regions to closed regions after some cycles of the presentation of the input matrix. This result ratifies the previous simulation with the simple clusters. An interesting result, however, is in the case of Figure 6.7 where closed decision regions are not entirely obtained. As can be seen in this case, the generation of closed regions is observed for only one of the cluster class whereas for the second class the classification regions is practically unchanged from the original configuration for the MLP without bootstrapping. It is suspected that this particular problem is caused by the scanned presentation to the network of the test points in the input matrix, where all points are fed sequentially in a given pre-defined order (for example, from left to right, top to bottom). This is an artificial situation which biases the development of new hyperplanes towards the start of the scanning process and thus creates rejectable points correlated in one region in advance of rejectable regions in other regions. In the example shown this appears to have created a region, on the right of the plane, in which improvement in rejection does not occur.

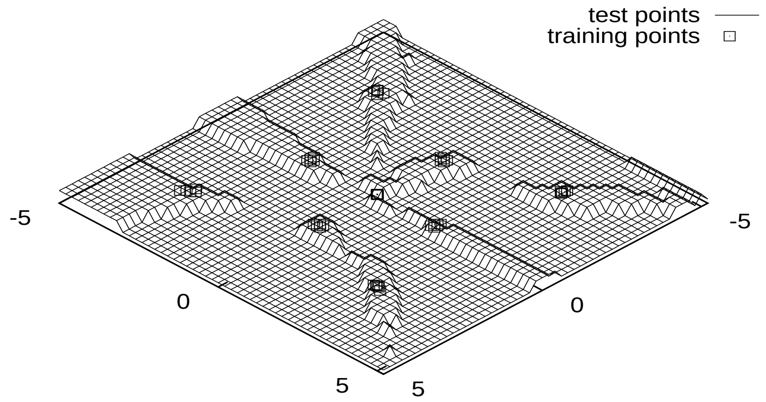
While the overall result is still clearly an improvement in terms of of rejection performance, the network has settled on a sub-optimal solution with respect to an ideal closed configuration for the decision regions. If, as conjectured, this is due to the artificial sequence of scans then in a real environment where rejectable events are more likely to occur in a arbitrary manner, then a more acceptable arrangement of hyperplanes is likely to result. It is clear, however, that the performance of the bootstrap method is problem dependent being influenced by the distributions of the training data and by the particular occurrence of rejectable points in each situation.



(a)

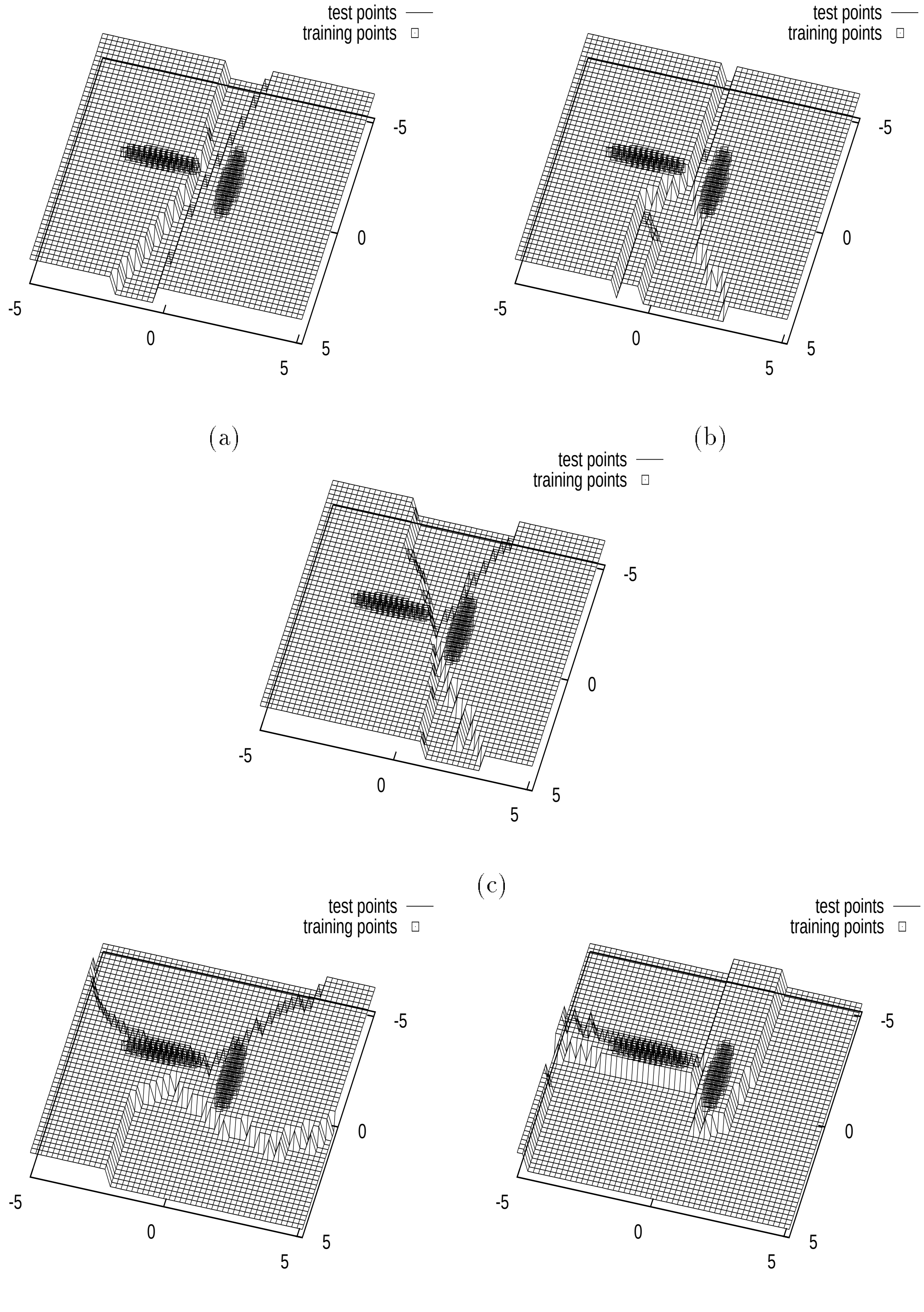


(b)



(c)

Figure 6.5: 9-class problem, MLP with the bootstrap (a) 1 test epoch, (b) 3 test epochs, (c) 5 test epochs.



(a) (b) (c) (d) (e)

Figure 6.6: 2-class problem with larger clusters (a) standard MLP, 1 test epoch; MLP with the bootstrap (b) 1 test epoch, (c) 3 test epochs, (d) 5 test epochs, (e) 7 test epochs.

