Nature of the Learning Algorithms for Feedforward Neural Networks

Elena Pérez-Miñana

Ph.D.
University of Edinburgh
1997

"Debemos preciarnos de ser razonables, pero no de tener razón; de ser sinceros, pero no infalibles." - Joubert

Abstract

The neural network model (*NN*) comprised of relatively simple computing elements, operating in parallel, offers an attractive and versatile framework for exploring a variety of learning structures and processes for intelligent systems. Due to the amount of research developed in the area many types of networks have been defined. The one of interest here is the multi-layer perceptron as it is one of the simplest and it is considered a powerful representation tool whose complete potential has not been adequately exploited and whose limitations need yet to be specified in a formal and coherent framework. This dissertation addresses the theory of *generalisation performance* and *architecture selection* for the multi-layer perceptron; a subsidiary aim is to compare and integrate this model with existing data analysis techniques and exploit its potential by combining it with certain constructs from computational geometry creating a reliable, coherent network design process which conforms to the characteristics of a *generative* learning algorithm, ie. one including mechanisms for manipulating the connections and/or units that comprise the architecture in addition to the procedure for updating the weights of the connections. This means that it is unnecessary to provide an initial network as input to the complete training process.

After discussing in general terms the motivation for this study, the multi-layer perceptron model is introduced and reviewed, along with the relevant supervised training algorithm, ie. *backpropagation*. More particularly, it is argued that a network developed employing this model can in general be trained and designed in a much better way by extracting more information about the domains of interest through the application of certain geometric constructs in a preprocessing stage, specifically by generating the Voronoi Diagram and Delaunay Triangulation [Okabe *et al.* 92] of the set of points comprising the training set and once a final architecture which performs appropriately on it has been obtained, Principal Component Analysis [Jolliffe 86] is applied to the outputs produced by the units in the network's hidden layer to eliminate the redundant dimensions of this space.

Acknowledgements

First of all I want to thank my family, my parents and sister, without whose support this piece of work would never have been completed.

I also want to thank my supervisors, Drs Peter Ross and John Hallam for having agreed to direct my work, for their patience and their helpful insights through this period.

Richard Caley has been the most wonderful assistant that anyone could ask for — thanks for helping with the debugging, the very useful discussions throughout, and for being generally supportive.

I would also like to thank those who agreed to take on the task of reading through the thesis, Mark Orr and Mark Wright, both provided very useful comments and their effort is sincerely appreciated.

Thanks to Yuri for the moral support throughout, her messages have been very welcomed in the hardest moments.

Jeremy, Ed, Chris, Rolando, Daqing, Nam, Kim, Sonia, Gherard, Alistair, — thanks for being around and for understanding some of my not very friendly moments.

Finally I want to thank the people of the universidad Simón Bolívar for helping me to find the funding that covered the costs of my studies.

Declaration

I hereby declare that I composed this thesis entirely myself and that it describes my own research.

Elena Pérez-Miñana
Edinburgh
July 28, 1997

Contents

List of Figures

Chapter 1

Introduction

1.1 The scope of this dissertation

A great deal has been written in the last decades on the subject of Neural Networks (*NN*) and their many potential applications. The most important results have been obtained when employing them as pattern classifiers, associative memories or predictors. Within this framework many things have been said about their apparent ability to learn from examples, deal sensibly with badly defined problems, perform generalisation, and be tolerant to faults, even though a fair amount of work has been done in an attempt to gain a fundamental understanding of these properties, there is as yet scope for further investigation into them. Reliable methods for *NN* design are still required because, even after several decades of study, much of it is based on a process of trial and error.

The comparison of *NN* with alternative techniques, and the theoretical analysis of the properties of the network models which are perceived to be of interest, are important areas of research as it is a means of providing formal insights into the *NN* model's functioning. Indeed in recent years a welcome development in neural network research has been the increase in the willingness of researchers to address this issue. Ripley [1992] and others have compared *NN* with standard statistics pointing out the weaknesses and strengths of each and also the advantages that might be gained in *NN* if certain of the existing statistical techniques are applied as measuring sticks for evaluating a network's performance. Gallinari and his research group [1991] studied the relations between discriminant analysis and multi-layer perceptron used for classification tasks showing the equivalence between the two models; the problem of that study is the strong assumptions used to develop their demonstration, a matter which will be discussed in more detail in an ensuing chapter. Oja and his collaborators,[1992], [1992b], developed a network model which converges when the vectors comprising the weight matrix are those corresponding

to the basis of the principal components of the input space. Other examples are available, these show different types of networks applied to different areas and compared against other models which have been created, apart from the statistical ones already mentioned. Given this situation, it is important to accentuate, at this point, the specific type of network model that has been looked into in this research, of which an analysis and description are contained in chapter 2 of this dissertation. It corresponds to the multi-layer perceptron, a network with no feedback loops, in which the nodes are arranged in layers so that only connections between nodes in subsequent layers are allowed. Figure 1.1 depicts an example of a N-layer perceptron, with $N = 2$, and it is to such structures, the term network will refer to in this work, unless otherwise stated.

In the context of this particular model the contribution of this research concerns the integration of: a) the theoretical elements included in some of the existing geometric constructs defined in the last decade, b) those comprising the multi-layer perceptron, trained using rprop [Riedmiller & Braun 93] and, c) the information provided by the application of Principal Component Analysis (*PCA*) [Jolliffe 86] to the output generated by the units of the network's hidden layer. All these pieces of information were combined in a coherent and complete network design process which achieves two aims: it provides a complexity measure which allows one to decide in a reliable way whether the multi-layer perceptron is a convenient model to capture the mapping of interest and results in a more effective and less ad hoc training process which generates a reasonable architecture with an adequate expenditure of computational resources.

Two specific properties of the network model are of interest: generalisation performance and the selection of a good network architecture. These aspects, and the reasons for studying them are subsequently explained. In both cases, the alternative techniques employed are not only the product of research in the *NN* domain, they also include theories mainly developed in the areas of statistics and computational geometry.

The reader should note that nothing is said about any supposed relationship between *NN* and actual biology. Although it is true that much can be learned by looking at the properties of actual neurons and synapses *etc*, it is not relevant to the study presented, which will be based entirely in the realms of engineering, mathematics and computer science.

1.2 Multi-layer Perceptrons

The basic principles of pattern classification and function approximation, and the use of the multi-layer perceptron model therein, are well documented and therefore it is not necessary to provide an exhaustive review. It is assumed that the reader has some prior knowledge of these

areas, including for example knowledge of statistical pattern classification, standard network models such as the *perceptron*, and an understanding of the concept of supervised learning. In this review a brief description is included, mainly for the purpose of providing some basic notation and nomenclature for what follows. Further detail can be found in, for example, [Hertz *et al.* 91], [Michie *et al.* 94], [Muller & Reinhardt 91], [Rumelhart *et al.* 86a].

1.2.1 Pattern classification

In pattern classification, the main goal is to assign sets of measurements, taken in a given situation, to specific classes. The set of all possible measurements forms an input space I; it is usual to deal with $I = R^n$ for some positive integer n and this is the case of interest in the ensuing discussion. A specific instance provides us with an input vector $\mathbf{x} \in I = R^n$, the elements of which in general correspond either to measurements taken directly from the real world, *eg.* heart pulse, petal length, or perhaps the pixel values of an image, or to features derived from it, *eg.* certain morphological categories, *morphemes*. The objective is to map input vectors $\mathbf{x} \in I = R^n$ to pattern classes. For a problem involving q classes a set $C = \{c_1, c_2, ..., c_q\}$ of q class indices can be defined and hence this process can be modelled by a mapping $f : I \rightarrow C$.

As a very simple example of this type of problem, for which $q = 2$, consider input vectors $\mathbf{x} \in I = R^n$ which contain the symptoms of a patient. It is required that the output of the system be c_1 if the set of symptoms represent a patient classed as diabetic and c_2 otherwise. Clearly the design of a pattern classifier of the type described can be regarded as the design of a device which implements a mapping $f : R^n \rightarrow C$; the multi-layer perceptron provides one of many alternative ways of designing and implementing the desired mapping.

The type of mappings that can be represented with the multi-layer perceptron depend on its constituents. The original *NN* created by McCulloch and Pitts [1943] which corresponds to a very simplistic model of the neuron has a very restricted representational scope. In subsequent years the model has been very much developed and extended and different alternatives have been considered, each generally providing a higher representational capacity than the previous one. The unaltered element in all of these is the basic constituent, the neuron (using biological terminology) or unit (in neurocomputing terms). In the multi-layer perceptron the units belong to one of three categories: a) inputs whose only role is to feed input patterns $\mathbf{x}$ obtained from R^n to the rest of the network; b) hidden units which have no direct connection to the outside world; and c) output units from where the final computation is read off. The main difference between them is the form of the activation function, (g_i associated with unit i); for the input units it is the input value, although some people might consider activation an inappropriate name; for the other type of units, hidden or outputs, the activation function has a particular

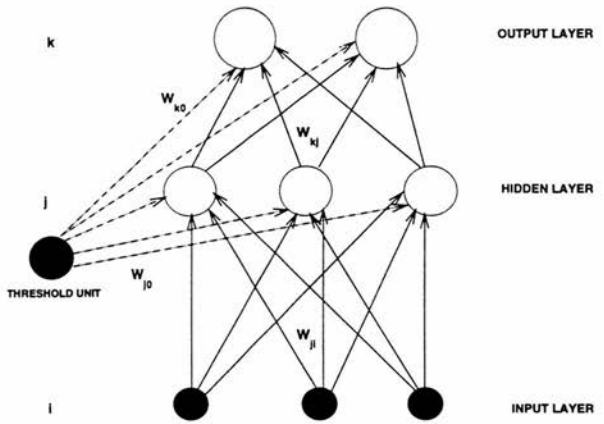


Figure 1.1: **Typical multi-layer perceptron with two outputs and one hidden layer**

form and its selection is important because depending on the one used, the network will be provided with a greater or lesser degree of representational power. In mathematical terms, the forms g_i can take in the different types of units that comprise an architecture can be expressed as:

- $g_i = x_i$ is the form of g_i for the units in the input layer of figure 1.1, *i.e.* $x_i \in R$, x_i is the ith input to the network.
- $g_i = g(\sum_{j=1}^{n} w_{ij}\, x_j - w_{i0})$; where $g(x)$ is the activation function (also known as a transfer function) that ought to be continuous and differentiable; the set, $\{w_{i1}, w_{i2}, ..., w_{in}\}$ corresponds to the weights of all the connections going into unit i, x_j is the output of all the units going into unit i through the set of connections and w_{i0} is the threshold value (bias) associated with unit j and which to simplify the calculations will be considered the weight associated to an artificial unit, the *threshold unit*, so that the expression $\sum_{j=1}^{n} w_{ij}\, x_j - w_{i0}$ becomes $\sum_{j=0}^{n} w_{ij}\, x_j$. The general form of g_i applies equally to the hidden or output units of the network, although its exact form does not necessarily have to be the same. Many applications employ an architecture in which the hidden units use a sigmoid activation function whilst, the output units have a linear function.

A typical network with three inputs, one hidden layer including three hidden units, and a two unit output layer is illustrated in figure 1.1.

The main distinguishing feature of the network when compared to the available alternatives is its architecture. It can be considered a system of interconnected processors, directed interconnections resulting in an ordering of the processors which allows for their categorisation. The network has n inputs which generally correspond to the elements of an input vector $\mathbf{x} \in I = R^n$, and one or more outputs which indicate the class corresponding to any input $\mathbf{x}$. The processors (*nodes*, *units*, p_i) are relatively simple, and each has an associated vector w_s of parameters or *weights* which can be used to alter the specific function that it computes; these are real-valued and in figure 1.1, for example, the vector w_j for hidden unit h_j has the form $(w_{j0}, w_{j1}, ..., w_{jn})$, *i.e.* one element for each of the input units to which h_j is connected. w_{j0}, is the strength of the connection associated to the *threshold* component represented as an extra unit linked to all the p_i comprising the network which clearly is a model able to compute a function $f_{\mathbf{w}} : R^n \rightarrow C$, where $\mathbf{w}$ is a vector containing all of the available variable weights.

1.2.2 Multi-layer perceptron as general function approximators

The networks of interest have also been used to perform more general function approximation tasks. This has been possible because of the choice of activation function associated with the units comprising the architecture. An important work on their use as function approximators was developed by Cybenko [1988] who proved that a continuous multi-layer perceptron with at most three layers of weights can be used to approximate any particular set of functions to an arbitrary degree of accuracy, if the units in the hidden layers have a *sigmoid* activation function. Further evidence in relation to this matter can be found in [Funahashi 89], [Hecht-Nielsen 89]. A result that is of special interest to this investigation is the one arrived at by Cybenko [1989], who showed that with a 2-layer perceptron model comprised of *sigmoid* units it is possible to approximate any continuous mapping if enough hidden units are provided, a number that is not generally known. Similar results were also arrived at by Hecht-Nielsen [1989].

In the context described in the previous section these cases can be modelled by allowing the set of class indices to be $C = R$, such that the network computes some function $f_{\mathbf{w}} : R^n \rightarrow R$. This type of scenario arises if one wants the network to be used to generate the value of a particular measure, *eg.* wind speed, associated with a multidimensional set of other measures; for the wind speed case it could be height of sea wave crests, rope tension on a boat's sails, *etc.* A prediction task in a time series is another possible application of networks used as function approximators.

It must be pointed out that the main results of the research described herein concern a discrete set C of class indices given that it is for this type of task that the multi-layer perceptron with *sigmoid* units has been particularly useful. Therefore the potential of the multi-layer perceptron

as function approximators in any other context, apart form that of the classification tasks, will not be discussed further.

1.3 Supervised learning

Obviously the actual function computed by a multi-layer perceptron network can be altered if any of the parameter vectors $\mathbf{w}_i$ associated with the individual processors is modified; let F denote the class of all functions $\mathbf{f_w}$ that can be computed by a multi-layer perceptron. The task of designing a network using the model involves three main choices: 1) the class of functions computed by each unit, 2) the specific architecture used and, 3) the actual values for the parameters. In the context of *NN* the design of the required network is partly achieved by an algorithm which performs the task of tuning the architecture under its control; this is realised taking into consideration the information of the domain on which it will subsequently operate that is available. The process of adjusting the network extracting the necessary information from the available data, constitutes the model's learning algorithm. For the relevant network the type required corresponds to supervised learning, in particular backpropagation as it was the first effective learning algorithm defined to train a network based on it.

For this type of algorithm there is access to a training sequence,

$$T_p = \{(x^1, o^1), (x^2, o^2), ..., (x^p, o^p)\}$$

of p training examples where $x^i \in R^n$ are inputs to the network and $o^i \in C$ are the corresponding classifications. The sequence T_p will typically be constructed by collecting actual inputs $\mathbf{x}$ for which the correct classification is known, and in general includes only a subset of the possible inputs to the network. It may either provide a true representation of the underlying problem or, more realistically, only the outlook of a portion of the complete underlying structure. The network is designed entirely on the basis of the available examples. Usually the network is built so that an appropriate measure of its *error* with respect to the available training sequence is minimised; this idea is made more precise in subsequent chapters. Once the network has been trained, it is hoped that it will exhibit some ability to *generalise*, *i.e.* that it will be able to make good predictions on the outputs associated with input patterns it did not process during training.

Given the importance of a multi-layer perceptron's generalisation ability, one of the major aims of this dissertation is, to define a supervised learning algorithm that will allow the production of a computationally efficient network, *i.e.* one requiring no more or perhaps a little more computational cost than any other network that could handle the same problem. **Computa-**

tional efficiency includes generalisation capacity and time spent on achieving the *minimum* error during training. It can be phrased in these terms, because the underlying theoretical framework corresponds to one of non-parametric optimisation, of which one of the existing possibilities involves the minimisation of a cost function.