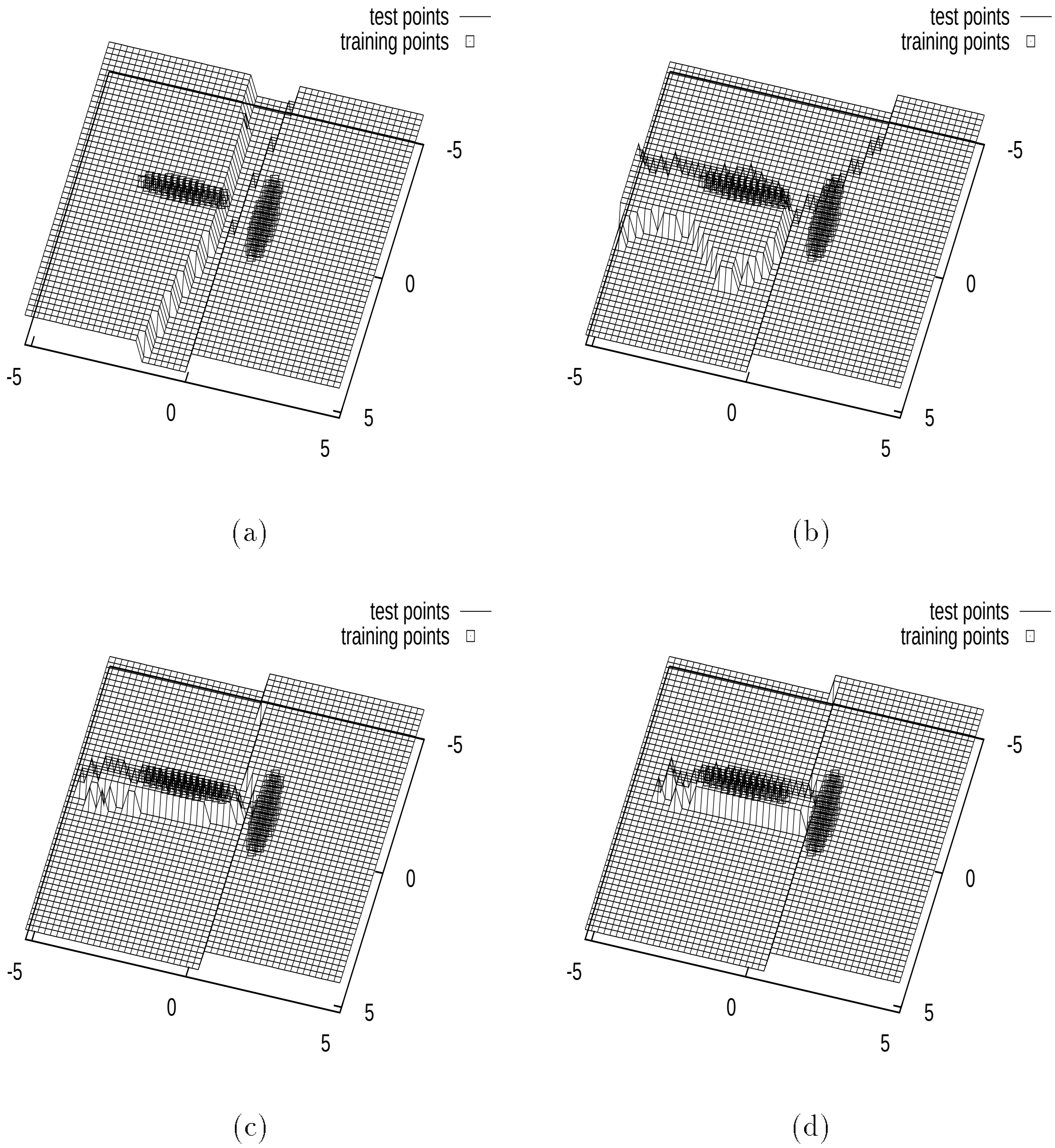


Figure 6.7: Another trial for the 2-class problem with larger clusters (a) standard MLP, 1 test epoch; MLP with the bootstrap (b) 1 test epoch, (c) 3 test epochs, (d) 5 test epochs.

The evidence to back up the argument given above is the fact that simulations with different order of presentations of the input matrix provide different solutions. For example, in Figure 6.6 the simulation corresponds to a different presentation for the input points from that of Figure 6.7 and in this case the resulting decision regions are closed. A last experiment carried out in fact confirms such observations. In Figure 6.8 the input points are selected from the input matrix and presented to the network in a completely random order. The result of several trials have shown that in every case the evolution of the classification regions tend to be such as the one shown in Figure 6.8, with a much more satisfactory arrangement of the decision boundaries in terms of rejection performance. It is suggested for a further work a fuller investigation of how the bootstrap mechanism behaves in many different complex arrangements of data clusters such as, for example, in the known 2 spirals problem.

With the aim of investigating how the rejection mechanism would scale up to real-world problems and in order to establish a relation between this method with the other approaches described in the previous chapters, the next section considers its operation in the practical application of the classification of handwritten characters.

6.5 Experimental Results

Extensive experiments have been carried out to investigate the performance of the proposed technique in the handwritten character recognition application employed in the previous chapters. The database of digits used is, as in the previous experiments, divided into two subsets composed of 1000 and 2000 characters, used respectively to train the network and to test its recognition performance. The set of letters used as a model of spurious patterns consists of 2600 patterns with 100 patterns per class. The MLP networks composed of 80 units are run several

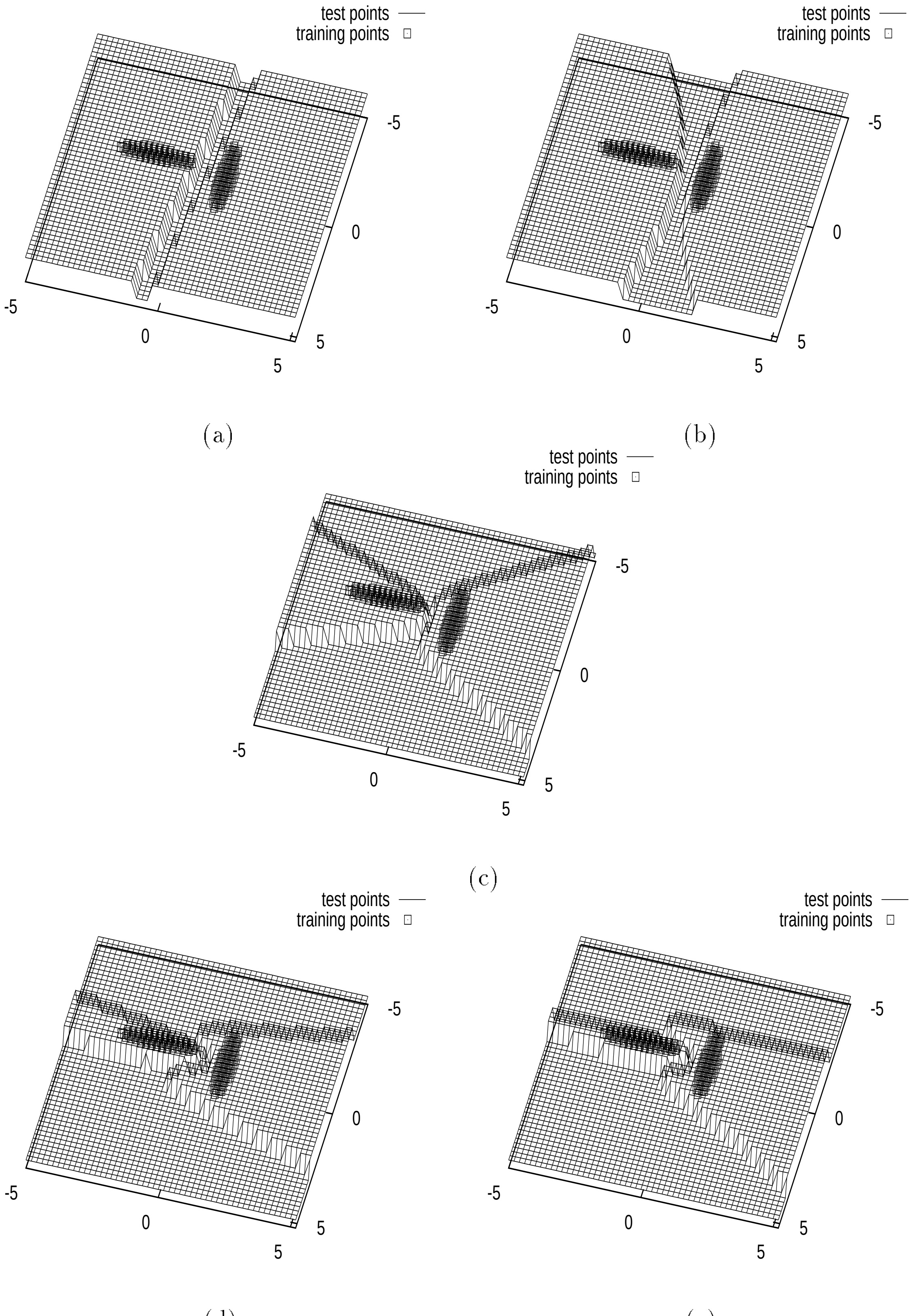


Figure 6.8: 2-class problem with larger clusters, random presentation of points (a) standard MLP, 1 test epoch; MLP with the bootstrap (b) 1 test epoch, (c) 3 test epochs, (d) 5 test epochs, (e) 7 test epochs.

times and the results are measured for the different values of the confidence level (from 0.15 to 0.65). In accordance with the explanation given in section 6.3, the certainty level is chosen to be small (0.15).

The experiments can be divided into two parts. The first part consists of presenting the whole test set of digits and letters to the network and of measuring the recognition and rejection rates both when there is no bootstrapping mechanism operating and when the mechanism is present. Furthermore, the results can illustrate the enhancement in performance that can be obtained with respect to a test set of completely novel patterns, never seen before by the network. The second part consists of, again, presenting the character database to the network but this time alternating the entire set of digits and letters several times in order to measure the evolution of the classification rates in the presence of bootstrapping. Although this second procedure cannot be considered completely realistic in quantifying any improvement in performance in a real environment, due to the fact that there is no repetition of patterns during the recall phase, it provides a convenient way of qualitatively evaluating any possible change in performance, particularly in terms of degradation in the recognition rate for valid patterns as the system evolves to a more reliable structure for rejecting spurious inputs.

Table 6.1 outlines the rejection results for the complete set of experiments. The results obtained may first be compared in the absence of the bootstrap (epoch 0 in Table 6.1) [2] with those obtained when the mechanism is operating (epoch 1 in Table 6.1). It is observed that with the application of this technique the rejection rates for the letter test set are substantially increased, especially for smaller values of the confidence level. For example, for a confidence level of 0.15, the rejection rate changes from 17 % to 32 % (an increase by a margin of 15 points over the original rate). For the other values of the confidence level, there is an improvement by 9.5 % on average. With respect to the correct recognition of valid patterns

[2] In this sense, an epoch is defined as the presentation of the whole set of test patterns in the recall phase of the networks; there is no supervised training involved in this process.

epochs	Confidence Level					
	0.15	0.25	0.35	0.45	0.55	0.65
0	17	25	32	46	48	64
1	32	39	45	53	56	67
$\Delta 1 =$	+15	+14	+13	+7	+8	+3
3	53	52	61	63	66	67
6	61	64	67	70	73	76
9	66	68	70	74	76	76
12	68	69	71	76	77	79
15	68	70	72	75	78	79
$\Delta 15 =$	+51	+45	+40	+29	+30	+15

Table 6.1: Letter rejection rates for the MLP with bootstrapping

(digits) there is no degradation in performance as illustrated in Table 6.2. In reality, in this case, some improvement in the recognition rates is noticed. Results are also measured in terms of error rate in the recognition of digits, which are presented in Table 6.3. Although not significantly apparent at this stage, these results show a tendency for a reduction in the rate of misclassified patterns with the employment of the bootstrap.

With respect to the second set of experiments, the network is run through a further 14 test epochs in the recall phase (represented by the epochs 3 to 15 in Table 6.1). It is then possible to observe a gradual increase in the letter rejection rate, epoch after epoch, with a further increase to 68 % after the 15^{th} epoch, for the value of 0.15 in the confidence level. With respect to the other values an improvement by over 40 % is obtained after the 15^{th} epoch. Results measured in terms of the recognition rate of the digits (see Table 6.2) show an advantageous trade off between reliability in rejecting invalid characters and introducing some degradation in the recognition of valid patterns. It can be observed that initially

	Confidence Level					
epochs	0.15	0.25	0.35	0.45	0.55	0.65
0	88.1	88.1	84.4	80.8	81.3	76.0
1	89.4	88.3	85.9	84.5	82.0	81.0
$\Delta 1 =$	+1.3	+0.2	+1.5	+3.7	+0.7	+5
3	88.5	87.1	86.6	84.2	82.6	81.3
6	87.1	84.7	85.2	82.4	81.3	77.6
9	85.8	82.8	82.9	81.7	78.5	78.3
12	84.2	82.1	81.8	79.9	77.9	76.0
15	84.2	82.2	81.0	79.3	77.1	75.7
$\Delta 15 =$	-3.9	-5.9	-3.4	-1.5	-4.2	-0.3

Table 6.2: Digit recognition rates for the MLP with bootstrapping

there is no degradation in the digit recognition rate but, eventually, as the network runs through the test phase, the onset of degradation is observed. For the value of 0.15 for the confidence level after the 15^{th} epoch the decrease is by 3.9 % and for the other values it is by about 3 % on average. These results clearly demonstrate an advantageous compromise between spurious pattern rejection and the recognition rate of valid characters.

Another interesting characteristic observed in Table 6.3 (which shows the evolution of the digit error rates in the recall phase) is the fact that there is a substantial decrease in the error rate in all the cases reported as the network changes towards the more reliable structure. This is consistent with the objective of increasing the reliability of systems, where patterns for which there is low confidence about their identity are rejected, instead of being misclassified.

	Confidence Level					
epochs	0.15	0.25	0.35	0.45	0.55	0.65
0	5.3	5.3	3.7	2.6	3.0	1.7
1	5.3	4.3	3.7	2.4	2.8	1.7
$\Delta 1 =$	-0.0	-1.0	-0.0	-0.2	-0.2	-0.0
3	3.4	3.5	2.2	1.9	1.8	1.3
6	3.0	2.5	1.8	1.6	1.3	0.9
9	2.2	1.9	1.4	1.3	1.1	1.1
12	2.1	1.8	1.4	1.1	1.0	0.9
15	2.2	1.8	1.4	1.1	0.8	0.8
$\Delta 15 =$	-3.1	-3.5	-2.3	-1.5	-2.2	-0.9

Table 6.3: Digit error rates for the MLP with bootstrapping

6.6 Integrating the Bootstrap with Other Network Configurations

A significant advantage of the idea of bootstrapping is that this procedure can be integrated with other network configurations, more reliable than the standard MLP regarding the rejection problem, in order to further enhance their rejection capabilities.

In Chapter 4, it was shown how the standard MLP can be turned into two more reliable structures in this respect through modifications in the network structure. The first configuration is the GMLP network, where each processing unit defines a semi-localised receptive field as a result of the use of a Gaussian as the network's activation function, instead of the usual sigmoid, in conjunction with the inner product as its propagation rule. The second is the MLP which uses additional direct connections from the input to the output layer of the network (DMLP). In this section, a comparison is made between the results obtained with the GMLP and the DMLP operating alone in the pattern classification problem described above with the results achieved when the rejection mechanism is incorporated into the network. The experiments are carried out with the chosen value of 0.15 for confidence level, for illustrative purposes.

The recognition and rejection rates are measured after each presentation of the test set of 2000 digits and 2600 letters over a period of 15 test epochs. The results are shown in Table 6.4, also including the previous results with the standard MLP integrated with the bootstrap mechanism. As in the case of the previous experiments, epoch 0 in Table 6.4 represents the results with the networks when no bootstrap mechanism is applied and epoch 1 corresponds to the case where all the patterns presented are previously unseen.

A further improvement in the rejection rates is seen in all cases when the bootstrap mechanism is employed, and this is seen to be especially effective for the GMLP network. For this network, the rejection rate is increased from 34 % to 47 %. This corresponds to an improvement of about 30 % over the performance of the standard MLP architecture (17 %). After the 15^{th} pass 77 % of the letters are rejected by the GMLP network. Again, the fast enhancement in rejection performance also introduces, with time, some degradation in the recognition rate of valid digits. In the case of the GMLP, there is no initial decrease in the recognition rate but after the 15^{th} epoch the degradation is by about 9 % (from 87 % to 78 %). Once again, some trade-off is observed between establishing a more reliable structure and losing some performance in the correct classification ability. This is supported by the fact that the digit error rate also drops from 3.8 % to 1.4 % in this case.