In relation to the time required to reach the *minimum* during training there are two main components in the whole training process on which it is dependent. In the first place, there is the network architecture chosen to tackle the problem because an adequate one will provide a more stable minimisation process given that the number of estimators, *i.e.* hidden units, required to generate a correct network's output is being accounted for. This means that, in the first instance at least, there is no practical reason for the generation of saturated hidden units, *i.e.* units with essentially fixed outputs $\{0, 1\}$ due to the large negative (positive) inputs, because these constitute one of the main problems that must be contended with in the search for the *minimum* in the error surface traversed during training. What must be carefully managed once this correct decision has been made, is in relation to the avoidance of the multiple *local minima* populating the error surface. To this effect the second main component comes into play. It is comprised of the different parameters which must be defined before the training algorithm is set to work on the network of interest. These are the learning rate (α), the momentum (μ), the derivative term employed in the weight update process, *etc.* The first versions of backpropagation used as training processes, [Rumelhart *et al.* 86b], [Rumelhart *et al.* 86a], assigned default values to most of them and for the cases for which they were used the algorithm worked very well. As the number of possible applications grew, some of them dealing with domains that were, for example, not completely known or finite, comprised of classes having low inter-class correlations and high intra-class correlations, *i.e.* not easily discernible clusters in the input space, the task of producing the appropriate network for the problem of interest became more complex and, the need for an effective supervised learning algorithm greater. It is for this reason that many variants of backpropagation have been defined. An important review of many of the existing alternatives can be found in [Schiffmann *et al.* 93].

Computational efficiency must be a major issue independent of the problem that will be solved using the network. In this context it means achieving good training performance and good generalisation at a reasonable computational cost. Although an adequate generaliser is not necessarily the direct result of good training performance it is certainly the case that not achieving the first does not help in reaching the second. Furthermore, many studies have shown ([Alpaydin 91], [Bishop 95], [Fahlman & Lebiere 90], [Fritzke 93], [Hallam & Ross 93], [Mezard & Nadal 89]) that the generalisation capacity of a multi-layer perceptron network is very dependent not only on the value of the weights associated to the connections, their number

and the amount of units involved also play an important part. A way to solve this situation is to extend the training process so that, in addition to updating the values of the weight vectors it is able to arrange the architecture according to the difficulties encountered during the design process, the main idea underlying a generative supervised learning algorithm, which explains it being a better option. Its execution involves two main tasks: a) operating on different network architectures and integrating an analysis of their performance into the training process,thereby providing information which aids in deciding whether an adequate architecture is being created; b) continuously updating the parameters of the learning algorithm by adapting them to the shape of the section of the error surface being traversed during training. This type of supervised learning algorithm can be further subdivided according to the strategy pursued to build the different architectures:

(a) generative constructive algorithm: starts with a minimal architecture, *i.e.* one comprised only of the input-output units. The algorithm grows (expands) the network until a reasonable performance is achieved. Further details on what is understood as **reasonable** in this context will be provided in chapter 2.

(b) generative pruning algorithm: in this case, the initial network is too large for the problem of interest and the algorithm works by reducing the architecture to a more effective size.

(c) generative mixed algorithm: this type of algorithm operates integrating, pruning and constructive techniques for manipulating the network under design.

In order to motivate the study of generalisation, as this property of networks is crucial to this research, a summary of the issues involved is provided in the following section.

1.4 Generalisation

In an historical context, the fact that *NN* usually exhibit some ability to perform generalisation has always been perceived as one of their most important properties, and it is fair to say that this ability has been one of the major reasons for the considerable degree of interest shown in the subject. Although there is no universally accepted definition of the term *generalisation*, it is invariably used, in the context of pattern classification, to describe the ability of a network or other pattern classifier to correctly label an input vector which was not present in the set of examples used to train the system. There are different measures available to quantify this property but the one mostly used corresponds to the percentage of patterns comprising the test set that the trained network is able to correctly classify. It becomes harder to quantify as the complexity and size of the problem domain increases because, as it is measured only on

samples of it, larger domains require evaluations using larger samples so that the conclusions arrived at might be considered reliable.

From the definition of *generalisation* it is clear that the same is closely related to the information capacity of a network, that it depends on the degree of complexity of the network architecture and that whatever measure is defined to quantify it, the same should be useful in the process of answering questions such as: how many relevant features of an input space is a particular architecture able to store? or, how many elements of the input space does it need to see during training to be able to capture them?. In a few words, for a particular network architecture, what is the minimum size of the training set it should process during training so that it might be possible to assert, allowing for a certain margin of error in the assertion, that it will be able to generate the correct output when presented with additional elements of the input space not processed in the tuning stage?.

Much work has been done looking for answers to the aforementioned questions but it has only been possible to provide detailed ones for very particular cases: for tasks involving binary inputs and for network architectures comprised of only one layer of weights and threshold units, *i.e.* the perceptron network model. The computation of the capacities of this type of network are circumstantially discussed in [Gardner 87], [Gardner 88].

The concept of Network Generalisation, can comprise a greater number of restrictions depending on whether the context of study is associated to binary input problems or if it deals with the more general case of real-valued inputs. For the first type of problem as the domains involved are totally specified, countable and finite, it is possible to aim for an architecture that corresponds to the smallest one required for storing the domain under consideration. In problems that involve real-valued domains, the network generalisation capacity can at most be viewed by measuring its performance when presented with novel inputs because it is not possible to guarantee a minimal architecture given that, the domains cannot be completely known. A good generalisation ability is particularly relevant for building a statistical model of the process which generates the data [Bishop 95] given that a network showing good performance on data it did not process during training can be considered to have reliably captured this model.

In the *NN* this is reflected through the generation of a function which will be a faithful representation of the model that is to be captured in the architecture designed. In the particular case of the network of interest, it is necessary to consider whether the tools available therein really provide the means to build the type of mapping that is wanted. Given that Kolmogorov [Kolmogorov 57] proved that it is feasible to build a continuous function of several variables by superpositions of functions of one variable, and that it is possible to view the basic building blocks of the multi-layer perceptron in this manner, there are formal elements which reassert

the usefulness of considering it an alternative for generating the underlying statistical model of the task of interest.

What is not clearly defined is how to proceed at the moment of choosing the architecture required to represent a particular mapping. Given the close existing relation between network modelling and non-parametric statistical inference, it is possible to guide the process by using some of the devices this area provides. Applying some of them has made it possible to produce a detailed definition of the problems involved in the search for a good generaliser and the means to control them. They mainly concern the correct generation and manipulation of the relevant information associated to the domain contained in the application of interest.

1.4.1 Structure of the problem domain

Part of the information subsumed in the statistical model of a particular application corresponds to the structure of the underlying problem domain. A true model must provide a well specified description of the inherent properties of the space to which it is associated and, a very common way to do this is by depicting the inter-relations between the points that populate it. These inter-relations can take a variety of forms, of which the covariance matrix of a sample is a good representative. Depending on the diversity of the existing associations between the elements of a particular problem's domain, the form of the elements describing its inherent properties will be more or less complex, which means it will be easier or harder to build the statistical model.

In the context of the multi-layer perceptron, and in that of many similar methods, this is achieved through the optimisation of a function. For this reason, the training of a network proceeds so that a cost function is minimised. The function can take different forms, but the one of interest in this study corresponds to the mean square error (*MSE*), which is an average measure over the number of patterns comprising the training set, of the difference between the desired output pattern o^p associated to each of the input patterns x^p and, that computed by the network architecture net^p designed to perform the mapping. Assuming that the output space is N-dimension the error measure is computed with the following expression:

$$E = \frac{1}{PN} \sum_{p=1}^{P} \sum_{i=1}^{N} (o_i^p - net_i^p)^2 \tag{1.1}$$

When creating the network, given that the training corresponds to a case of supervised learning, it is necessary to compute various of these errors, one associated with each of the data sets extracted from the problem domain. There are different types of sets for each of the existing

implementations of the network training algorithm, but for the most part the ones considered are: the training set which is the one used for computing the amount by which the weights should be updated so as to reach the *minimum error*, the validation set and the test set, both of which provide a clearer idea of the degree of generalisation achieved with the architecture that is being trained. The validation set allows one to exert control over the overtraining phenomenon and, the test set is used to check on the generalisation level achieved by the final network. Ideally, the optimal architecture would be that with which all the different error measures computed are minimised, *i.e.* the network achieves correct classification of all the elements comprising any of the sets under consideration, but given the intrinsic nature of the optimisation process and, because the aim is to produce a good generaliser, the most adequate architecture usually corresponds to one which generates appropriate values for the errors and recognition rates of the different intervening sets and, which do not necessarily have to be the optimal values. This is achieved by building a sufficiently complex architecture to deal with the problem, one comprising the number and type of elements needed to correctly represent the inherent properties of the space to which it is related. A similar situation is present in the process of curve fitting using polynomials, in which it has been shown [Bishop 95] that using a polynomial with too few coefficients gives poor predictions for new data, *i.e.* poor generalisation, a phenomenon that is also manifest in a polynomial with too many coefficients because in this case it fits too much of the noise on the training data.

To better understand what is meant by the number of required elements for describing the inherent properties of a problem's domain consider the **XOR** problem, shown in figure 1.2, where I is comprised of four points, two having 0 *output*, these are associated to the filled circles in the figure, and two having *output* 1 which are denoted with the empty circles contained in 1.2. Due to the manner the points in I are placed, it is not possible to compute a line that will correctly separate the points with 0 output from those with unitary output, the only way to achieve a proper partition of the plane is through the computation of at least two lines such as the dotted ones shown in the figure, those or any other two with a similar orientation would produce the appropriate partition, *i.e.* the problem domain corresponds to a not linearly separable space. The example, used by Papert and Minsky [1969] to describe the inadequacies of the simple linear *perceptron* in the sixties, identified as the main cause for the restrictiveness in its use, its lack of representational capacity of domains that were not linear separable. The situation was subsequently overcome by the introduction of the hidden layer in the network's architecture. With a multi-layer perceptron the condition of linear separability is no longer required because there are more building blocks available for describing more complex inter-relations that might exist between the elements of a problem's domain.

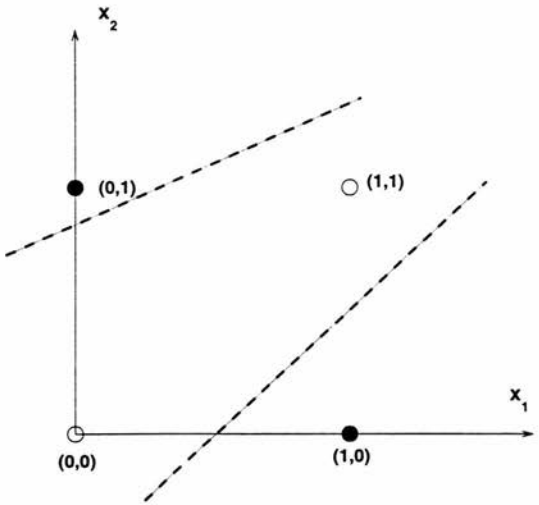


Figure 1.2: **Input space to the XOR problem.**

The difficulty that now needs to be surmounted is how to go about computing the number of required basic building blocks provided by the network model from the training set available and it is at this point that the procedures associated with many existing optimisation processes become useful.

1.4.2 Generalisation and *a priori* knowledge

A generally perceived advantage of *NN* was that they did not usually need to be supplied with a great deal of *a priori* knowledge regarding the task to be solved. This was a definite improvement when compared against other knowledge representation forms, which required relatively complex processes to subtract enough information that would permit the creation of an adequate model. For example, when creating a rule-based knowledge base for an expert system for a diagnosis task, extensive studies have to be performed so as to build the most informative set of rules for the system.

Nowadays, this view has changed and it is very much an open problem to what extent it will in general be necessary to incorporate *a priori* knowledge of a particular problem when designing a network in order to build an appropriate architecture, *i.e.* one showing a good degree of generalisation [Bishop 95]. It can be done in different ways, all of which will require either pre-processing of the input data, post-processing of the network's output or both.

The pre-processing might result in a transformation of the input data or in an elicitation of information providing an insight into the kind of architecture needed to solve the problem under consideration. Post-processing mainly refers to transforming the network's outputs to produce the desired ones.

Part of the alternative solution provided in this dissertation, is comprised of a set of pre-processing techniques. These produce a more manageable input to the network and provide an initial maximal network architecture that is very easily trained and subsequently pruned through an iterative process which finally results in an adequate generaliser.

A priori knowledge has also been used in the form of an extra term which is added to the error function that has to be minimised during training. It corresponds to the technique of regularization, eg. Hinton's work on learning in parallel architectures [Hinton 87], which even though it has been implemented in different ways, can be summarized as a process that encourages smoother mappings by adding a penalty term Ω to the error function to give:

$$\tilde{E} = E + \upsilon\Omega \tag{1.2}$$

Here E is the standard error function described before, and the parameter υ controls the extent to which the penalty term Ω influences the form of the solution. The resulting network mapping is a compromise between fitting the data and minimising Ω which itself can take various forms. The type of decision involves the same type of problem that was described when trying to fit a polynomial to a particular error curve.

1.4.3 Generalisation and network architecture

The different issues which have been mentioned up to now, as important elements to take into account when building a network showing a good degree of generalisation, are relevant because the kind of goals to be achieved generally match those of statistical inference [Bishop 95]: the multi-layer perceptron and the training algorithms used to adjust the weights of a particular instance can be interpreted as algorithms for this type of reasoning.

The aforementioned fact brings into relevance a failing in many of the existing training algorithms. Viewing learning as a method of nonparametric inference enables it to be characterized by the property of consistency: in the appropriate asymptotic limit, these methods achieve the best possible performance for any learning task given to them however difficult this task may be [Geman *et al.* 92a]. This result guarantees convergence of mean-square error towards zero in the large-sample-size limit if the smoothing parameters associated with the estimation method of interest are properly adjusted. For the multi-layer perceptron, the hidden units play the role of smoothing parameters, therefore deciding on the number of them that might be

required is crucial and partly explains the reason for the inappropriate results generated by many of the existing training implementations, as they limit their update process to adjustment of the weights and expect the designers to decide on the matter of the number of units that should comprise the architecture. The appropriateness of the majority of these methods relies on the assumption that the network architecture is the correct one, which leaves the real bulk of the decision process in the network generation with no guidelines on how to proceed.

This state of affairs explains the reason why the learning algorithms which include mechanisms to update a network's architecture generally show much better results. The main problem this new type of algorithm is confronted with, concerns the asymptotic nature of the results described because it means there are little guidelines for smoothing-parameter selection when faced with a fixed and finite training set and is what has led to a search for smoothing parameters based on some function of the data itself. The most relevant theoretical material developed on this subject concerns mainly multi-layer perceptrons with binary inputs. For such networks, a theorem defined by Vapnik and Chervonenkis [1971] provides an upper bound on the number of training patterns required so that a particular architecture will in effect capture some of the structure of the data and can therefore be considered to be a good generaliser. This limit, denoted as the Vapnik-Chervonenkis dimension (d_{VC}) of the network, corresponds to the largest set of data points shattered by it. Further results have been brought forward by Baum and Haussler [Baum & Haussler 89] which give an upper bound on the VC-dimension of a feed forward network of M threshold units having a total of W weights,

$$d_{VC} \leq 2W \log_2(e\,M) \tag{1.3}$$

where e is the base of natural logarithms.

In the context of *sigmoid* multi-layer perceptrons there are no such results but a few mechanisms developed for the analysis of nonparametric estimation in statistics have been heavily exploited (and sometimes improved on) for the study of certain learning algorithms, especially least-squares algorithms to train such networks. For this particular case, it has been possible to divide the error measure into the *bias* and *variance* components [Geman *et al.* 92a] and to subsequently explain the problem that has to be solved through the reconciliation of these two terms. In order to achieve a *minimum* error, which in terms of the *bias* and *variance* is expressed as $error = bias + variance$, [Geman *et al.* 92b], it is necessary to perform a balancing process because it is only possible to reach the desired *optimum* by sacrificing a small amount of the *bias* given that this is the most effective way to get rid of most of the *variance*. Therefore an *optimum*, at least an appropriate result, *i.e.* a reasonable network *generaliser* is only possible through an adequate reconciliation procedure. Further information on how this has been tackled in the development of multi-layer perceptrons will be provided in

a subsequent chapter.

1.5 Benchmarking

One of the reasons for the limited progress of research in *NN* learning algorithms and related issues such as specification of appropriate architectures, *etc.* is due to the disordered state of affairs in this area. A recent study of the evaluation performed in journal papers on the subject, [Prechelt 96], showed that most of them present performance results of the algorithm they describe only for a very small number of problems – rarely more than three. In most cases, one or several of these problems are purely synthetic problems, for instance from the parity/symmetry/encoder family.

Different researchers have tried to create a set of standard benchmarks for some areas of *NN* training algorithm research, with the aim of alleviating this situation. The ones analysed were those described in [Dietterich 94], [Prechelt 94] and because the network model of interest in the present study is the multi-layer perceptron, it was finally decided that the methodology proposed in [Prechelt 94], *i.e.* **Proben1** was the most appropriate.

Proben1 is a collection of problems for neural network learning in the realm of pattern classification and function approximation plus a set of rules and conventions for carrying out benchmark tests with these or similar problems. It contains fifteen data sets from twelve different domains. All datasets represent realistic problems which could be called diagnostic tasks and all but one consist of real world data. The datasets are all presented in the same simple format, using an attribute representation that can directly be used for neural network training. Along with the datasets it defines a set of rules for how to conduct and document neural network benchmarking. The collection of problems for neural network learning in the realm of pattern classification plus the set of rules and conventions for carrying out benchmark tests with these or similar problems included in **Proben1** constitutes the framework for realizing the set of experiments presented in this dissertation. The choice of method was based on the detail it provided, in relation to the rules that had to be followed so as to be able to perform a reliable comparison of the results obtained, and because at the time of requiring a benchmark it was one of the most recent and complete available in the public domain. It is worth noting, that this collection does not pretend to be a complete sample of the space of possible problems, rather it presents a varied number of cases for which the multi-layer perceptron is an adequate modeling tool which was the aim of its designer.

1.6 Data analysis techniques

The task of *classification* occurs in a wide range of human activity. At its broadest, the term could cover any context in which some decision or forecast is made on the basis of currently available information, and a *classification procedure* is then some formal method for repeatedly making such judgements in new situations.

In this dissertation the term is restricted to the description included in section 1.2.1 but due to the broadness of the context in which a classification task is fundamental, many problems can be regarded as classification or decision ones often using complex and very extensive data, a wide variety of approaches have been created to support it. A set of techniques which are particularly useful for this task, is the set of existing data analysis techniques because they have all been defined with the precise aim of providing information. Most of them have originated in statistics but there are other areas, such as computational geometry, that also include mechanisms for producing certain types of information which were considered to be particularly useful in the context of multi-layer perceptron design and which can be considered as a subset of the available data analysis techniques.

In this research, the statistical methods considered to be of use are those that provide information of the important dimensions of an input space. Applying them to the study of the hidden units that constitute part of a network architecture, requires viewing each unit as a dimension of a particular space, *i.e.* the space of the outputs of the hidden units of the network. Subsequently it is generally possible to detect whether there are superfluous dimensions by measuring how many of them are really important for characterizing the space to which they are associated.

The set of techniques taken from computational geometry provide information on the existing degree of proximity of the points comprising the problem's domain, *i.e.* those contained in the training set. This is relevant for the design of a network because in conjunction with details of the network's expected output associated with each of these points a mechanism has been devised for deciding on an initial maximal network architecture that can be trained very quickly to completely recognise the elements of the training set. Also it is possible to measure the degree of complexity linked to it regarding the possibilities of producing a multi-layer perceptron that might be considered a good generaliser.

The succeeding sections give a brief description of the techniques selected together with the main bibliographical references studied to understand their potential.

1.6.1 Voronoi Diagrams; Delaunay Triangulations

Given Cybenko's proof described in [Cybenko 89], *i.e.* that a multi-layer perceptron with one layer of hidden units can approximate any continuous function if enough units are provided in the hidden layer, it is reasonable to restrict the network design process to one hidden layer architectures which means that the decision to make concerns the number of hidden units that should comprise it. An additional argument supporting this approach stems from the fact that, if there is need for more than one layer of units, as such a requirement is induced by a high degree of complexity of the elements comprising the problem's domain, for such cases there are other representation forms that are much more convenient, therefore the network design process need concern itself only with the determination of the number of units in one hidden layer.

The key condition in Cybenko's proof, *i.e.* that enough hidden units be employed, can be expressed in geometric terms as there being enough surfaces to correctly partition the input space. This means generating enough hyperplanes for dividing that space in regions such that, the set of points associated to a particular region all belong to the same class. An estimate on the number of surfaces required can be obtained through the analysis of the domain's structure, an approach already pursued in many current supervised learning algorithms [Alpaydin 91], [Bose & Garga 93], [Fritzke 93], [Karnin 90], [Riedmiller 94], [Romaniuk & Hall 93]. A certain group of geometric constructs which have recently become important concepts in the area of computational geometry, the Voronoi Diagram and its dual the Delaunay Triangulation [O'Rourke 94], [Okabe *et al.* 92], have proven particularly effective in this context as they both provide information on the degree of proximity of the points in an input space, making it possible to assess the degree of difficulty involved in generating a suitable network, and provide the means to compute a reasonable estimate of an initial maximal architecture.

Given a set of points $P = \{p^1, p^2, \cdots, p^n\}$, such that each p^i is taken to be a *site* of the space containing it, it is possible to partition such a space generating a series of convex regions, each containing all the points that are nearest to a site, p^i. The union of all these regions (there is one for each of the elements of P) is what constitutes the Voronoi diagram of P and its dual structure, *i.e.* the Delaunay triangulation, contains the boundaries of each region which when computed can be used to derive a good initial estimate of the number of hidden units the network architecture should have together with a reasonable initialisation of the weight vectors in the input-hidden layer.

The major drawback with this particular set of techniques lies in the computational costs involved in their calculation. Very efficient algorithms have been defined to generate the Delaunay

triangulation, Voronoi diagram of a set of points in a 2-dimensional and 3-dimensional space, [O'Rourke 94], but for higher dimensions the only algorithms available are of an exponential nature [Barber *et al.* 95] which is not very convenient for the design of multi-layer perceptron networks. This difficulty has been solved by generating the geometric constructs always in a 3-dimensional space and using quickhull, an algorithm with a computational cost of $O(n \log(r))$ for an input of n points and an output of r sites. Subsequently the components comprising the geometric structure generated with quickhull are expanded to an (N+1)-dimension space through a process described in Chapter 5 which also includes the computation of the number and value of initial elements comprising the network architecture considered to be necessary to deal with the problem at hand. Although it involves many complicated steps, given that they generate essential information for determining the kind of network required, the computational cost involved is more than justified.

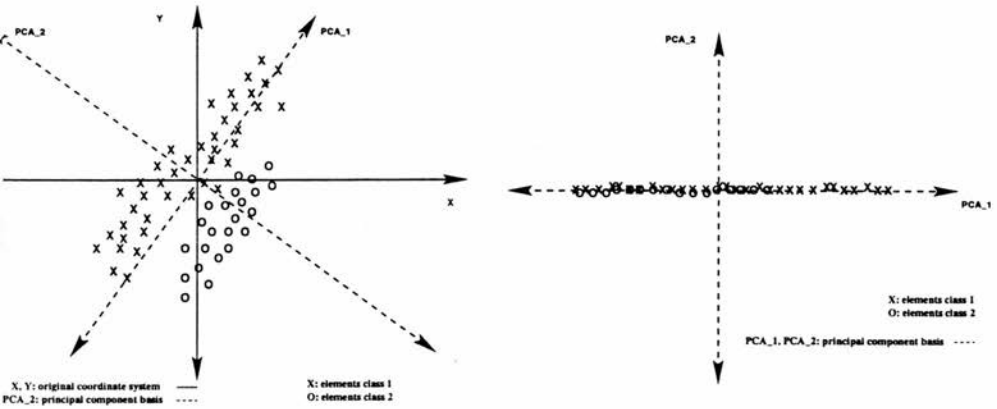
Finally, it is worth noting that there is an alternative implementation for building the geometric constructs required which, at least theoretically, allows for the generation of the Delaunay triangulation of a set of n points in the plane in $O(n)$, *i.e.* the randomised incremental algorithm [Motwani & Raghavan 95]. Nevertheless, as the empirical results described in [Barber *et al.* 95] show that their performance is very similar to that of the one selected in this research, that there is better support for quickhull and, that the randomised algorithm used did not provide as good a level of floating point manipulation as the quickhull program, it did not seem favourable to opt for the randomised alternative.

1.6.2 Principal component analysis, *PCA*

The central idea of *PCA* is to reduce the dimensionality of a data set which consists of a large number of interrelated variables, while retaining as much as possible of the variation present in the data; [Jolliffe 86] provides a very detailed and complete study of the method and its possible applications.

This reduction is achieved by transforming the data set to a new basis comprised of those dimensions which provide important information concerning the domain of interest. The new basis corresponds to the eigenvectors of the *covariance matrix* of the data set being analysed and there is a complete theory explaining and demonstrating the properties which make *principal components* such useful elements.

In the development of *NN* they are useful at two different stages of the network design process: a) the initial data set might be more manageable if it is transformed and reduced to the basis comprised of a subset of its *principal components*. This alternative requires careful consideration



(a) (b)

Figure 1.3: **Badly behaved input set after transforming to the *PCA* basis**

as it might be the case that the transformation process complicates the shape and separability of the regions comprising the input therefore increasing the degree of difficulty for achieving a correct classification with a network; such a situation is shown in the 2-dimension input space presented in figure 1.3, in which if all the points are projected to the highest dimension of the covariance matrix it becomes extremely difficult to distinguish which of those points are members of class 1 and which of class 2, something that was easy to discern in the original space; b) during the training process the output of the hidden units can be analysed and the eigenvectors/eigenvalues of the covariance matrix of this particular set can be calculated providing insight into whether there are superfluous units or if the number is too restricted and they have reached saturation point which means the architecture being trained is not large enough for the problem of interest. A more detailed description of the information provided by this technique and the ways in which it is used to guide the network design process is included in Chapter 4.

1.6.3 Other Data Analysis Techniques

Although there are many methods for data analysis aside of the ones described in the preceding subsection, most of them rely on a small set of fundamental sources of information, which for each method are combined in a particular way and may be applied in conjunction with other types of information. This fundamental set of sources includes: the sample mean, the

covariance/correlation matrix, between-class matrix and the within-class matrix. For a sample X such that:

$$\begin{aligned} X &= \{x^1, x^2, ..., x^n\}, X \subset R^N, |X| = n, \\ x^p &= (x_1^p, x_2^p, \cdots, x_i^p, \cdots, x_N^p), \; attribute \, x_i^p \in R \\ p &\in [1, n], x^p \in C_k, |C_k| = n_k, k \in [1, c]; \end{aligned} \tag{1.4}$$

the terms considered to be basic sources of information are computed using the following expressions:

$$\begin{aligned} \mu &= \frac{1}{n} \sum_{p=1}^{n} x^p && \textbf{sample mean} \\ \sum &= \frac{1}{(p-1)} \sum_{p=1}^{n} (x^p - \mu)(x^p - \mu)^T && \textbf{covariance matrix} \\ r_{ij} &= \frac{\sum_{ij}}{\sum_i \sum_j}, R = [r_{ij}], i \in [1, N], j \in [1, N] && \textbf{correlation matrix} \\ W &= \sum_{k=1}^{c} \sum_{p=1}^{n_k} (x_k^p - \mu_k)(x_k^p - \mu_k)^T && \textbf{within-class matrix} \\ n_k, & \quad \text{number of elements in class } k \\ T &= \sum_{k=1}^{c} \sum_{i=1}^{n_k} (x^{ik} - \mu_k)(x^{ik} - \mu_k)^T && \textbf{total-sum of squares} \\ B &= T - W && \textbf{between-class matrix} \end{aligned} \tag{1.5}$$

where, $\sum_i, \sum_j$, correspond to the square roots of the variances of the attributes x_i^p, x_j^p and, $\sum_{ij} = \frac{\sum_{k=1}^{n} (x_i^k - \mu_i)(x_j^k - \mu_j)}{(N-1)}$ measures the extent to which the two attributes x_i^p, x_j^p are linearly related. In the previous equations it is also possible to identify a main sub-expression and variable, *i.e.* $(x_k^i - \mu_k)(x_k^i - \mu_k)^T$ and μ respectively. The existing equivalence of some of the fundamental terms used in geometry and statistics partly explains the reason for considering them to be useful tools to solve the problem described. A detailed discussion supporting the set of techniques selected and the manner in which they were used in the realm of multi-layer perceptrons is provided in subsequent chapters.

1.7 Organisation of the dissertation

This work is divided into four parts, each containing the following information:

Part I: composed of the present chapter and, chapter 2 which provides a detailed description of the multi-layer perceptron and of conventional backpropagation together with its variants, including those operating on a fixed architecture and those which add/delete units and/or connections during its execution. Finally a detailed argument favouring generative learning algorithms is presented which will be clearly supported by the examples described at the end of this work.

Part II: formed by chapters 3, 4, provides details of the set of techniques on which the generative supervised learning algorithm defined is founded. Chapter 3 contains the kind of information that can be elicited from the geometric constructs generated during the design process of a network, explaining how it might be used together with the reasons behind the particular way in which it is obtained, finally it describes how it is possible to build a complexity measure with this information which shows the degree of difficulty of designing a network for correctly partitioning the problem domain. Chapter 4 provides a detailed discussion of *PCA* describing how it might be applied to decide on the number of hidden units comprising the network architecture.

Part III: considers in detail the problem of designing a good training algorithm for the multi-layer perceptron and of showing its usefulness. Chapter 5 includes a detailed description of it which provides an explanation of the integration process of the complete set of information handled. Chapter 6 provides a description of the kind of benchmark methodology employed and includes various examples that give empirical proof of the advantages obtained by using the algorithm defined; all of them comply with the rules specified in the benchmark procedure followed.

Part IV: is composed of chapter 7 which provides conclusions and suggestions for future research.

Chapter 2

The Multi-layer perceptron and Supervised learning

2.1 Introduction

In common with most research areas the field of *NN* has a vast literature, and an encyclopaedic review would not be feasible or appropriate. This chapter therefore summarises what is considered to be the most important and pertinent work in the context of this dissertation, and attempts to give a representative sample of the published theoretical and experimental results for the different types of multi-layer perceptrons, together with the most relevant learning algorithms that have been designed to train such networks. Given the properties the model has been shown to have [Cybenko 89], [Kolmogorov 57], in relation to the kinds of function that it can implement, it is believed that the main problems found in designing networks using it, lie with the different existing training algorithms. In the subsequent discussion it will be shown that an appropriate algorithm can be specified if the following measures are put into practice:

m1: allow the algorithm to manipulate the complete network architecture. This requires the definition of a set of rules for deciding on the issue of the number of connections and/or units that need to be included and/or deleted.

m2: define a reliable and effective pre-processing phase that will shape the data into a more manageable form for the network building procedure.

m3: proceed from a maximal architecture to a simpler version because there are much more reliable reasons for pruning units than for incorporating units. The initial network that should be input to a pruning process is already a solution to the problem, *i.e.* it is at least able to completely recognise the elements of the training set, therefore the process of eliminating redundant components is able to proceed in a consistent manner stopping

when the network under consideration is not able to achieve such a good recognition rate. The possibility of proceeding in this consistent manner when deleting units is not attainable with a constructive-generative algorithm because, when building a network it is not only important to have enough units, the position of the weight vectors associated to each of those units is also crucial otherwise it will not be an effective classifier. If it is possible to grow an architecture so that, each new unit will be linked to a weight vector situated in an adequate position, then the pruning and the constructive learning algorithms are equally effective. Assuming that it is viable to generate the amount of space required by a generic maximal network, at present it is easier to build more than enough well-situated weight vectors and corresponding hidden units, than it is to work in the other direction.