A similar pattern of behaviour is also seen in the case of the DMLP network, although in this case it is less marked. After the 1^{st} epoch, the increase in the letter rejection rate is from 29 % to 41 %, with a small decrease in the rate of

	Digit Recog.			Letter Rejec.			Error Rate		
epochs	MLP	GMLP	DMLP	MLP	GMLP	DMLP	MLP	GMLP	DMLP
0	88.1	87.0	89.5	17	34	29	5.3	3.8	4.6
1	90.2	88.7	88.2	32	47	41	5.3	4.3	4.1
$\Delta 1 =$	+2.1	+1.7	-1.3	+15	+13	+12	+0.0	+0.5	-0.5
5	87.3	88.3	85.0	57	68	59	3.0	2.0	3.3
10	85.0	81.6	83.5	68	70	65	2.1	2.2	2.8
15	84.2	78.2	82.6	69	77	67	2.0	1.4	2.5
$\Delta 15 =$	-3.9	-8.8	-6.9	+52	+43	+38	-3.3	-2.4	-2.1

Table 6.4: Networks integrated with the bootstrap mechanism

correct classified patterns by 1.7 %. After the 15^{th} epoch, the letter rejection rate is improved further to 67 % (an increase by 38 %) and the degradation in the digit recognition rate is from 89.5 % to 82.6 % (a decrease by 6.9 %).

6.7 Practical Consideration of the Bootstrap Mechanism

In order to operate the bootstrap mechanism in a real-world environment, some practical issues concerning the monitoring of the system performance have to be addressed. It has been observed in the experiments reported that the continuous use of the bootstrap mechanism may incur some loss of performance in terms of the recognition of valid patterns. Although it has been seen that this degradation in performance is advantageous when it is considered against the benefits gained in terms of spurious pattern rejection, it is important to explore some means of dealing with these possible degradations in system performance.

One of the strategies that can be proposed is to continuously evaluate the performance of the network in order to decide if the bootstrap rejection mechanism should be allowed to continue its operation or if it should be stopped at a certain point to maintain at the current level the recognition of valid patterns / rejection of spurious patterns. Figure 6.9 illustrates the components of a practical neural network system designed on the basis of the bootstrap mechanism.

The system is mainly composed of two networks, one of which is *static* most of the time and which, therefore, does not change its structure after being trained on the a priori defined training classes, and the other of which is *dynamic* and modifies its responses to the inputs whenever a bootstrap signal indicates that it should do so. The static network is the one that actually provides the system outputs to the input patterns.

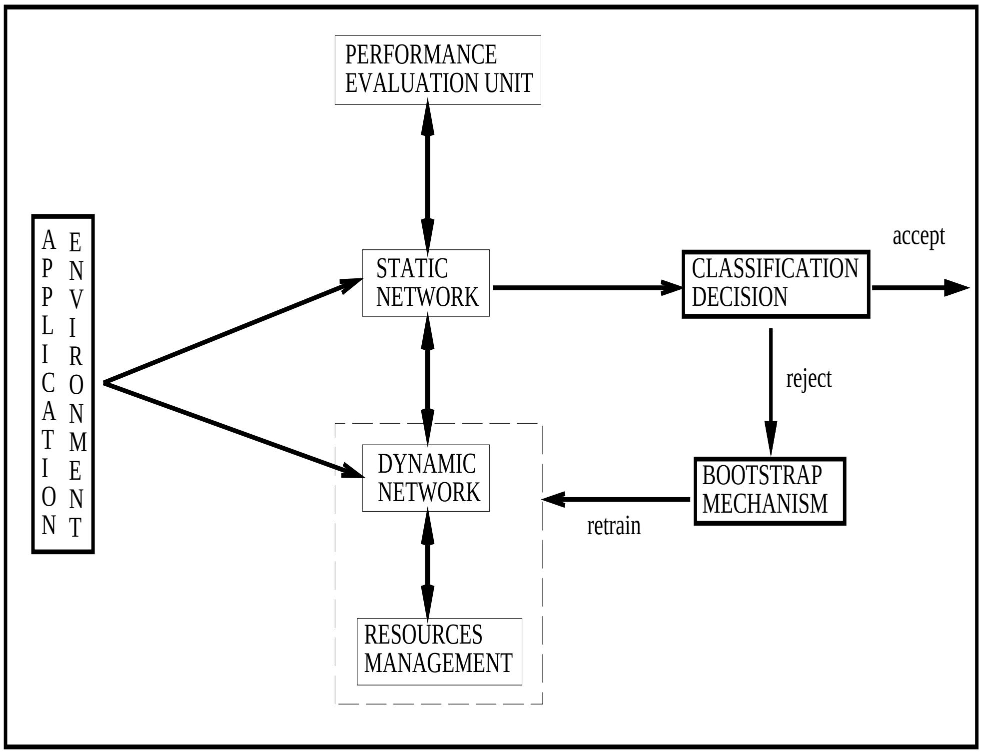


Figure 6.9: Neural network based system with bootstrapping

The steps required before applying the system to the task environment are as follows : initially, after being submitted to the supervised training process, the performance of the static network is estimated on an independent test set belonging to the trained classes, as usual. This performance is stored into a *performance evaluation unit*, responsible for making comparisons of the system performance in the future. The values of the parameters of the static network are then fully copied onto the dynamic network and the system now passes to the phase where it can be used in the application environment.

In the usage phase, when a pattern is rejected by the static network in accordance with the criterion of the certainty level (see section 6.3), the bootstrap signal activates the re-adaptation of the dynamic network. The static network, however, continues its operation as normal in order to not affect the throughput of the system. When the re-adaptation ceases, the parameters of the dynamic network are copied onto the static network and the classification process carries on as before.

From time to time, the performance evaluation unit carries out an estimation of the current performance of the dynamic network on the same test set used previously before copying its parameters to the static network. If there is any substantial degradation, the values of the parameters in the static network are not changed and, instead, it is the dynamic network that receives the values of the weights from the static network. Again, the classification process continues indefinitely.

The final component of the system that has to be considered in practice, as shown in Figure 6.9, is a *resources management unit* which is responsible for evaluating the level of information saturation in the dynamic network. The role of this unit is also to add new processing elements whenever it determines that the network performance is being affected as a result of too many input/output mappings being stored into the network. Different strategies can be considered for determining how the dynamic network is augmented with the inclusion of new units. Again, the static network is only modified after the process of re-adaptation is completed in the dynamic network.

6.8 Conclusion

The work reported in this chapter has considered the problem of the rejection of spurious patterns in MLP networks through the development of a dynamic mechanism based on the ideas of bootstrapping which can be defined to gradually modify the response of an MLP network across the input space, in the recall phase, without the need of supervised training. Although this mechanism is dependent on the occurrence of patterns rejected by the network originally in order for re-adaptation to take place, it provides a way of changing the network's knowledge of the pattern space if and when any spurious pattern is presented to the network. Moreover, it has been shown that it can be used with network configurations which are inherently more reliable in this respect (the GMLP and the DMLP) in order to further enhance their rejection capabilities.

In practical situations, the occurrence of spurious patterns should be much less frequent than that assumed in the experiments of letter rejection reported with the bootstrap rejection mechanism. However, the experiments have been successful in demonstrating how the network performance can be altered to boost rejection performance in an interesting and powerful unsupervised manner.

Future work will consider the actual implementation of the system described and its evaluation in a fully practical environment.

Chapter 7

Conclusion

This thesis has studied the problem of the detection of spurious or novel patterns in feedforward neural networks in the context of classification tasks. The main object of this investigation being the multilayer perceptron network trained with the backpropagation algorithm.

The major objectives of this work have been to understand the reasons why the multilayer perceptron classifies, as valid, patterns completely different from the training classes with a high degree of confidence and to propose different strategies for increasing the rejection abilities of the model. The main contribution of this research has been to show how the observed inability of the multilayer perceptron for detecting spurious patterns can be consistently improved both through the introduction of novel modifications in the network structure and unit function — the DMLP structures and the GMLP network — and through the addition of auxiliary mechanisms integrated with the network — the guard unit mechanism and the bootstrap mechanism.

The investigation of the proposed methods through the use of the iterative inversion of multilayer networks by gradient descent, through the visualisation of the decision regions generated by the networks in 2-D classification problems and through the experiments in the practical application of the classification of handwritten characters provide an overall and comprehensive examination of the performance and abilities of these different methods.

7.1 Summary of the Thesis

Chapter 2 described the problem of poor extrapolation from the training classes observed in the multilayer perceptron where patterns with random characteristics are accepted as genuine by the network. The technique of iterative inversion in multilayer networks through gradient descent was used to illustrate this problem in the classification of handwritten characters. A very useful visualisation experiment was developed in Chapter 2 which aids the understanding of the factors that influence the poor performance of MLP networks in the rejection of spurious patterns. This is achieved through the observation of the decision regions generated by the network in 2-dimensional classification problems.