The multi-layer perceptron can be considered to be the most popular and most frequently used network model but there are still unanswered questions in relation to the behaviour an application using it shows. In order to provide a detailed description of the different forms it can have, the most important types are described in Section 2.2. The simplest of them, a linear perceptron with no hidden layers, is discussed in Subsection 2.2.1. Subsection 2.2.2 contains a definition of other types of perceptron which differ from each other on the activation function used. Subsection 2.2.3 is comprised of a precise definition of the type of network model which is investigated in detail in this research, *i.e.* the multi-layer perceptron comprised of hidden layers incorporating *sigmoid* units.

Section 2.3 includes all the information concerning the learning algorithms that have been defined to train networks based on the model of interest. Given the amount of material on this issue, it is further subdivided in more specific concepts. Subsection 2.3.1 includes a review of the first fundamental training algorithm, *i.e.* backpropagation, outlining its shortcomings. Subsection 2.3.2 provides a description of the first group of important variants that have been defined to tackle these problems which are generally based on the definition of particular ways of manipulating the algorithm's parameters. Their effectiveness, in the literature reviewed, is for the most part explained under the assumption that the network's architecture on which the algorithm is operating is comprised of an appropriate number of hidden units therefore it is not surprising that they only provide better alternatives in very few cases and did not supply an overall solution to the problem of properly training a multi-layer perceptron network. Subsection 2.3.3 includes a detailed description of the implementation which is used as part of the network design method defined in this research. Finally, subsection 2.3.4 contains a discussion on the most relevant network design methods that have been defined in the last decade which, in addition to incorporating a training process, have a set of rules which operate

on the network architecture creating and/or deleting units and/or connections.

In Section 2.4 the relevance of the task of building an adequate architecture is discussed in detail, pointing out how much the success of the task depends in the relative disposition of the points comprising the problem domain, examining the difficulties encountered in the determination of the *minimum* network required for an application, and outlining a set of arguments supporting the construction of a generative supervised learning algorithm. The contents in Section 2.5 explain the reasons for incorporating an effective pre-processing phase, given that the network construction is based on the exploitation of the geometric information of the problem domain. The relevance of this issue is shown in a specially constructed problem. Section 2.6 includes an example requiring a multi-layer perceptron network that is solved using certain geometric information, in this manner the core concern of this study is effectively exemplified. Finally, Section 7.1 summarises the conclusions reached with the analysis described throughout the chapter.

2.2 Multi-layer perceptron

A multi-layer perceptron network takes as input a vector $X^p \in R^n$,

$$(\mathbf{X^p})^t = (x_1^p, x_2^p, \cdots, x_n^p)\,, \tag{2.1}$$

where p indexes the occurrence of $\mathbf{X}$ in a training set and $(\mathbf{X^p})^t$ represents its transpose. The network produces as output a class index $C = (c_1, \cdot\cdot, c_j, \cdot\cdot, c_m)$, $c_i \in \{0,1\}$ and $C \in \zeta$. Every class index, is represented in the set of possible outputs, ζ, with a binary pattern $(0,0,\cdot\cdot,1,\cdot\cdot,0,0)$, in which only one of the components has a unitary value. $c_i = 1$ in the pattern, indicates that the binary label is associated to class i. It should be remarked that it is possible to define ζ in this manner because the multi-layer perceptron is being designed for a classification task otherwise, as already mentioned in section 1.2.2, if the network is used as a function approximator, then the appropriate definition would be, $\zeta = R, m \geq 1$.

The ensuing description of the model is divided in three parts, each corresponding to a particular type of perceptron and each incorporating more general and complex building components than the previous one.

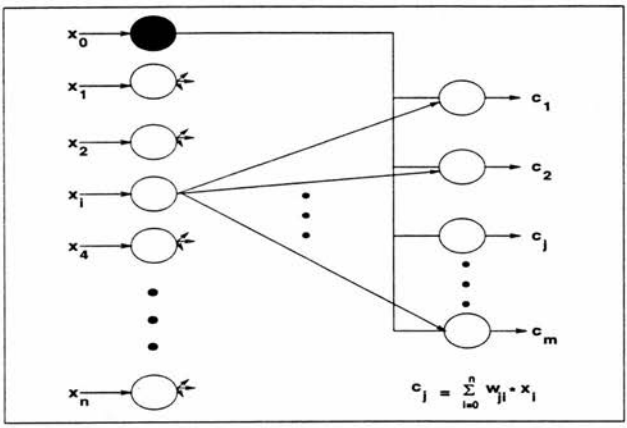


Figure 2.1: **Linear Perceptron with n inputs and m outputs**

2.2.1 Linear perceptron

The linear perceptron has the simplest kind of feedforward structure as illustrated in figure 2.1. Each node i of a linear perceptron computes the function,

$$c_i^p = \sum_{i=1}^{n} w_{ij}\, x_j^p - w_{i0} \tag{2.2}$$

the terms x_j^p are the components of $(\mathbf{X^p})^\mathbf{t}$, and the vector of variable weights associated with the jth node is $\mathbf{W_j} = (w_{j1}, w_{j2}, \cdots, w_{jn})$. If unit j's threshold value, w_{j0}, is represented as the weight of a connection coming from an additional input unit x_0 with a fixed input value of -1, as shown in figure 2.1, the actual computation performed by a linear perceptron network can be formulated as:

$$\mathbf{W} \otimes (\mathbf{X^p})^\mathbf{t} = c^p \tag{2.3}$$

where:

a.- X^p is a known element of the expression as it is part of the training set employed in the design process of the network. This means, that there is a tuple (X^p, o^p), where $o^p \in \zeta$ is X^p's expected output.

c.- c^p is the result of evaluating equation 2.3 using the vector X^p.

b.- $\mathbf{W}$ is an $[m \text{ x } (n+1)]$ matrix, whose components are unknown. It has as many rows m, as there are output units in the linear perceptron and, each row is an $(n+1)$ vector that must be estimated.

It is possible to compute W using equation 2.3, taking into account the information contained in the training set, *i.e.* the tuples (X^p, o^p). In this context, the matrix $\mathbf{W}$ should be comprised of elements such that, at the moment of computing equation 2.3 for a specific X^p, the resulting c^p will be sufficiently close to o^p, *i.e.* $c^p \approx o^p$. This condition must be satisfied with all the elements in the training set.

The learning algorithm is the network model's device responsible for performing this estimation which consists of the minimisation of an error measure. It can be implemented as a non-parametric estimation method because there exists a training set providing information about the relevant problem domain. There are many measures available, all potential candidates for the minimisation process [Hertz *et al.* 91], [Bishop 95], [Rumelhart *et al.* 86b]. Since the measure used in the majority of existing implementations is the error sum of squares (*SSE*), and given that it is the one that has been mostly used in the literature reviewed, the design method described herein is also based on *SSE*. Assuming that the training set is:

$$T_p = \{(x^1, o^1), (x^2, o^2), ..., (x^p, o^p)\} \tag{2.4}$$

The *SSE* has the following form:

$$E = \frac{1}{2}\sum_{k=1}^{p}(o^k - c^k)^2 = \frac{1}{2}\sum_{k=1}^{p}(o^k - W^t \otimes X^k)^2 \tag{2.5}$$

and its minimisation by the learning algorithm is achieved through an iterative procedure that modifies the $W's$ components until *SSE* is sufficiently close to zero.

The formal definition of the linear perceptron is very similar to that of the model on which the linear regression analysis estimation process is based.

The linear regression analysis model involving n independent variables $\{x_1, x_2, ..., x_n\}$ required to approximate the dependent variable Y has the form:

$$Y = \beta_0 + \beta 1\, x_1 + \beta_2\, x_2 + ... + \beta_n\, x_n + E \tag{2.6}$$

where $(\beta_0, \beta_1, ..., \beta_n)$ is the vector of parameters; *estimates* that must be calculated. This is done making use of a sample set of the problem organised as a matrix of observations $\mathbf{X}$ and subsequently applying the least squares solution, which means using the same kind of error

function as in the case of the linear perceptron. If the domains involved in the linear regression analysis process satisfy certain conditions, this solution ensures the minimisation of the error E. Assuming the set of observations contains p occurrences of the model's independent variables $\{x_1, x_2, ..., x_j, ..., x_n\}$, in algebraic terms it can be written as:

$$\mathbf{X^t} \otimes \beta + E \tag{2.7}$$

which allows one to express the least-squares solution for the *estimates* of the regression coefficients in the parameter vector β in a compact way. This least-squares solution, in matrix terms, is that vector $\hat{\beta}$ which minimises the *SSE*:

$$SSE = (\mathbf{Y} - \mathbf{X^t} \otimes \hat{\beta})^{\mathbf{t}} \otimes (\mathbf{Y} - \mathbf{X^t} \otimes \hat{\beta}) \tag{2.8}$$

The solution to this minimisation problem, which is obtained by applying Matrix Calculus, yields the following expression (see [Kleinbaum *et al.* 88] for further details):

$$\hat{\beta} = (\mathbf{X^t} \otimes \mathbf{X})^{-1} \otimes (\mathbf{X^t} \otimes \mathbf{Y}) \tag{2.9}$$

Considering the linear perceptron model with $m = 1$ outputs, the set of unknowns also corresponds to a vector of *estimates* which differ from those included in linear regression analysis only in the terminology used. In the linear perceptron the unknown is a weight matrix ($\mathbf{W}$) while in linear regression analysis the least square estimates comprise the vector $(\beta_0, \beta_1, ..., \beta_n)$. The computations performed for the linear perceptron model yield values for the variable c_i whereas linear regression analysis calculates the *dependent* variable Y. Equation 2.3 is the expression employed for generating c_i, whilst Y is obtained by solving equation 2.6. Taking this information into account, renaming the appropriate variables and reorganising the set of equations described, *i.e.* the least squares result in linear regression analysis, in the linear perceptron case the *SSE* is computed with the following expression:

$$SSE = (\mathbf{C^t} - \mathbf{X^t} \otimes \hat{\mathbf{W}})^{\mathbf{t}} \otimes (\mathbf{C^t} - \mathbf{X^t} \otimes \hat{\mathbf{W}}) \tag{2.10}$$

which means that the most adequate $\hat{W}$ can be calculated directly by:

$$\hat{\mathbf{W}} = (\mathbf{X^t} \otimes \mathbf{X})^{-1} \otimes (\mathbf{X^t} \otimes \mathbf{C^t}) \tag{2.11}$$

The $\hat{W}$ computed in equation 2.11, is a much more adequate way of initialising the $\mathbf{W}$ matrix associated with a linear perceptron network, definitely much more reliable than the *random* initialisation used in most applications. Nevertheless, care must be taken when proceeding in accordance with the set of computations described above. The possibility of obtaining adequate results depends on whether the domains involved in the task of interest, satisfy the set of assumptions on which the feasibility of a linear regression analysis estimate is based. These can be summarised as follows:

h1: **Existence.** For any value of X, it is possible to calculate a specific Y. All these calculations show a behaviour that makes it possible to consider Y as a random variable with a certain probability distribution having finite mean and variance.

h2: **Independence.** The Y-values are statistically independent of one another. Thus the outcome of the calculation obtained using a specific X^1 should not be conditioned in any way to the results achieved by using another X value.

h3: **Linearity.** The mean value of Y is a straight-line function of X. The model on which linear regression is based, in mathematical terms, complies with a linear function and, as the aim of the analysis is to reproduce the underlying mapping that exists between the variables X and Y, this will only be possible if that mapping is naturally linear because, being the simplest type of mathematical model, it cannot be used to reproduce more complex functional relations, *eg.* a quadratic dependence.

h4: **Homoscedasticity.** The variance of Y is the same for any X.

h5: **Normal distribution.** For any value of X, Y has a normal distribution. An assumption that makes it possible to evaluate the statistical significance of the relationship between X and Y as reflected by the fitted line. It is not a compulsory condition to satisfy in the search for an appropriate solution since it is possible to rely on the results computed even if the normal condition is not satisfied.

A complete and detailed discussion of these assumptions can be found in [Kleinbaum *et al.* 88]. Of them all, the most important one, in the context of the linear perceptron model, is **h3** because the other conditions are easily satisfied by applying the central limit theorem [Freund 92], given that what is modeled with the linear perceptron network is a classification task. In this context, the linearity assumption can be expressed in terms of the possibility of producing as many regions as there are classes in the problem so that one region is comprised only of input points belonging to one class. The partitioning is achieved through the generation of appropriate hyperplanes, defined with the weight vectors connecting the input and output units, and the feasibility of their calculation depends on the relative disposition of the points that constitute the input space. For example, in the **binary XOR** problem it is not possible to produce appropriate hyperplanes because, the disposition of the points does not allow the computation of the ones required to correctly partition the input space.

The main advantage of the equivalence described, is that it provides a sound technique to compute an initial set of values for the weight matrix. If the linear perceptron network that results from this initialisation does not recognise the elements comprising the different sets

associated with the complete training process, *i.e.* training, validation and test sets, as it should according to linear regression analysis theory, then one or more of the regression model's assumptions are being violated in which case a linear perceptron is not an adequate option to solve the problem. An alternative solution must be sought.

Extending the output domain to an m-dimensional space, m > 1, the estimation problem can be expressed with:

$$\mathbf{X^t} \otimes \mathbf{W} = \mathbf{C^t} \tag{2.12}$$

and, as the results of linear regression analysis can be easily extended to higher dimension spaces, the desired *optimum* for the linear perceptron in this case is computed with the following $\hat{\mathbf{W}}$:

$$\hat{\mathbf{W}} = (\mathbf{X} \otimes \mathbf{X^t})^{-1} \otimes \mathbf{X} \otimes \mathbf{C^t} \tag{2.13}$$

if the set T_p satisfies the assumptions of interest.

Once again initialising $\hat{\mathbf{W}}$ with the values obtained by applying equation 2.13, subsequently presenting the available data to the network, and calculating the error measure for each pattern, is a reliable, theoretically sound way of determining if the model is a good alternative to solve the problem of interest. A more detailed description of the linear perceptron can be found in [Gallinari *et al.* 91], [Georgiou 93] and particularly [Baldi & Hornik 94] which includes a very detailed analysis of the equivalence of the model and its learning algorithm with linear regression analysis theory.

2.2.2 Non-linear Perceptron

The difference between the non-linear perceptron and the linear perceptron described earlier is restricted to the form of the activation function. There are no hidden units, only forward connections and each and every one if the input units is connected to the complete set of intervening *output* units. Therefore the network shown in figure 2.1 also represents the architecture of a non-linear perceptron.

In this case the activation function does not have to take a particular form, the only restriction being that it be non-linear. Of all the existing functions that comply with this restriction the ones that have been employed most often are:

- $\sigma(net_i) = \frac{1}{(1+\exp(-D\,net_i))}$ for $net_i \in R$ and $D \in R^+$
- $\sigma(net_i) = \tanh(-D\,net_i)$ for $net_i \in R$ and $D \in R^+$

in both cases,

$$net_i = \sum_{j=0}^{n} w_{ij}\, a_j, \tag{2.14}$$

$$a_j = \begin{cases} x_j \\ \sigma(net_j) \end{cases} \tag{2.15}$$

the form of a_j varies according to the type of source, if the connection is coming from an input unit, x_j, or from a hidden unit, $\sigma(net_j)$ and, D is a steepness parameter.

The development of applications employing the non-linear perceptron with no hidden units has not been very widespread since the usefulness of non-linear functions makes more sense in this context as the activation function of the hidden units. It is well known [Werbos 74], [Rumelhart *et al.* 86b], that the condition for requiring a hidden layer in a network is that the pertinent problem domain be not linearly separable because the hidden units make it possible for a network to deal with the situation, whereas just changing the form of the activation function does not produce the transformation required by a problem to become linearly separable. Additionally as can be seen in figure 2.6 the range of both these functions is very limited, $\frac{1}{(1+\exp(-x))} \in (0,1)$, $\tanh(x) \in (-1,1)$, which greatly reduces the applicability of the model, at least as function approximators; the objection does not apply in those cases in which the model is employed as a *classifier*. Each of the output units of a non-linear perceptron with n inputs and m outputs, using the *sigmoid* as an activation function is computed by:

$$c_i^p = \sigma(\sum_{j=0}^{n} w_{ij}\, x_j^p) \tag{2.16}$$

with $\sigma(x) = 1/(1+\exp(-D\,x))$.

As before the *SSE* is the measure most frequently used in the learning algorithm to decide whether the level of performance shown by the network is appropriate and also to determine the amount by which $\hat{\mathbf{W}}$ must be updated so as to achieve better performance. With a *sigmoid* function its form now becomes:

$$\frac{1}{2}(\sum_{k=1}^{p}(o^k - c^k)^2) = \frac{1}{2}(\sum_{k=1}^{p}(o^k - \sigma(W^t \otimes X^k))^2) \tag{2.17}$$

Clearly equation 2.17 has no equivalence with linear regression analysis as in the case of the linear perceptron which means that the set of results therein cannot be extended to this case. Given that this particular type of model is not very useful it was not analysed in great detail and will not be discussed any further.

2.2.3 Multi-layer Perceptron with hidden layers

This type of perceptron model allows the specification of more complex networks which can represent more general functions than the ones that could be replicated with the previously described perceptron alternatives. This enhanced representation capacity makes it the most powerful of all the models described in this chapter and, the one that has been used most frequently in the network applications developed in recent years, therefore the one that is of real interest in this research.

It originated as a result of the set of alterations effected on the linear perceptron which, together with the learning algorithm defined to operate on it, were the main reasons for the renewal of interest in the network model. Of all the applications that have been developed using this new definition the most important one, or at least the one more widely known about, is the network created to solve the **binary XOR** problem [Werbos 74], [Rumelhart *et al.* 86b], because it provided a very effective counter-example to attack the limitations concerning the linear perceptron which were depicted by Minsky and Papert [1969]. The fundamental changes which resulted in a representation form with a much wider applicability were:

1. The flexibility regarding the number of hidden units and, the type of activation function that can be specified for each of them. It is valid to define various layers of such units between the network's inputs and outputs, organised and connected in a feedforward manner. A different activation function can be associated to each of the ones defined in an architecture. For simplicity, the units of any single layer are usually assumed to have the same activation function.

2. The learning algorithm defined to iteratively update the weight matrix $\mathbf{W}$ until the desired *estimate* ($\hat{\mathbf{W}}$) is computed. As this algorithm employs the first derivative of the activation function for updating the different w_{ij} it is necessary that the function be differentiable. This is one of the reasons for the appearance of the alternative activation functions described in the preceding section. Other reasons for the choice of those functions may be found in [Hertz *et al.* 91].

These changes were the result of independent work developed by various authors, [Werbos 74], [Rumelhart *et al.* 86b], in the years following Papert's and Minsky's disheartening discovery [Minsky & Papert 69]. The architecture of a multi-layer perceptron is shown in figure 2.2 and it is usually the case, that the number of connections increases considerably with the definition of a hidden layer. Defining a hidden layer of h units in the network requires the definition of:

$$h \times ((num(units)_{prev-layer} + 1) + num(units)_{next-layer}) \tag{2.18}$$

number of connections whose associated weight values (w_{ij}) have to be appropriately tuned. This means that care must be taken at the moment of deciding whether or not to create a hidden unit and even more when the issue concerns a hidden layer. Consider a network with m inputs and n outputs, this means that there are $n \times (m + 1)$ connections, taking into account the threshold of each output unit. Incorporating a layer of h hidden units to the architecture, the new total number of connections becomes: $h \times (m + n + 1) + n$. Looking at the difference between the number of connections in both cases, *i.e.* $n \times (m + 1) - h \times (m + n + 1)$, there is a particular type of network architecture for which incorporating a hidden layer might result in a simpler network, at least, one with fewer connections. This type of multi-layer perceptron has many more input and output units than hidden units. It is an extreme and very unrealistic kind of example, taking into account that in the majority of the real-world problems, the larger number of units are either in the input space or the hidden layer, which ascertains the need to be careful when deciding on whether or not to incorporate a hidden layer to an architecture.

Since the model first appeared, different authors have provided detailed studies of its representation capacities which show that it is worthwhile to consider it as a useful alternative for knowledge representation. Of all the existing analysis, Cybenko's [1988], which showed that a sigmoid multi-layer perceptron with one layer of hidden units could approximate any continuous mapping is one of the most important given it was one of the first to appear. In [Cybenko 89] he proved that the same model but with two layers of hidden units and, linear activation function for the output units, could approximate any function to an arbitrary degree of accuracy. Similar studies which discuss these properties can be found in [Funahashi 89], [Hecht-Nielsen 89], [Hornik 91], [Judd 87]. The general idea underlying many of them is based on the possibility of expressing any function as a linear combination of step functions whose arguments are linear combinations of the variables associated to the input domain of the function that is to be represented, *i.e.* the universality property [Blum & Li 91]. This property is applicable in the context of sigmoid activation functions because it is possible to approximate the step function using a sigmoid by providing it with sufficiently large-valued arguments, which in the network model is easily achieved by using sufficiently large weights in the connections involved [Bishop 95].

The results described suggest that the efforts should be concentrated on the development of a learning algorithm that will generate an adequate architecture of at most two hidden layers for the application of interest which was the approach taken in this study.

As in the previous cases at the moment of defining a multi-layer perceptron network there is a well-specified training set,

$$T_p = \{(x^1, o^1), (x^2, o^2), ..., (x^p, o^p)\} \tag{2.19}$$

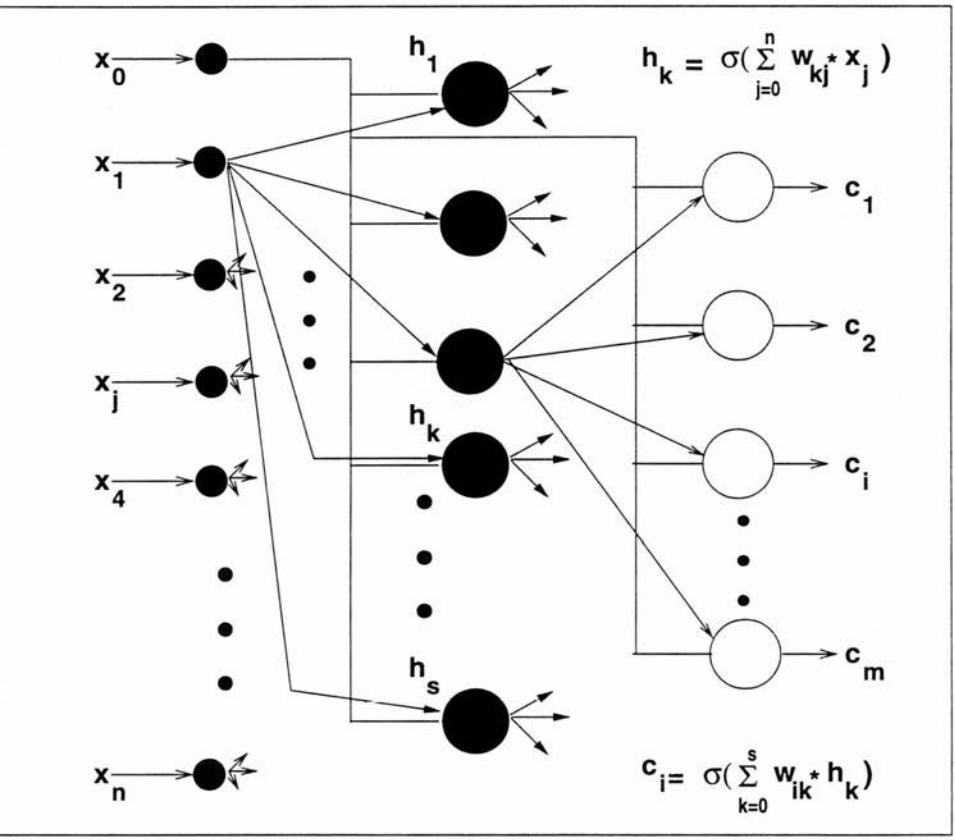


Figure 2.2: **Multilayer perceptron**

and the mathematical definition of the *one-hidden* layer network that has to be created for recognising such a training set, has the following form:

$$\mathbf{c_i} = \sigma(\sum_{j=o}^{l}(w_{ik}\, h_k)) \tag{2.20}$$

where

$$\mathbf{h_k} = \sigma(\sum_{j=0}^{n}(w_{kj}\, x_j^p)) \tag{2.21}$$

corresponds to the output of a generic hidden unit h_k of the architecture; σ represents the activation function employed, which subsequently will be considered to be the *sigmoid* unless otherwise stated, and $\mathbf{c_i}$ is the value produced by the *ith* output unit of the network.

In the same way as with the previous models, the set of unknowns are all the w_{ij} and the related learning algorithm performs the estimation of the most appropriate $\mathbf{W}$ through the minimisation of some kind of cost function. At present, even though the type of objects, *i.e.* functions, that can be represented with this type of network model have been extensively studied and, are pretty well defined, there has been little advancement towards achieving the complete specification of an effective process for building network instances for particular problems. It has not yet been proven that any of the existing supervised learning algorithms is the most effective tool, in a computational sense, for generating the most appropriate multi-layer perceptron network for a particular task. Furthermore, there is no unequivocal specification of what the term most appropriate multi-layer perceptron really implies. It is clear that as its generation is governed by a minimisation process, to be able to consider a particular network a reasonable solution it must minimise the cost function used during its creation. This in itself is a problem because it might very well be the case that for some data the network seems to have reached the *minimum*, for example, when presented with the elements of the training set after a training process has been successfully completed but, looking at the error achieved by the same network working on data it did not process during training, its value might be far from the desired *minimum*, a network showing this kind of behaviour is labeled in this context a bad **generaliser** and, it is towards the achievement of good generalisers that most learning algorithms are designed at this time.

From the above, it is possible to discern that the optimal learning algorithm, is the one with which, using the smallest amount of time and memory resources, it is possible to build the best generaliser. A network is a good generaliser if it achieves the smallest error when working on data not processed during training, smallest relative to the error computed by other similar networks designed for the same classification problem but, built by other learning algorithms of the same type. The difficulty of providing this sort of algorithm still concerns this research area and, the way in which it has been tackled since the appearance of the network model and,

an outline of the most popular solutions is the topic of discussion in the following sections.

2.3 Supervised Learning vs. Generative Supervised Learning

The sort of learning that takes place when training a multi-layer perceptron network, as mentioned previously, can be expressed as a minimisation problem over a many dimensional parameter space, namely the vector space spanned by the weights. A typical technique to perform this kind of optimisation is gradient descent and it is the principle on which the learning rule of backpropagation (*BP*), defined by various authors [Rumelhart *et al.* 86b], [Werbos 74] is based. Many variants of the first *BP* algorithm have been defined with the aim of trying to solve the different problems that have been detected when applying it. This section includes a review of the major work developed in relation to the detection and explanation of these problems and a description of the most important variants defined, important as they provide an adequate context to this dissertation.

Any of them can be considered to be comprised of two basic elements: 1) the architecture's components, 2) the learning algorithm. The existing differences between them generates the ensuing classification:

M1 Methods requiring that the network architecture be specified before the learning algorithm can operate on the training set in search of the appropriate weight values for the network's connections. *BP* belongs to this class.

M2 This type of method is comprised of two modules: one responsible for adjusting the network's architecture (units and/or connections) in view of the actual network performance during training, the other is comprised of the learning algorithm that effectively controls the process of traversing the training set and updating the weight values. Both functions can be viewed as a whole, *i.e.* a special type of learning algorithm, in which the set of actions that perform both the training and architecture adjustment constitute one global task. Alternatively, some methods consist of compact modules, each having a special function, these are subsequently integrated into one highly structured learning system in which it is relatively easy to discern the steps that make up the modules.

Both of the previous categories can be further classified according to the following criteria:

1. The amount of information that must be specified before the learning algorithm is executed. This refers to the number of parameters requiring an adequate initialisation to

ensure a successful network design. Each learning algorithm has its particular set of parameters and includes the rules for controlling their manipulation but, as there is a lot of overlap on these two elements, across all the existing learning alternatives, it is possible to build a global set admitting all the existing variants, or at least the ones found in the literature reviewed. A description of this global set of parameters and an analysis of the effectiveness of this type of design method is included in the subsequent section. Given that **M1** and **M2**, both comprise a learning algorithm, this classification criteria applies to both of them. Examples of this particular type of learning algorithm are quickpropagation (*QP*) described in [Fahlman 88] and rprop [Riedmiller 94].

2. The design methods of class **M2** can be further categorised depending on the rules governing the evolution process of the network architecture. Some start with a minimal network, the number of units comprising the initial model are restricted to those required for the input/output layer which are exclusively dependent on the dimensions of the problem domains; a more complex architecture is evolved (constructed) depending on the outcome of the training process, the ease with which a particular architecture recognises the elements comprising the training set. The other type of criteria requires the definition of a large network and a pruning process is what basically takes place, given that too many hidden units and/or layers have been defined the redundant ones are detected and deleted, finally producing an adequate architecture. A detailed discussion of the **M2** type methods is included in the second part of this section.

As it is easier to explain the behaviour shown by applications developed using the multi-layer perceptron with practical examples, part of the discussions concerning the relevant learning algorithm will include different executions performed on the continuous parity task. This is a classification problem with an input domain comprised of n-dimension continuous patterns, of the form described in equation 2.1, such that the x_j^i that constitute a particular pattern X^i all correspond to real numbers in the interval $(0, 1)$. Pattern i is assigned to one of two classes depending on whether the total number of its components satisfying the requirement, $x_j^i > 0.5$, is even or odd following a scheme similar to the one that governs the n-parity binary domain. The details of the exact procedure used to generate training and test sets for this type of problem are included in appendix A. The continuous parity problem is relevant in this area given its background which originates in the **binary XOR** problem, the example used by Minsky and Papert [1969] to outline the limitations of the perceptron. In an initial stage of development in network research, the important benchmark was the **binary parity** problem because it is the generalisation to a n-dimension ($n > 2$) space of the **XOR** problem. This situation made it the most appropriate example for showing the effectiveness of a training

algorithm given that at the time, [McCulloch & Pitts 43], it had already been proved that a multi-layer perceptron with one layer of hidden units could represent any boolean function, if enough hidden units were defined, an amount which happens to grow exponentially with the number of inputs and is therefore not very practical [Hertz *et al.* 91]. The situation provoked a line of research in the search for efficient network solutions, *i.e.* architectures for representing boolean functions which grew only polynomially in the number of inputs, the Network Efficiently Representable Functions (**NERF**) functions [Denker *et al.* 87]. This class has still not been clearly specified but, as it seems to be the case that it is hard to find a useful boolean function that is not **NERF**, it becomes more necessary to define mechanisms for building efficient network architectures.