Chapter 2 also examined the effectiveness of the negative training approach to avoiding the recognition of "false" patterns. This technique is based on an extensive presentation of random patterns during the training phase that is intended to create boundaries between the training classes and the rest of the input space. Experimental work carried out on the classification of handwritten characters in the difficult problem of separating letters and digits made it possible to conclude that this method does not provide very reliable solutions in practical applications. This is because it relies on the assumption that the random patterns used to train the network are good representatives of the rejection classes. These observations were also corroborated by results obtained through the inversion of MLP networks applied to the same training data.

Chapter 3 has shown how the integration of the MLP with an auxiliary network, the guard unit structure, makes it possible to combine the good generalisation properties of the standard MLP with a network more suited to the detection of patterns that are very different from the training classes. The concept of using a combined architecture is shown to provide an overall enhancement in classification performance where part of the network, the multilayer perceptron part, is responsible for deciding to which training class an input pattern is most likely to belong and, the auxiliary structure, the guard unit layer, is accountable for checking the degree of similarity between the input and the training classes so as to decide whether or not it can be accepted as a genuine pattern (no matter to which class it might belong). Network inversion carried out with these combined structures have clearly demonstrated the benefits obtained, and experiments based on the classification of handwritten characters have also shown that the system's rejection capabilities are substantially enhanced with only a small degradation in the correct recognition rates. More importantly, the error rates observed do not increase with this integrated architecture and these rates are, on the contrary, usually reduced.

Chapter 4 has investigated ways of modifying the standard MLP so that the network itself can become, without the use of secondary mechanisms, more rigorous with respect to the acceptance of input patterns. This chapter has described and proposed various novel alternative network configurations, the MLP^{par}, the MLP^{nor}, the DMLP and the GMLP networks, and has used the technique of visualising the decision regions constructed by these networks in 2-dimensional classification problems to give some insights into how these different network constructions should perform in relation to the rejection problem.

The experimental work carried out on the classification of handwritten characters has identified two particular configurations as providing the best improvements in terms of spurious pattern rejection capabilities. These are the MLP with additional direct input-output connections (DMLP) and particularly the MLP which uses a Gaussian as its activation function (GMLP). An overall enhancement in rejection performance was obtained with these networks at the cost of a slight loss in the recognition rate for valid characters, but with a reduction in the error rates observed. The GMLP and the DMLP also represented an improvement over the MLP in terms of general classification capacity with respect to the error rate of valid patterns and the rejection rate of spurious patterns obtained at similar rates of recognition performance.

The technique of network inversion was also employed to further illustrate that the GMLP and the DMLP show a higher degree of correlation between acceptable inputs and class prototypes. The most important aspect about these network configurations is that they improve rejection performance without any substantial change in the original network in terms of both network size and the computational complexities involved in the process of network training.

Chapter 5 examined radial basis function classifiers and an experimental comparison was made with the other models described in the thesis. These experiments confirmed the natural ability of RBFs to deal with the rejection problem but have also importantly shown that networks such as the GMLP and the DMLP may offer a better overall classification performance, with a much smaller number of processing units required to solve a given problem. Although RBF networks can offer the benefit of fast network training, it requires many more processing units than the standard MLP to provide good generalisation. The GMLP and the DMLP presented an enhancement over the MLP network comparable in many cases to that achieved with the RBFs with only a judicious change in the MLP structure.

Finally, Chapter 6 presented a novel bootstrap rejection mechanism for gradually enhancing the rejection performance of the MLP network during its usage phase. The concept of the bootstrap-like rejection mechanism was developed in response to the imposed constraint that in real environments there is usually very little (or no) a priori information about the classes of patterns that should be rejected in a given situation, which in turn makes it impractical for a supervised training procedure to be applied with respect to these classes. The most appealing and interesting aspect of this mechanism is exactly the fact that it modifies the decision regions generated by a network in a classification task in a completely unsupervised manner, taking into consideration only the responses of the network itself to change its classification decisions. It was shown, through the experimental work in the character recognition problem, that the mechanism not only can improve the performance of the standard MLP but it can also be employed with the DMLP and the GMLP to further enhance their rejection capabilities.

7.2 Comparing Different Strategies

Table 7.1 presents a comparison of the results obtained with the multiple guard unit network (MGU) in the handwritten character recognition application with those of the GMLP, the DMLP^t_{80} and RBF networks. The results show different characteristics of the MGU as compared to the other networks, related to the values of the confidence level used to classify the input patterns. Consider initially the comparison between the MGU and the GMLP and the DMLP^t_{80}.

Examining the absolute values of the rejection rates achieved by the networks with each confidence level, it is seen that when the confidence level is small the performance of the MGU structure is superior to those of the GMLP and the DMLP^t_{80}. In the particular case of the confidence level equal to zero, while the GMLP and the DMLP^t_{80} cannot reject any pattern, the MGU is able to reject 28 % of the letters at a small cost of reducing the digit recognition rate by around 2 % (when compared to the GMLP and the DMLP^t_{80}). With the increase of the confidence level, the performances of the GMLP and the DMLP^t_{80} approximate that of the MGU, and this leads eventually to the achievement of better rejection rates. This is explained by the fact that the *relaxation parameter* employed in the MGU network was kept constant and only the confidence level associated with the MLP part of the integrated architecture (MGU/MLP) was changed from 0 to 0.65.

	Digit Recognition Rate				Letter Rejection Rate			
confidence	GMLP	$DMLP^t_{80}$	RBF	MGU	GMLP	$DMLP^t_{80}$	RBF	MGU
0	91.0	92.2	91.1	89.4	0	0	0	28
0.15	87.5	88.3	85.7	86.7	36	29	44	41
0.25	86.7	85.7	80.3	84.7	44	42	63	44
0.35	83.5	82.9	74.9	83.4	55	53	74	52
0.45	77.6	78.4	66.7	79.2	65	62	82	57
0.55	71.4	73.6	55.8	75.8	76	71	89	69
0.65	67.5	66.5	—	72.9	80	79	–	72

Table 7.1: A comparison between different networks

Using the digit recognition rates as a point of reference by matching the results obtained for each network with similar results of the others (see Table 7.1), it is seen that the MGU, the GMLP and the $DMLP^t_{80}$ all presented similar letter rejection performance (with a slight advantage for the GMLP). Consider, for example, the following comparisons : for recognition rates of 86.7 %, 85.7 % and 86.7 % obtained with the MGU, the $DMLP^t_{80}$ and the GMLP, respectively, the corresponding rejection rates are 41 %, 42 % and 44 %. In another case, for recognition rates of 83.4 %, 82.9 % and 83.5 % associated with the MGU, the $DMLP^t_{80}$ and the GMLP, in this order, the rejection rates are 52 %, 53 % and 55 %. Yet in another example, recognition rates of 72.9 %, 73.6 % and 71.4 % for the MGU, the $DMLP^t_{80}$ and the GMLP, respectively, provided 72 %, 71 % and 76 % of letter rejection performance.

Comparing the results obtained with the MGU with those of the RBF, a similar level of evaluation can be made. In terms of the absolute values of the rejection rates, for the case of the confidence level equal to zero, the MGU presents a rejection rate of 28 % as opposed to 0 % with the RBF. For increasing values of the confidence level the performance of the RBF is superior to that of the MGU, as can be seen in Table 7.1. In terms of the rejection rates observed with similar recognition rates, the results with the RBF are slightly better than those of the MGU (by around 3 % on average). The following correlations can be made. For recognition rates of 84.7 % and 85.7 % obtained with the MGU and the RBF, respectively, both networks achieve 44 % of rejection rate. Finally, for recognition rates of 79.2 % and 75.8 % with the MGU and recognition rates of 80.3 % and 74.9 % with the RBF, corresponding rejection rates of 57 % and 69 % are obtained with the MGU and rejection rates of 63 % and 74 % are obtained with the RBF.