In the context of the **NERF** functions the **binary parity** class is specially interesting because the rule by which each of its input patterns is associated to a particular class, generates the most complex space a network must operate on given that the amount of class separability is minimal. This constitutes the main reason for the popularity this type of problem had as benchmark, for measuring the level of performance of a particular learning algorithm on problems defined in n-dimensional binary domains.

An important limitation when aiming for the achievement of efficient network representations lies in the fact that the minimal, *i.e.* the smallest, architecture required to represent the elements of a training set does not necessarily correspond to the best generaliser therefore, a compromise must be reached at the time of defining a learning algorithm regarding these properties, [Fahlman & Lebiere 90], [Hertz *et al.* 91], [Bishop 95]. It constitutes the core in Occam's razor principle, [Bishop 95], which is concerned with modeling complexity and reflects the fact that the best fit to the training data in a *maximum* likelihood setting represent over-flexible, over-complex models which do not give the best generalisation.

If the most important inputs to the relevant learning process correspond to binary-valued domains it makes sense to aim for a method that will guarantee the generation of efficient network architectures because, for this type of domain, assuming noise-free inputs, all its elements can be generated and included in the training set, therefore the level of generalisation can be relinquished as it does not make sense to consider the possibility of having to process data not available at training time. On the other hand, if the most important problem domains are continuous-valued, given the impossibility of enumerating all the elements of a particular problem, it is more pertinent to define a learning algorithm that will guarantee the generation of better network generalisers because, it is certain that any network built with it, will have to handle patterns not available at the time the architecture was being designed.

Given that, the most interesting problems in this research correspond to classification tasks

set in continuous-valued domains, when analysing the difficulties encountered when building a multi-layer perceptron to represent the domain of any of such problems, those associated to the design of networks showing a good generalisation level become particularly important. For this reason, even though the n-dimensional binary parity problem has been so relevant to researchers in the neural network area, the n-dimension continuous parity was regarded as more pertinent for discussing the issues related to the definition of a learning algorithm that will guarantee the generation of reasonable network generalisers.

It is worth noting that, although the rule used to classify the elements of the n-parity continuous problem corresponds to the one used to create instances of the n-parity problem in the binary domain, it does not produce the same sort of complex input structures for the continuous case and, in general, there are other much more interesting problems, *eg.* the 2-spiral, [Hwang *et al.* 94]. This is evidenced in the plots presented in figure 2.7. The one contained in figure 2.7.a corresponds to a sample of a 2-continuous parity problem, *i.e.* the continuous **XOR**, and it is clear from the manner the points belonging to different classes are placed with respect to each other, *i.e.* the degree and structure of class intersection, that the task of designing a network for this case is much simpler than it would be for either, its binary counter part or, the 2-spiral problem. Nevertheless, the specific architecture will require a layer of hidden units which, depending on how closely the clusters are placed, will have to be comprised of a larger or smaller number of them, thereby providing a useful framework for discussing the points considered in sections 2.4 and 2.5.

2.3.1 The Backpropagation algorithm

In a multi-layer perceptron network each unit i computes an activation σ_i with respect to the input it receives and the weights of the connections through which it receives this input, complying with the mathematical expressions of equations 2.20, 2.21. As the training set has the form described in equation 2.2.1, performing *BP* consists in presenting each input pattern $\mathbf{X^p}$ to the network, propagating the information *forward* through all the layers of the architecture by means of the calculation of the activation of the different units comprising it and, once the last output of the units in the *output* layer is obtained, the outcome of each is used to compute the error function. This last usually takes the form of the *SSE* specified in equation 6.3 and provides a measure of the level of *learning* achieved and the amount by which the strength of each of the connections should be updated so as to improve the network's performance. This process is repeated for each of the elements of the training set and various times over the complete set until the learning goal is attained, *i.e.* the *global minimum* of the cost function is reached. To this end the weights in the network are changed along a search

direction $d(t)$, driving the weights in the direction of the estimated *minimum*:

$$\Delta w_{ij}(t) = -\eta\, d(t) \tag{2.22}$$

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t) \tag{2.23}$$

where the learning parameter η scales the size of the weight step and the variable t indexes the number of times the training set has been presented to the network. To determine the search direction $d(t)$, first order derivative information, namely the gradient $\nabla E = \frac{\partial E}{\partial w_{ij}}$, is commonly used following the principle of gradient descent. The final weight update is a scaled step in the opposite direction of the gradient which means multiplying the negative derivative by a constant value, the learning rate η. Mathematically this corresponds to:

$$\Delta w(t) = -\eta \nabla E(t) \tag{2.24}$$

which, for a single weight becomes:

$$\Delta w_{ij}(t) = -\eta \frac{\partial E}{\partial w_{ij}}(t) \tag{2.25}$$

This extremely simple mathematical model is described in great detail in various references, [Hertz *et al.* 91], [Bishop 95], [Schiffmann *et al.* 93]. It was considered a very attractive learning process when it first appeared mainly because it did not involve much pre-processing of the associated elements, eg. the architecture, the number of units and/or connections, before it was put into operation. Precisely because of its inherent simplicity, it does not provide the means to convey information of a particular problem, reducing enormously the control that can be exerted on the training process and consequently the means by which to guarantee a good performance independent of the problem in question. For example, consider the information provided by the derivative term, $\frac{\partial E}{\partial w_{ij}}$, which corresponds to the shape of the error surface that is traversed during training and which varies from problem to problem. Given the arguments associated with a particular instance of the training process, no information is gained at each step on the general outlook of the error surface because it is very localised, it usually represents just one point in time, eg. one run over the training set, and the derivative computed at that point, $\frac{\partial E}{\partial w_{ij}}$, does not provide enough information to guarantee that the process is in effect traversing the error surface towards the *global minimum* or if instead, it is stuck in one of the many *local minima* that populate it and which hampers the process of reaching the required goal. Many methods have been defined with the aim of trying to solve the difficulties encountered with *BP* and the remainder of the section discusses the ones that were considered the most relevant, remarking on the author(s) and the main problem being tackled in each of them.

Finally, there is an important assumption underlying the process of gradient descent on which *BP* is based, *i.e.* that the network architecture on which it is operating is comprised of the

necessary terms (units and connections) required to capture the form of the mapping subsumed in the training set. If such is not the case, this particular training algorithm will never reach the cost function's *global minimum* due to the restricted representation capacity, which means that the most it can achieve is a *local minimum*. Therefore, it is not surprising that *BP* is successful only for certain problems as it does not incorporate means for deciding on the size of the network architecture on which it should subsequently operate.

2.3.2 M1 Learning Algorithms: *BP* variants

The behaviour shown by a particular instance of the procedure for finding a weight vector that minimises an error function, which is depicted in equation 2.22, varies with the problem on which it is applied and is definitely shaped by the manner in which the term $\Delta w_{ij}(t)$ is specified. For gradient descent it takes the general form of equation 2.23 which is, as already explained, the basis of *BP*. Other methods will compute $\Delta w_{ij}(t)$ using other elements but being optimisation alternatives they conform to the requirements a network training algorithm should satisfy and is the reason why some of them have been applied to this area.

The different descriptions included in this section, granted that they correspond to learning algorithms operating on networks of fixed architectures, will be realised assuming that the training process is working on a sufficiently complex network for the problem to which it is being applied.

The various minimisation methods that have been used in the definition of a learning algorithm for training networks can operate in one of two modes which, result in two types of process:

(a) This class consists of minimisation techniques that start at a small-valued point of the error surface and, assume that the region containing the global *minimum* can be reached by gradient descent, therefore it is only necessary to continuously descend until it has been reached. The methods that proceed under this assumption, have the restriction that if they start operating, or fall into, a *local minimum* there are no means of escaping it, [Bishop 95]. Learning algorithms defined according to conjugate gradient and quasi-Newton methods operate under this policy and their effectiveness is very dependent on the initial working conditions, given the underlying assumption of the optimisation policy on which they are based.

(b) This second group incorporates a stochastic component in the search for the global *minimum* because it might be the case that the search starts in the wrong region and it is necessary to jump out of it and proceed in another section of the error surface. Gradient descent is this type of minimisation process and, has the advantage over methods such

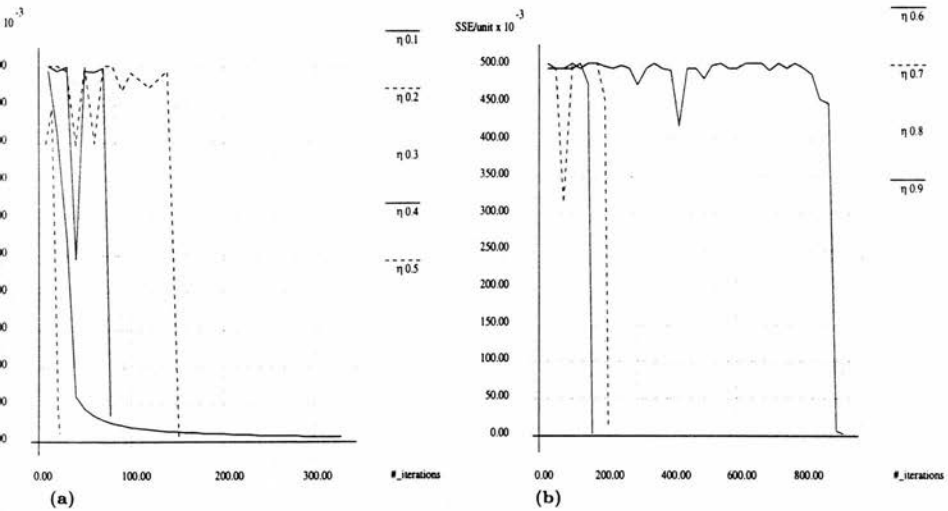


Figure 2.3: **Error curves for different learning rates,** *2-continuous-PARITY*: **(a)** η **in (0.1,0.5), (b)** η **in (0.6,0.9)**

as conjugate gradient of being more immune to the initial conditions under which the search process starts operating.

An additional advantage the type (b) methods have, *eg* gradient descent, over the other type of optimisation techniques is, that their computation is faster and simpler because, methods like conjugate gradient use high order information, such as the second derivative of the error function, which involve computationally intensive calculations.

Given the theoretical framework underlying *BP* there are two possible ways in which it can be modified so as to obtain new learning procedures: using other optimisation methods to compute $\Delta w_{ij}(t)$, or varying the steps of gradient descent employed within *BP*. The ensuing analysis is divided according to this criteria and considers in the first part the implementations that resulted from upgrading gradient descent and in its second part the alternatives obtained from the application of the information provided by other optimisation methods.

Enhancing gradient descent

The general form of gradient descent complies with the expressions shown in equations 2.24 and 2.25 and, as the manipulation of any of the elements included therein might result in an alternative manner of processing the weights of a network architecture, it has been the way

learning rate	*SSE/u* on test	*% correct*
0.1	0.004	100
0.3	0.002	100
0.6	0.002	100
0.9	0.002	100

Table 2.1: Effect on generalisation rate of using different η

in which much of the research aimed at improving *BP* applied to multi-layer perceptrons has been conducted. The resulting versions fall into one of the following categories:

1. Update policy: given that the error has to be computed for every element of the training set, there is a $\Delta w_{ij}(t)$ value for each of them which means that $w_{ij}(t)$ can be updated in two different ways, after the error has been computed for the whole training set, *i.e.* periodic update, or do it after the network computes an output for all the patterns in the training set, *i.e.* continuous update. In most of the existing implementations of *BP* and its variants it is possible to select the update policy and, although there is some difference in the manner the process traverses the error surface for each case, the experiments performed showed that it was not significant when compared against the effect the variation on some of the other parameters have on the whole training, eg. using different values for η. Consequently, as the results obtained by using the periodic alternative were better, this was the policy applied in the algorithm defined in this research.

2. The manipulation of the learning rate η: choosing a suitable value for this term in gradient descent is important to guarantee the convergence of the whole process. Different ways for its manipulation have been defined, given how much it influences the overall performance, as the error curves depicted in figure 2.3 show. The possibility of producing such varying forms of the whole training process by simple adjustments to η, induced various researchers to define many *BP* variants centered in the manipulation of this parameter because it seemed to be an efficacious way for avoiding local *minima* and accelerating the training procedure.

3. Extending the model: adding new terms to the general form, that will allow advancing towards the *minimum* in alternative ways to the ones commanded by the derivative term. For example, using a *momentum* term and incorporating it, to the term computed in equation 2.25.

Detailed information on algorithms defined for improving *BP* through an appropriate treatment of the η parameter can be found in much of the literature on multi-layer perceptrons

[Schiffmann *et al.* 93], [Riedmiller 94], [Hertz *et al.* 91] contain good summaries of the most important alternatives. An adequate manipulation of η results in a considerable improvement in the convergence time of the algorithm as a small η produces long convergence in a flat error surface, whereas a large η might possibly lead to oscillations, preventing the error from falling below a certain value [Hertz *et al.* 91]. The plots of figure 2.3 which include the error curves generated during training of a network for a **2-continuous-parity** problem give a good idea of the effect η has on the learning process; each curve corresponds to a particular η, the plot included in part (a) of figure 2.3 contains the curves for $\eta \in (0.1, 0.5)$, whilst part (b) includes the curves for $\eta \in (0.6, 0.9)$. This particular problem required a network which achieved complete recognition of the training set in a reasonable number of iterations in all of the training processes executed, one for each of the η considered, but there is a clear difference in the shape and length of the error curve associated with each of the executions.

The errors plotted in figure 2.3 apparently indicate that a reasonable architecture was produced given that their values reflect complete recognition of the elements included in the training set and it allows the conclusion that a well-chosen η will result in a reasonable architecture in a shorter amount of time. Nevertheless, to give more conclusive information on the importance any element of the model has on the overall design process, considering that the aim of this research was to provide a learning process that would result in better generalisers, it is necessary to register the performance of the network when presented with a test set, because it is the error computed for this set, the one that provides the measure that allows consideration of the appropriateness of the network's generalisation capacity. This means that when evaluating the importance and effectiveness of any of the parameters involved in the learning algorithm and in the network model, it is necessary to look at performance not only at the level of the training set but also to look at the response of the architecture obtained when it processes the test set. A proper selection of the parameter η certainly results in faster training, but as the contents of table 3.1 show, there is no real difference on the generalisation capacity achieved for the different learning rates employed in this example.

The interesting implementations of this particular type of *BP* variant for the most part operate by checking whether a particular weight update ($\Delta w_{ij}(t)$) did actually decrease the cost function. If it didn't then the process overshot and η should be reduced. On the other hand if several steps in a row have decreased the cost, then it might be advantageous to increase η. They differ from each other in the strategy employed for varying η, as is evident in each of the descriptions found in [Cater 87], [Franzini 87], [Vogl *et al.* 88].

The third type of *BP* variant for the most part is comprised of processes which aim at faster training through the manipulation of the derivative term, *i.e.* $\frac{\partial E}{\partial w_{ij}}$ so that its overall effect in

the $\Delta w_{ij}(t)$ will result in a more reliable training process. In this category falls one of the oldest and most frequently used which is the process that adds a momentum term to $w_{ij}(t)$ each time it is updated so that its value will tend to change in the direction of the *average* downhill force, and training in general will prove to be more stable [Hertz *et al.* 91]. The equation for $\Delta w_{ij}(t)$ with this additional term becomes:

$$\Delta w_{ij}(t) = -\eta \frac{\partial E}{\partial w_{ij}}(t) + \mu \Delta w_{ij}(t-1) \tag{2.26}$$

where the μ parameter scales the influence of the previous weight-step on the current one. As with the η parameter variants of *BP* have been developed based in the manipulation of the μ term. Some include dynamic adaptation of both η and μ as training progresses and they correspond to the ones showing the best overall training performance. The most successful implementations of this type of algorithm are:

1. **Rpropagation**: resilient propagation is described in [Riedmiller 94] and is an efficient learning scheme that performs a direct adaptation of the weight step based on local gradient information. In different analyses that compare the effectiveness of various *BP* alternatives when training networks operating on the same problem, this particular training process has provided the best set of results [Schiffmann *et al.* 93], [Tveter 96]. It operates by adjusting the weight values by a fixed amount depending on the sign of the derivatives at the present iteration and the previous iteration because this way it is possible to conclude whether there has been a favourable change in the manner the error curve is being traversed. Considering that various examples realised in this research showed the effectiveness of this particular type of training, it was incorporated as the training component in the network design method defined in this study. Details on the implementation that is used are included in the following section.

2. **Quickpropagation**: the algorithm was designed by Fahlman [1988] and treats the weight values as quasi-independent terms. The idea is to approximate the error surface as a function of each of the weights by a quadratic polynomial, *i.e.* a parabola, and then to use two successive evaluations of the error function, and an evaluation of its gradient to determine the coefficients of the polynomial. At the next step of the iteration the weight parameter is moved to the *minimum* of the parabola assuming that the result of the local quadratic fit gives a parabola with a *minimum*. If instead it leads to a parabola with a *maximum* or, if it is nearly flat, special operations are incorporated to ensure a correct traversal towards the error's global *minimum*.

3. **Delta-bar-delta**: this algorithm is described in [Jacobs 88] and controls the learning rates by observing the sign changes of an exponential averaged gradient. Different η

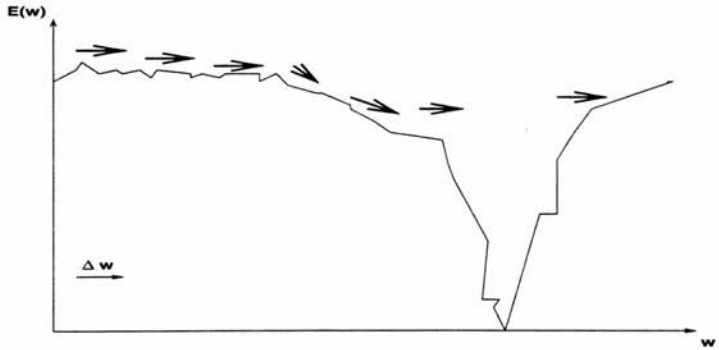


Figure 2.4: **Problem of gradient descent: dependence on shape of the error surface**

are manipulated at each execution and it requires the definition of further appropriate parameters in addition to the initial η term.

The aim in the existing *BP* variants, which were created through the modification of the terms associated to the gradient descent process, has been to eliminate the counter intuitive influence of the partial derivative on the size of the weight step. If the error function is shallow, the derivative is quite small, resulting in a small weight step. On the other hand, in the presence of steep ravines in the error surface, large derivatives lead to large weight steps, possibly taking the algorithm to a completely different region of weight space as can be seen in figure 2.4 [Hertz *et al.* 91]. The most successful of them, *i.e.* quickpropagation, rpropagation, all aim at generating additional information during the execution of a training cycle that will allow them to detect these extreme cases so that they may be treated appropriately. Even though the processing involves the manipulation of η and μ, the main objective is to override the inadequate behaviour sometimes produced by the way the derivative term evolves during the optimisation process.

Using other optimisation methods

The cost (error) function $E(W)$ that is minimised in the search for the required vector of weights W can be expanded about an initial W_0 and rewritten in the following way [Hertz *et al.* 91]:

$$E(W) = E_0 + (W - W_0)^t \otimes \nabla E(W_0) + \frac{1}{2}(W - W_0)^t \otimes \mathbf{H} \otimes (W - W_0) + \cdots \qquad (2.27)$$

where ∇E is simply the vector with components $\frac{\partial E}{\partial w_{ij}}$ and, $\mathbf{H}$ is the second derivative **Hessian matrix**:

$$H_{ij} = \frac{\partial^2 E}{\partial W_i \partial W_j} \qquad (2.28)$$

evaluated at W_0. Differentiating the term in equation 2.27 results in a similar expansion for the gradient:

$$\nabla E(W) = \nabla E(W_0) + (W - W_0)^t \otimes \mathbf{H} + \cdots \tag{2.29}$$

The desired *minimum* of $E(W)$ is $\nabla E(W_0) = 0$. The most relevant optimisation methods applied to network generation differ from each other in the approach taken to reach this *minimum*:

(a) newton's method: sets the term in equation 2.29 to zero ignoring the higher order terms which means that the expression to optimise has the following form:

$$\nabla E(W_0) + \mathbf{H} \otimes (W - W_0) = 0$$

alternatively:

$$\mathbf{W} = W_0 - \mathbf{H}^{-1} \otimes \nabla E(W_0)$$

The method does provide a precise expression for calculating the desired *optimum* but the computational costs involved, because of the mathematical expressions that comprise it, has been the reason for its lack of use, although it served as a basis for the definition of other, cheaper training algorithms.

(b) line search: uses only first derivative information, combined with line searches along selected directions. The philosophy underlying a line search starting from W_0 in direction $\mathbf{d}$, involves staying on the line:

$$\mathbf{W} = \mathbf{W}_0 + \lambda \mathbf{d}$$

and choosing λ to minimise $E(\mathbf{W})$. There are many ways of performing this one-dimension minimisation. The different ways in which it has been specified has resulted in various alternative learning algorithms for multi-layer perceptrons [Hertz *et al.* 91].

(c) steepest descent/conjugate gradient: corresponds to the simplest approach using line search in which $\mathbf{d} = -\nabla E(W_0)$. Thus the minimisation takes place along a line in the gradient direction, which is revaluated at each step until the desired *minimum* is reached. The conjugate gradient method resulted from a modification to the term comprising the steepest descent which was modified considering that a better approach was to use a compromise between the gradient direction and the previous search direction:

$$\mathbf{d}^{\mathbf{new}} = -\nabla E^{new} + \beta \mathbf{d}^{\mathbf{old}}$$

for some appropriate β which is chosen so that each new search direction spoils as little as possible the minimisation achieved by the previous one.

One implementation based on these ideas that combines the information of two of the optimisation methods mentioned, was developed by Leonard and Kramer [1990]. It uses conjugate

gradient to calculate an exponential averaged gradient, and a line search process is used to generate the most appropriate value for the learning rate, *i.e.* one that will minimise the error with respect to the actual search direction. Its performance, which is compared against that of the *BP* variants described in section 2.3.2 in [Schiffmann *et al.* 93], is quite disappointing and explains why it has not been greatly exploited.

Further results on the application of conjugate gradient for training multi-layer perceptron networks can be found in [Watrous 87], [Webb & Lowe 88], [Kramer *et al.* 89]. The main drawback many of them have is the type of information they require which, for the most part, is generated by very computer intensive processes. The methods previously described show how part of the difficulty has been solved by not computing the whole block of information theoretically demanded and using instead approximations which, even though they are, at least from the optimisation point of view, more effective than gradient descent, are still unsatisfactory in the network context because they are difficult to use and do not solve the performance dependency on the network's initial conditions this group of methods have.

The main handicap of all the learning algorithms described up to this point, stems from the fact that the definition of a sufficiently large architecture in a previous step is mandatory, because if it turns out to be the case that the architecture that is being processed has limited representation capacities then none of the *BP* variants mentioned will produce an effective network, at least not in terms of a good generaliser.

2.3.3 Resilient propagation (rprop)

In the discussion, this particular *BP* variant receives special attention because it is the training component incorporated to the network design method defined in this research. It was developed by Martin Riedmiller and is discussed in detail in [Riedmiller & Braun 93].

It is a local adaptive learning scheme, that performs supervised periodic learning in multi-layer perceptrons. The basic principle of **rprop** is to eliminate the harmful influence of the size of the partial derivative on the weight step. This is done by only taking into account the sign of the derivative in the procedure responsible for performing the weight update of each of the network's connections. The exact amount by which each of the weights is modified is determined by a weight-specific update-value $\Delta_{ij}(t)$:

$$\Delta w_{ij}(t) = \begin{cases} -\Delta_{ij}(t) & : \quad \frac{\partial E}{\partial w_{ij}}(t) > 0 \\ +\Delta_{ij}(t) & : \quad \frac{\partial E}{\partial w_{ij}}(t) < 0 \\ 0 & : \quad otherwise \end{cases} \tag{2.30}$$

The second step in this learning process consists in determining the new update-values Δ_{ij}.

This is based on a sign-dependent adaptation process similar to the one employed in other learning methods, *eg* quickpropagation. It evaluates how the term $\frac{\partial E}{\partial w_{ij}}(t)$ evolves during the minimisation process for deciding on how much it should increase/decrease the update-value:

$$\Delta_{ij}(t) = \begin{cases} \eta^+ \Delta_{ij}(t-1) & : \frac{\partial E}{\partial w_{ij}}(t-1) \frac{\partial E}{\partial w_{ij}}(t) > 0 \\ \eta^- \Delta_{ij}(t-1) & : \frac{\partial E}{\partial w_{ij}}(t-1) \frac{\partial E}{\partial w_{ij}}(t) < 0 \\ \Delta_{ij}(t-1) & : otherwise \end{cases} \tag{2.31}$$

where $0 < \eta^- < 1 < \eta^+$.

The terms η^-, η^+, correspond respectively, to a decrease and increase factor. They were incorporated to the algorithm with set, fixed values to reduce the number of freely adjustable parameters which often lead to a tedious search in parameter space. The factors used in this implementation were chosen following the recommendations in the material reviewed which discussed the performances achieved by using this algorithm, [Schiffmann *et al.* 93], [Riedmiller & Braun 93], [Tveter 96]. This information also determined the values for the initial $\Delta_{ij}(0)$, *i.e.*, $\Delta_0 = 0.01$.

This particular choice was made based on the results achieved on a set of examples which concerned a comparison of different learning processes. It showed that the best performance in the majority of the problems was achieved using **rprop** and, as the conclusion was reinforced by other studies of a similar kind, [Schiffmann *et al.* 93], [Prechelt 94], it seemed an appropriate choice.

2.3.4 M2 Learning Algorithms: Generative Learning

This type of algorithm has shown a higher rate of success in the task of designing an adequate network for a particular application, because it includes additional techniques for subtracting more information about the problem that has to be solved. This additional data is used to guide the training algorithm and is also applied to the task of tailoring the network architecture such a process is acting upon, so that the requirements that must be met according to the information that has been subtracted from the domain are satisfied.

The taxonomy outlined in Figure 2.5, defined and discussed in [nana & Ross 96], clearly reflects the existence of a wide variety of approaches aimed at improving the design of a multi-layer perceptron network through the manipulation of the architecture, in conjunction with a series of techniques aimed at improving the optimisation performed during the training phase of a particular network instance.

Most of the methods depicted in the taxonomy use information of a statistical nature about the domain from which the training and test sets used for building the network have been extracted,

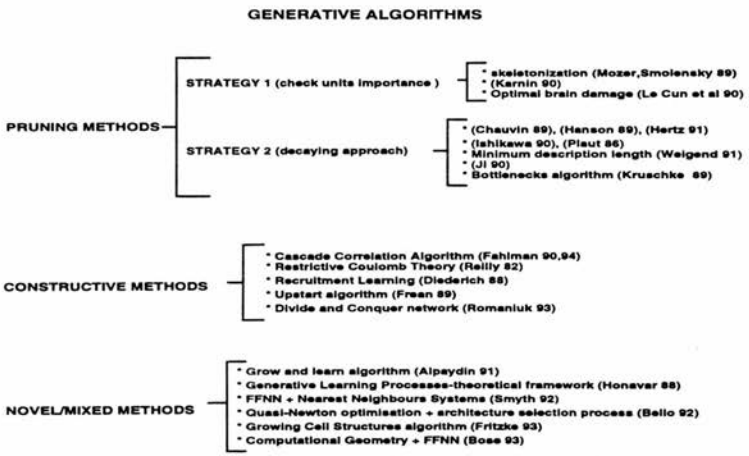


Figure 2.5: **Taxonomy of Generative (dynamic) Supervised Learning Algorithms**

to decide on the appropriate number of units and/or layers that should comprise the network architecture. In the introduction of section 2.3, two types of **M2** methods were defined: the constructive-generative algorithms, and the pruning-generative algorithms, [Reed 93], incorporates an excellent discussion of this last type of generative method. It is possible to consider the existence of a third group, as the taxonomy of figure 2.5 shows, the mixed methods which make use of pruning and constructive techniques and try to exploit the advantages of both philosophies.

The strategy that is common to all of them is concerned with adjusting the network architecture so as to enhance the overall learning capacity, a major improvement when compared against the previously *ad-hoc* manner in which a network architecture had to be specified for the **M1** methods. The most popular example of this type of algorithm is cascade correlation, developed by Fahlman and Lebiere [1990]. It is applicable to problems with continuous output variables and is based on networks of sigmoidal hidden units. To begin with there are no hidden units and every input is connected to every output unit by adjustable connections. The output units may be linear or may have sigmoidal non-linearities depending on the application. At this stage the network can be trained like a normal perceptron by applying one of the learning algorithms described in the previous section. Fahlman and Lebiere use the quickpropagation algorithm, for tuning the network's weights for a period of time, *i.e.* until a user-defined parameter satisfies some requirement and then a sigmoidal hidden unit is added to the network. This is

followed by further training, alternating with the addition of hidden units, until a sufficiently small error is achieved. When a new hidden unit is incorporated to the network it takes inputs from all of the input units and the outputs of all the existing hidden units. This organisation enables the adaptation of the connections in two phases. In the first one, only the connections going into the new hidden unit are updated and, once the effect of adding the new unit has propagated over the complete set of previously existing hidden units, the second phase, which consists in the adjustment of the connections that go from the hidden units to the outputs, including those of the new hidden unit, is realised. This manner of adding units to the network creates a cascade architecture and results in many hidden layers, one of the objections against the process, surmounted in a later stage of development with the definition of a flattening process described in detail in [Fahlman & Baluja 94], which consists in compressing as many of the layers generated in the first part of the building process as is possible without deteriorating the generalisation ability of the network being developed. In the growing stage of the cascade correlation algorithm after adding a hidden unit to the architecture, the weights of the connections to it are computed through the maximisation of the correlation between the output of the unit and the residual error of the network's outputs prior to its addition so that the new unit might be guided towards the correct identification of the patterns that were not properly classified by the previous smaller network.

Cascade correlation is an example of a constructive learning algorithm. It begins operating on a minimal network, one comprised only of the input and output units, and subsequently proceeds to enlarge it until an architecture that is evaluated as providing a satisfactory representation of the application domain is found. The problem with this type of approach lies in the uncertainty associated with each intermediate step. The process for deciding on the incorporation of a hidden unit and/or layer to the network is made in a rather ambivalent manner because, the two main causes that induce an inadequate architecture are reflected in the same kind of erratic network behaviour and, each of them must be dealt with in a different way. These causes are: **(a)** too simple an architecture, *i.e.* not enough hidden units or, **(b)** the weight vectors associated to a particular network instance are not situated in the region of the input space that will produce a correct classification of the training data. Both circumstances produce an inappropriate network something that is evidenced by **saturated** hidden units resulting from the type of activation function used, *i.e.* the *sigmoid*. In a learning environment, it indicates that the algorithm is stuck in a *local minimum* which, depending on whether it has been brought about by cause **(a)** or cause **(b)**, as defined above, should be treated in a different manner: the first by incorporating more hidden units, the other by shifting the weight vectors to another region of the problem domain. At present, as it is not possible to locate precisely the reasons for an inadequate network while it is being grown, it is hard to decide whether the

better way to proceed so as to successfully complete the network design is by, adding a unit or, moving the existing weight vectors. This uncertainty reduces the effectiveness of this type of learning algorithm.