In view of the results observed, one important characteristic to take into account when deciding about which network to use refers to the relative computational complexities associated with each of them. The RBF network is fast to train and presented a slightly superior performance than that attained with the MGU, but requires a much larger number of processing units in the network. For example, while the MGU consisted of 220 processing units in total (200 units corresponding to the guard units), the RBF network was composed of 1000 units.

In the case of the GMLP and the DMLP^t_{80}, it can be said these networks represented the best compromise in terms of memory requirements and classification performance, when compared to both the MGU and the RBF. Both networks used the same number of 80 units and still provided rejection capabilities similar to those of the other networks. The MGU is expected to provide a more direct control of rejection performance than the GMLP and the DMLP^t_{80} because it can make use of more (or less) rigorous values of the relaxation parameter according to the needs of each particular application. On the other hand, the GMLP and the DMLP^t_{80} present a natural enhancement in performance without the inclusion of any additional computational mechanism into the network.

7.3 Future Work

There are various ways in which the ideas and methods presented in this thesis can be further explored. In this section, some of these potential investigations are considered for the future.

Perhaps, the most natural extension for the work presented in Chapter 4 is to examine the combination of different MLP configurations into a single network. For example, the addition of direct connections between the input and output layers can also be employed in the GMLP network. Although in principle, it can be conjectured that a further improvement in pattern rejection might be obtained with such a network version, only a detailed exploration of the method can point out whether or not any real benefit is gained with this combination.

The integration of the different strategies developed in the various chapters can also be considered such as the association of GMLPs and DMLPs with the MGU network. One interesting aspect to investigate in this case would be the application of one of these structures (or even, the standard MLP) to an environment where the MGU network is used to teach the MLP part to improve its rejection abilities. For example, when an input pattern is rejected by the MGU network, this information can now be used to inform the MLP part that the current input should not be accepted as valid and the MLP weights can be altered to store this recent information. The tendency is that the MLP part of the system would, with time, absorb the "knowledge" of the MGU network and, consequently, this would make possible the removal of the MGU from the architecture. There are, however, crucial issues that have to be examined concerning the effective control of a practical system with these characteristics, and this obviously requires a very detailed investigation of the system's operation.

The application of some of the approaches explored in the thesis to solve many practical problems is also of great interest. One investigation particularly attractive is to observe the effectiveness of the bootstrap mechanism of Chapter 6 in the control of processes. Consider the control of a process in a factory, a chemical process for example, in an environment characterised by the possible occurrence of many abnormal situations. One strategy for applying the mechanism is to define an experimental phase where the system monitors the process under constrained operating conditions until it reaches a point where it can be left operating by itself. In this situation, a mixture of approaches can also be considered such as the use of the negative training strategy in the system's training phase together with the bootstrapping mechanism during its operating phase. This would provide the system with some prior knowledge about the invalid areas of the pattern space making it easier for the bootstrap mechanism to further enhance the system's knowledge gradually during its normal course of operation.

Another possible study is the application of other models of feedforward neural networks in conjunction with the bootstrap mechanism. Taking into consideration the characteristics of bootstrapping, the class of constructive networks is one type of neural network that might be profitably used in combination with this mechanism [8, 20]. Since the application of bootstrapping introduces a form of continuous incorporation of new information into the network, the scheme behind constructive networks of the gradual addition of more processing units into the existing structure, when necessary, introduces a very natural procedure for network size optimisation in the presence of bootstrapping.

Finally, a very useful future development of a completely different nature, is the construction of a software tool for the continuous animation of the decision regions as they are created by a network when solving a task. The definition of the visualisation procedure used in this thesis to examine the neural network models provided very useful information about their operation, and the addition of an animation tool would constitute an important asset in understanding the behaviour of many network models in a step by step manner.

7.4 Final Remarks

The work reported in this thesis has examined a problem which has direct practical consequences to the application of feedforward neural networks in many real-world tasks, from a fundamental point of view — the problem of the detection of spurious or novel patterns. The investigations carried out have contributed to the understanding of the operation of some important neural network models in relation to this problem, in particular the multilayer perceptron network, and it is hoped that the methods and ideas investigated have provided some contributions to the consolidation of the field of neural networks as an approach that can be efficiently used to solve many practical problems in pattern recognition.

Appendix A

Redefining the Delta-Rule

In this appendix, the equations that define the generalized delta rule are re-examined in order to fulfil the requirements for training the normalised MLP network (MLP^{nor}), described in Chapter 4, with its modified propagation rule.

It was seen in Chapter 1 that the modification in the weights of an MLP trained with backpropagation is achieved by minimization of the sum-squared error obtained at the output layer of the network, according to :

$$\Delta_p w_{ji} \; = \propto - \frac{\partial \; E_p}{\partial \; w_{ji}}. \tag{1.1}$$

The above equation is then re-written as the product of the following factors, which represent the change in the error E_p as a function of the change in the net_{pj} input, and the effect that changing a given weight (w_{ji}) has on the net_{pj} input :

$$\frac{\partial \; E_p}{\partial \; w_{ji}} \; = \; \frac{\partial \; E_p}{\partial \; net_{pj}} \; \frac{\partial \; net_{pj}}{\partial \; w_{ji}}. \tag{1.2}$$

The second part of the above equation is now redefined to capture the modification in the propagation rule, as :

$$\frac{\partial\ net_{pj}}{\partial\ w_{ji}} = \frac{\partial}{\partial\ w_{ji}} \frac{\sum_k\ w_{jk}o_{pk}}{\sqrt{\sum_k\ o_{pk}o_{pk}}},$$

which, after the computation of the appropriate derivatives, leads to :

$$\frac{\partial\ net_{pj}}{\partial\ w_{ji}} = \frac{o_{pi}}{\sqrt{\sum_i o_{pi}^2}}. \tag{1.3}$$

Now, taking into consideration the first part of Equation 1.2 and defining it to be :

$$\delta_{pj} = -\frac{\partial\ E_p}{\partial\ net_{pj}}, \tag{1.4}$$

the following final equivalent form for Equation 1.2 is obtained :

$$-\frac{\partial\ E_p}{\partial\ w_{ji}} = \delta_{pj}\frac{o_{pi}}{\sqrt{\sum_i o_{pi}^2}}.$$

As in Chapter 1, in order to implement gradient descent in E, the weight changes in the network have to be given according to :

$$\Delta_p w_{ji} = \eta\delta_{pj}\frac{o_{pi}}{\sqrt{\sum_i o_{pi}^2}}. \tag{1.5}$$

It is seen, however, that Equation 1.4 is also dependent on the propagation rule and has, therefore, to be re-examined. To compute $\delta_{pj} = -\frac{\partial\ E_p}{\partial\ net_{pj}}$, the chain rule is applied again to write this derivative as the product of two factors :

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}} = -\frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial net_{pj}} \tag{1.6}$$

The second factor remains unchanged since $o_{pj} = f_j(net_{pj})$ and therefore :

$$\frac{\partial o_{pj}}{\partial net_{pj}} = f_j'(net_{pj}),$$

which is the derivative of the activation function f for unit u_j.

Now, to compute the first factor in Equation 1.6 two cases have to be considered. For the case where u_j is an output unit, no change in the original equation is necessary and the value of δ_{pj} for any output unit u_j is given by :

$$\delta_{pj} = (t_{pj} - o_{pj})f'(net_{pj}).$$

where t_{pj} is the value for the j^{th} component of the desired output pattern for the input pattern p.

For the case where unit u_j is not an output unit, the chain rule is used to define :

$$\sum_k \frac{\partial E_p}{\partial net_{pk}} \frac{\partial net_{pk}}{\partial o_{pj}} = \sum_k \frac{\partial E_p}{\partial net_{pk}} \frac{\partial}{\partial o_{pj}} \frac{\sum_i w_{ki} o_{pi}}{\sqrt{\sum_i o_{pi}^2}}$$

$$= \sum_k \frac{\partial E_p}{\partial net_{pk}} \frac{w_{kj}}{\sqrt{\sum_i o_{pi}^2}} - \frac{net_{pj} o_{pj}}{\sqrt{\sum_i o_{pi}^{n+1}}}$$

and finally yields :

$$= \sum_k \delta_{pk} \frac{w_{kj}}{\sqrt{\sum_i o_{pi}^2}} - \frac{net_{pj} o_{pj}}{\sqrt{\sum_i o_{pi}^{n+1}}} \tag{1.7}$$

where $n+1$ is the number of connection weights of unit u_j.

In this case, with the substitution of the two factors in Equation 1.6, the value of δ_{pj} is calculated by :

$$\delta_{pj} = f'(net_{pj}) \sum_k \delta_{pk} \frac{w_{kj}}{\sqrt{\sum_i o_{pi}^2}} - \frac{net_{pj} o_{pj}}{\sqrt{\sum_i o_{pi}^{n+1}}} \tag{1.8}$$

This concludes the modifications in the equations for training the MLP^{nor} network. Then, the normal process of computing the values of the δ's at the network's output layer and backpropagating them to the preceeding layers can be executed taking in consideration the altered equations.

Bibliography

[1] Bairaktaris, D. (1994). *Dynamic adaptation scheme (DAS) - feature discovery in dynamically evolving environments*. Technical Report, October, Human Communication Research Centre, University of Edinburgh, Scotland.

[2] Ballard, D. (1986). *Cortical structures and parallel processing: structure and function*. The Behavioural and Brain Sciences, 9, 67-120.

[3] Barto, A.G., Sutton, R.S., Anderson, C.W. (1983). *Neuronlike adaptive elements that can solve difficult learning problems*. IEEE transactions on Systems, Man and Cybernetics, Vol. SMC-13, No.5, pp. 834-846.

[4] Baxt, W. (1993). *The applications of the artificial neural network to clinical decision making*. Neural Information Processing Systems : Natural and Synthetic, November, Denver, CO.

[5] Beale, R., Jackson, T. (1991). *Neural computing: an introduction*. Adam Hilger, IOP Publishing Ltd.

[6] Bishop, C. (1994). *Novelty detection and neural network validation*. IEE Proceedings: Vision, Image and Signal Processing, Vol. 141, No. 4, 217-222.

[7] Broomhead, D.S., Lowe, D. (1988). *Multivariable functional interpolation and adaptive networks*. Complex Systems 2, 321-355.

[8] Burgess, N. (1992). *The generalization of a constructive algorithm in pattern classification problems*. International Journal of Neural Systems, Vol.3 (Supp. 1992), pp. 65-70.

[9] Burgess, N. (1994). *A constructive algorithm that converges for real-valued input patterns*. International Journal of Neural Systems, Vol.5, No.1, March, pp. 59-66.

[10] Bromley, J., Denker, J. (1993). *Improving rejection performance on handwritten digits by training with rubbish*. Neural Computation 5(3), 367-370.

[11] Carpenter, G.A., Grossberg, S.A., (1987). *A massivelly parallel architecture for a self-organizing neural pattern recognition machine*. Comp. Vision, Graphics, and Image Proc. 37, 57-115.

[12] Courrieu, P. (1994). *Three algorithms for estimating the domain of validity of feedforward neural networks*. Neural Networks, vol. 7, no.1, 169-174.

[13] Cover, T.M., Hart, P.E. (1967). *Nearest Neighbor Pattern Classification*. IEEE Transactions on Information Theory, vol. IT-13, no.1, 21-27.

[14] Chin, H., Danai, K, Lewicki, D. (1993). *Fault detection of helicopter gearboxes using the multi-valued influence matrix method*. In ASME Winter Annual Meeting, New Orleans, Louisiana, November.

[15] Cybenko, G. (1988). *Approximation by superposition of a sigmoidal function*. Mathematics of Control, Signals and Systems, 2, 303-314.

[16] Darken, C., Moody, J. (1991). *Note on learning rate schedules for stochastic optimization*. Neural Information Processing Systems. Lippmann, R.P., Moody, J.E. and Touretzky, D.S. (Editors), pp. 832-838.

[17] Dawson, M.R.W., Schopflocher, D.P. (1992). *Modifying the generalized delta rule to train networks of non-monotonic processors for pattern classification*. Connection Science, Vol. 4, No. 1, 19-31.

[18] Dorffner, G. (1994). *A unified framework for MLPs and RBFNs: introducing conic section function networks*. Cybernetics and Systems, 25(4).

[19] Duda, R.O., Hart, P.E. (1988). *Pattern Classification and Scene Analysis*. New York : Wiley.

[20] Fahlman, S.E. (1988). *An empirical study of learning speed in back-propagation networks*. Technical Report CMU-CS-88-162.

[21] Flake, G.W. (1993). *Nonmonotonic activation functions in multilayer perceptrons*. PhD Thesis, Institute of Advanced Computer Studies, Department of Computer Science, University of Maryland, College Park, MD 20742.

[22] Frean, M.R. (1990). *The upstart algorithm*. Neural Computation 2, 198-209.

[23] French, R. (1992) *Semi-distributed Representations and Catastrophic Forgetting in Connectionist Networks*. Connection Science, Vol. 4: 365-377.

[24] Goddard, N.H., Lynne, K.J., Mintz, T., Bukys, L. (1989). *Rochester Connectionist Simulator*. Technical Report 233, Computer Science Department, University of Rochester.

[25] Gorman, R.P., Sejnowski, T.J. (1988). *Analysis of hidden units in a layered network trained to classify sonar targets*. Neural Networks 1, pp. 75-89.

[26] Hinton, G.E. (1987). *Connectionist learning procedures*. Tech. Rep. CMU-CS-87-115, Carnegie Mellon University, Computer Science Department.

[27] Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Macmillan College Publishing Company, Inc.

[28] Haykin, S., Bhattacharya, T.K. (1992). *Adaptive radar detection using supervised learning networks*. Computational Neuroscience Symposium, pp.35-51, Indiana University-Purdue University at Indianapolis.

[29] Hertz, J., krogh, A., Palmer, R.G. (1991). *Introduction to the theory of neural computation*. Reading, MA: Addison-Wesley.

[30] Hinton, G.E., Nowlan, S.J. (1990). *The bootstrap Widrow-Hoff rule as a cluster-formation algorithm*. Neural Computation 2, 355-362.

[31] Kohonen, T., Barna, G., Chrisley, R. (1988). *Statistical pattern recognition with neural networks: bench marking studies*. IEEE Annual Int. Conf. on Neural Networks, San Diego, CA.

[32] Kohonen, T. (1984). *Self-organization and associative memory*. Berlin: Springer-Verlag.

[33] Kohonen, T. (1988). *An introduction to neural computing*. Neural Networks, Vol. 1, pp.3-16.

[34] Jacobs, R.A. (1988). *Increased rates of convergence through learning rate adaptation*, Neural Networks, vol.1, pp. 295-307.

[35] Jammu, V.B., Danai, K. (1995). *Unsupervised pattern classifier for fault detection of helicopter power train*. Vibration and Noise '95, April, Venice, Italy.

[36] Japkowickz, N., Myers, C., Gluck, M. (1995). *A novelty detection approach to classification*. Submitted to IJCAI 1995.

[37] Kazlas, P.T., Monsen, P.T., LeBlanc, M.J. (1993). *Neural network-based helicopter gearbox health monitoring system*. In IEEE-SP Workshop on Neural Networks for Signal Processing, Linthicum, Maryland.

[38] LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., and Jackel, L.D. (1989). *Backpropagation applied to handwritten zip code recognition*. Neural Computation 1, 541-551.

[39] LeCun, Y. (1985). *Une Procédure d'Aprentissage pour Réseau à Seuil Assymétrique*. In Cognitiva 85, 599-604. Paris: CESTA.

[40] Lee, Y. (1991). *Handwritten digit recognition using K Nearest Neighbour, Radial Basis Function, and Backpropagation Neural Networks*. Neural Computation 3, 440-449.

[41] Leonard, J.A, Kramer, M.A., Ungar,L.G. (1992). *Using radial basis functions to approximate a function and its error bounds*. IEEE transactions on Neural Networks 3, 624-627.

[42] Linden, A., Kindermann, J. (1989). *Inversion of multilayer nets*. IJCNN Washington.

[43] Lippmann, R. (1989a). *Review of Neural Networks for Speech Recognition*. Neural Computation vol. 1(1), 1-38.

[44] Lippmann, R. (1989). *Pattern classification using neural networks*. IEEE Communications Magazine, November.

[45] Lowe, D., Webb, A.R (1990). *Exploiting prior knowledge in network optimization: an illustration from medical prognosis*. Network 1, 299-323.

[46] Lucky, R.W. (1966). *Techniques for adaptive equalization of digital communications systems*. Bell System Technical Journal 45, 255-286.

[47] Martin, G.L., Pittman, J.A. (1991). *Recognizing hand-printed letters and digits using Backpropagation learning*. Neural Computation 3, 258-267.

[48] Maruyama, M., Girosi, F., Poggio, T. (1992). *A connection between GRBF and MLP*. Massachusetts Institute of Technology, Cambridge, MA, AI Memo No. 1291.

[49] McCulloch, W., Pitts, W. (1943). *A logical calculus of ideas immanent in nervous activity*. Bulletin of Mathematical Biophysics 5, 115-133.

[50] McCloskey, M., Cohen, N. (1989). *Catastrophic interference in connections networks: the sequential learning problem*. The Psychology of Learning and Motivation, 24, 109-165.

[51] Mezard, M., Nadal, J.P. (1989). *Learning in feedforward layered networks: the tiling algorithm*. Journal of Physics A 22, 2191-2203.

[52] Moody, J., Darken, C. (1989). *Fast learning in networks of locally-tuned processing units*. Neural Computation 1, 281-294.

[53] Minsky, M., Papert, S. (1969). *Perceptrons*. Cambridge: MIT Press.

[54] Niranjan, M., Fallside, F. (1990). *Neural networks and radial basis function in classifying static speech patterns*. Computer Speech and Language 4, 275-289.

[55] Omohundro, S.M. (1990). *Geometric learning algorithms*. Physica D 42, North-Holland, 307-321.

[56] Parker, D.B (1985). *Learning logic*. Technical Report TR-47, Center for Computational Research in Economics and Management Sciences, Massachusetts Institute of Technology, Cambridge, MA.

[57] Pratt, L., Hoeper, P. (1992) *An X-based hyperplane animator*. Technical Report, Computer Science Department, Rutgers University.

[58] Ploix, J., Dreyfus, G., Corriou, J., Psascal, D.(1994). *From knowledge-based models to recurrent networks: an application to an industrial distillation process*. Neuro-Nimes 1994.

[59] Poggio, T., Girosi, F. (1990). *Networks for approximation and learning*. Proceedings of the IEEE 78, 1481-1497.

[60] Pomerleau, D.A. (1992). *Neural network perception for mobile robot guidance*. PhD Dissertation, School of Computer Science, Carnegie Melon University, Pittsburgh, PA.

[61] Renals, S. (1989). *Radial basis function network for speech pattern classification*. Electronics Letters, Vol. 25, 437-439.

[62] Renals, S., Morgan, N., Cohen, M., Franco, H., Bourlard, H. (1992). *Improving statistical speech recognition.* International Joint Conference on Neural Networks. Vol.2, pp.302-307. Baltimore, MD.

[63] Riedmiller, M., Braun, H. (1992). *RPROP- A fast adaptive learning algorithm.* Technical Report (Also Proc. of ISCIS VII), Universitat Karlsruhe.

[64] Roberts, S., Tarassenko, L. (1994). *A probabilistic resource allocating network for novelty detection.* Neural Computation 6, 270-284.

[65] Rosenblatt, F. (1962). *Principles of neurodynamics.* New York: Spartan.

[66] Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986). *Learning internal representations by error propagation.* In Parallel Distributed Processing, Vol. 1, D.E. Rumelhart and J.L. McClelland, eds., pp. 318-362. MIT Press, Cambridge, MA.

[67] Rumelhart, D., Widrow, B., Lehr, M. (1994). *Neural networks: applications in industry, business and science.* Communications of the ACM, March 1994, Vol.37, No.3.

[68] Rumelhart, D., Widrow, B., Lehr, M. (1994). *The basic ideas in neural networks.* Communications of the ACM, March 1994, Vol.37, No.3.

[69] Sejnowski, T.C., Rosenberg, C.M. (1987). *Parallel networks that learn to pronounce english text.* Complex Systems, vol. 1, pp. 145-168.

[70] Schalkoff, R. (1992). *Pattern recognition : statistical, structural and neural approaches.* John Wiley & Sons.

[71] Schiffmann, W., Joost, M., Werner, R. (1993). *Optimization of the backpropagation algorithm for training multilayer perceptrons.* Technical Report, University of Koblenz, Institute of Physics, Rheinau 3-4, W-5400 Koblenz.

[72] Silva, F.M., Almeida, L.B. (1990) *Speeding up backpropagation.* Advanced Neural Computers, Eckmiller R. (Editor), pp. 151-158.

[73] Sontag, E. (1990) *On the recognition capabilities of feedforward nets.* Technical Report SYCON 90-03, Department of Mathematics, Rutgers University.

[74] Sontag, E. (1992)*Feedforward nets for interpolation and classification.* Journal of Comp. Syst. Sci., vol. 45, pp. 20-48.

[75] Smieja, F.J., Muhlenbein, H. (1992). *Reflective modular neural network systems.* Technical Report, German National Research Centre for Computer Science, Germany.

[76] Smyth, S.G. (1992). *Designing multilayer perceptrons from nearest neighbor systems.* IEEE Transactions on Neural Networks, vol.3, no.2, 329-333.

[77] Smyth, P. (1994). *Markov monitoring with unknown states.* IEEE Journal on Selected Areas in Communication, December.

[78] Vasconcelos, G.C., Fairhurst, M.C., and Bisset, D.L. (1993). *Enhanced reliability of multilayer perceptron networks through controlled pattern rejection.* Electronics Letters, Vol. 29, No. 3, 261-263.

[79] Vasconcelos, G.C., Fairhurst, M.C., and Bisset, D.L. (1993). *The guard unit approach for rejecting patterns from untrained classes.* Proceedings of the 1993 World Congress on Neural Networks (WCNN'93), IV 256-259, Portland, OR.

[80] Vasconcelos, G.C., Fairhurst, M.C., and Bisset, D.L. (1995). *Investigating feedforward neural networks with respect to the rejection of spurious patterns.* Pattern Recognition Letters 16 (2), 207-212, February, Elsevier Science B.V. (North-Holland).

[81] Vasconcelos, G.C., Fairhurst, M.C., and Bisset, D.L. (1994). *Efficient detection of spurious inputs for improving the robustness of MLP networks in practical applications.* Neural Computing & Applications, Springer-Verlag, To Appear.

[82] Webb, A.R (1993). *Functional approximation by feed-forward networks: A least-squares approach to generalization*. IEEE transactions on Neural Networks, 5.

[83] Werbos, P. (1974). *Beyond regression : new tools for prediction and analysis in the behavioral sciences*. PhD Thesis, Harvard University.

[84] Werbos, P. (1989). *Backpropagation and neurocontrol: a review and prospectus*. International Joint Conference on Neural Networks, Vol.1, pp. 209-216, Washington.

[85] Widrow, B., Narendra, G.K., Maitra, S. (1973). *Punish/Reward: Learning with a critic in adaptive threshold systems*. IEEE transactions on Systems, Man and Cybernetics, Vol. SMC-3, No.5, pp. 455-465.

[86] Widrow, B., Stearns, S.D. (1985). *Adaptive signal processing*. Prentice-Hall, Englewood Cliffs, NJ.

[87] Williams, R.J. (1986). *Inverting a connectionist network mapping by back-propagation of error*. In Proceedings 8th Annual Conference of the Cognitive Science Society, Lawrence Erlbaum, Hillsdale, NJ.

[88] Wolpert, D. (1988). *Alternative generalizers to neural nets*. Neural Networks, vol.1, supp.1, p.474, Abstracts of 1st Annual INNS Meeting. Boston.