Given that the aim of a training algorithm should be to produce as simple an architecture as is required in a reasonable amount of time and using an appropriate computational effort, it is sensible to consider there is a good possibility of designing a learning process that will be more efficient than cascade correlation in terms of the final architecture achieved and the amount of time employed to generate the adequate simpler network. A point that is proved by the enhanced cascade correlation algorithm described in [Fahlman & Baluja 94]. In the first place it is a method designed to simplify the kind of architecture generated by a cascade correlation process which means that, in terms of network architecture, the original cascade algorithm can be considered inefficient. In the second place the additional phase incorporated can be viewed as a type of pruning algorithm because it proceeds by reducing the architecture to a simpler network. Therefore if it is possible to find cheap means for building an initial maximal architecture, it cannot be denied that the pruning algorithm will be a more reliable and simpler means to design an adequate network.

The second type of algorithm which starts with a maximal architecture and proceeds by pruning the superfluous units has the inconvenience of requiring an initial good network, one that is able to recognise all the elements of the training set and in which it might be possible to measure the generalisation capacity so as to proceed with a pruning process mainly aimed at improving this property of the network. In the literature reviewed there were few practical examples showing the effective performance of a pruning algorithm, most of them included a detailed theoretical framework for such an algorithm but there were no table of results which might explain the reason why the literature seems to look more favourably on the constructive type of learning. It is less assumption laden, *eg.* size of maximal architecture does not need to be assumed. Nevertheless, if it is possible to find a method for building an initial maximal architecture then the following stages of the network design become much simpler because detecting saturated hidden units, the inability to properly train a particular architecture, or a deterioration in the generalisation capacity of the network, will simply mean that there are too few hidden units in the architecture because the kind of uncertainty that exists with the growing algorithm is not an issue in this case.

Further details on the assessment of the different strategies defined for the development of applications using the multi-layer perceptron model, that aim at improving the overall design process by manipulating the network architecture and, the learning procedure as such, can be found in Appendix B.

2.4 Hidden Units: the role they play in a multi-layer perceptron

The manner in which the research in multi-layer perceptrons has evolved shows the importance the hidden unit component has in the model, given that the most effective solutions that have been found to this moment have been computed through the application of design methods involving a strong element of architecture manipulation, *i.e.* hidden units addition/subtraction.

The representation capacity of a network with hidden units depends on the number of them that comprise the architecture, the number of layers in which they are organised and, on the type of unit employed. This last is determined by the activation function related to it which, if it happens to be non-linear, enhances the representation capacity of the model in the same way that a non-linear regression model is a more powerful statistical tool than a linear regression one [Sarle 94]. The curves of the kind of activation functions mostly used for the multi-layer perceptron are shown in figure 2.6, plot 2.6.a contains the linear function, 2.6.b corresponds to the threshold, plots 2.6.c, and 2.6.d are the non-linear functions usually employed with the model, plot 2.6.c is the sigmoid and 2.6.d the hyperbolic tangent. An additional reason for the wider applicability of the non-linear networks, in particular the sigmoid network, is induced by the fact that all of the other activation functions can be expressed in terms of the sigmoid.

When performing a modelling task with a multi-layer perceptron, apart from having to decide on the number of units, it is necessary to determine the number of layers that will comprise the architecture. This does not have to be a major issue if it were not for the need for generalisation given that it has already been proved, [Cybenko 89], [Lippmann 87], [Lapedes & Farber 88] that a network with two hidden layers of units can approximate any function to an arbitrary degree of accuracy and, one is enough to represent any continuous function, [Cybenko 88]. This means that providing the learning algorithm with the mechanisms to decide upon the creation of at most two hidden layers should be sufficient to tackle any problem. Furthermore, given that one hidden layer is enough to represent any continuous function and that there are many real-world problems that are satisfactorily simulated with a continuous function, the whole network design process can be built around the idea of creating one-hidden layer networks without loss of generality and adequately exploiting the capacities of the model. The existence of various other frameworks, *i.e.* mathematical, statistical, etc, that might be used for simulating purposes, some comprised of a greater representation capacity than the multi-layer perceptron provide additional good reasons for restricting the design of a network to just one layer of hidden units.

In the context of the one-hidden layer perceptron, it is useful to be able to measure how many

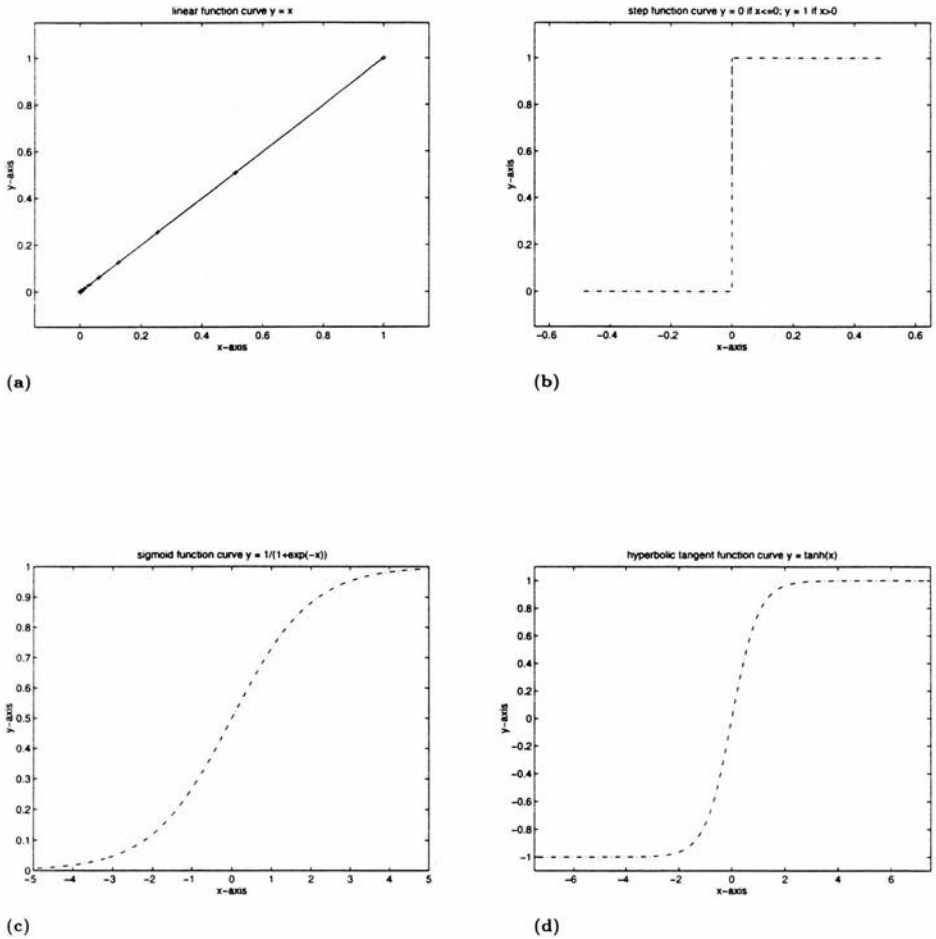


Figure 2.6: **Form of different activation functions: (a) Linear, (b) Step, (c) Sigmoid, (d) Hyperbolic tangent**

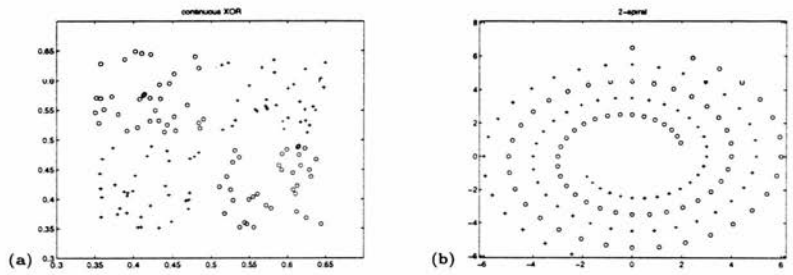


Figure 2.7: **Form of possible problem domains for a *MLP*: (a) *2-continuous-parity*, (b) 2-spiral**

hidden units are required for a particular set of patterns, *i.e.* the training set, because this gives a very good insight into the maximal architecture that can capture the form of the problem at hand. At the same time, if it is possible to evaluate how this number varies in relation to the size of the set, the information could be taken as a complexity measure. This will make it possible to decide on the appropriateness of using the model for the application under consideration, because if there is a high degree of dependency between the size of the set and the final number of hidden units required, it is possible to conclude that the possibilities of producing a good generaliser are not very high. If the size of the architecture is greatly affected by the number of elements in the training set the probability of producing a poor generaliser is greater because, a need for more hidden units strongly conditioned by the size of the training set implies that it is relatively simple to produce a test set on which the network will show poor performance.

If the analysis of an application's domain indicates that it is extremely difficult to produce a good network generaliser, there are grounds for looking for an alternative representation tool. The complexity measure computed in this type of analysis should make it possible to detect the differences between the problem domains, set in the two-dimension space, shown in figure 2.7; one of the diagrams (figure 2.7.a) shows a domain with a structure that will be easy to grasp in almost any sample extracted from it. This means there will be practically no difference in the size of the network required for completely recognising the elements of a set of such samples. By contrast, the domain whose structure is included in figure 2.7.b corresponds to a hard problem to solve because, performing the same type of analysis, there will be a lot of variance in the size of the architecture required by a series of samples extracted from the domain. This result is induced by the difficulty there is in capturing the relative disposition of the points belonging to the different classes associated to it, the number of hidden units required for a good network will vary greatly with the contents of the training set utilised to build it. Such

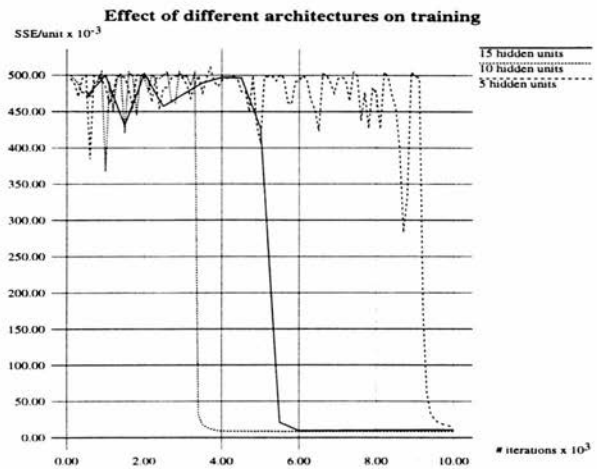


Figure 2.8: **Error curves for different architectures** *2-continuous-parity* **problem**

is the type of analysis that can be generated with the complexity measure whose computation and, information content will be discussed in greater detail in a subsequent chapter.

Figure 2.8 contains the *error curves* of three training processes performed on three different network architectures designed to solve the same problem. The training and test sets are comprised of instances of a *2-continuous parity* problem. The data used to build them is shown in figure 2.7.a and, corresponds to an output of the program described in Appendix A. Each curve has a particular form and length even though all of the processes employed the same training set and the same learning algorithm. The difference is produced by the type of network architecture used to solve the problem; all three were built using the multi-layer perceptron but each contains a different number of units in its hidden layer, *2-5-1*, *2-10-1* and *2-15-1* were the specific architectures considered. The following things are worth noting about this set of results:

1. Even though for the *2-15-1* network a higher number of weights have to be continuously updated, the associated error curve does not correspond, as might have been expected, to the worst training process, therefore there is no monotonic relationship between the number of iterations required in training and the network size.

2. Table 2.2 contains two different measures calculated using the elements of the test set. These were obtained for each of the architectures that are being considered. The values in column two correspond to the percentage of the test set each network correctly classified and, the ones comprising the third column correspond to the SSE/unit. In all three cases

hidden units	*correct*	*SSE/unit*
5	99.44	0.004
10	100	0.004
15	98.87	0.004

Table 2.2: Generalisation rate for different architectures in the *2-continuous parity* problem

the SSE/unit registered with the test set was the same but the percentage of correctly classified patterns is different. The best result was obtained with the *2-10-1* architecture which also has the best *error curve*. The information contained in each row of the table, granted that it was obtained using the test set, is what in this study is known as the generalisation rate of a multi-layer perceptron network.

3. Given that the final *SSE* value for both the training and test sets obtained with all the architectures is pretty similar, it seems to be the case that at the moment of deciding on the most appropriate architecture for the application it might be convenient to also take into consideration another measure of performance, *i.e.* the percentage of patterns in the test set correctly classified, because the combined information seems to be a much better measure of the generalisation capacity of the network that resulted from the training process.

4. The different results obtained distinctly show how necessary it is to clearly specify what is understood to be a good network architecture. Is it the one achieving the complete recognition of the training and test sets, or alternatively the one with the smallest number of hidden units that correctly classifies the training set?, maybe the one using a higher number of layers because even though it has a more complex structure it required a lower number of connections?

The error curves comprising the plots in figure 2.8 and those shown in figure 2.3 can be used to compare the effect a change in architecture has on the whole learning process against the effect induced by varying the learning rate η. The error curves of all the training processes registered in the figures were developed on the same instance of the *2-continuous-parity* problem. Each of the curves shown in figure 2.3 correspond to an execution of *BP* using a particular value for η; the curves included in figure 2.8 are of training processes performed on different architectures. It is clear that the effect of changing η is much weaker than that of varying the architecture size when considering the difference in the number of iterations required to achieve complete recognition of the training set. Regarding the generalisation capacity, while it remained constant for all the processes on which the varying element was η, it is clearly affected by a variation of the size of the network, something which can be concluded by looking at the contents of tables 3.1 and 2.2.

Further proof of the importance the appropriate definition of the network's hidden units component has is evident considering how the *NN* model's applicability widened following its redefinition and extension, together with the inclusion of the learning algorithm. Nevertheless, not much research in this area has dwelt on the possibility of enhancing the design process of a network by continuously measuring the state of the *hidden* units during training and using this information in deciding on ways of improving it.

There has been work aimed at comparing the capacity of the multi-layer perceptron as a *classifier* with that of certain data analysis techniques, by looking at the function the hidden unit component has in the model and describing how it affects training. See [Gallinari *et al.* 91], [Oja *et al.* 92a], [Oja *et al.* 92a] for particular descriptions of studies of the behaviour of the network model pursuing this approach. All of them refer to the relevance of the hidden units and conclude that it plays the *role* of the *important dimensions* of the input space. It is argued that as training progresses, the weight vectors associated with each of the units defined evolve so that when training is successfully completed, each corresponds to a source of information on the space's variability. What is not very clear is the mechanism used for deciding on the initial *number* of important *dimensions*.

Numerous studies specify a number of assumptions that guarantee the existence of a proof for what is being stated; many of which are difficult to justify and which should be considered reason enough for their invalidation. For example, Baldi and his research group [1994], study the relations between discriminant analysis and a one-hidden layer perceptron used for classification tasks showing the formal equivalence between the two techniques. Their proof is developed using linear algebra. With it they express the one-hidden layer linear perceptron in terms of the multiplication of two weight matrices: $\mathbf{W_1}$ the input-hidden connection's matrix, $\mathbf{W_2}$ the hidden-output connection's matrix. Both matrices have dimensions which depend on the number of hidden units, important input space dimensions which are not rigorously treated in [Baldi & Hornik 94]. Also it is possible to generate an input space requiring more *hidden* units than the *input* space dimension and for this case it would be impossible to produce the required *weight vectors* through discriminant analysis as the process is restricted to the number of dimensions of the input space.

A final point worth mentioning is the fact that the most successful *BP* variants that have been developed are those which update the network's architecture as training proceeds. They comprise the most interesting idea in terms of network development and it is the one pursued herein. Of this set of studies the most interesting one is the cascade correlation algorithm developed by Fahlman, Lebiere [1990] described in the previous section. None of these algorithms makes direct use of the information provided by the output of the hidden units during train-

ing. What generally happens is that decisions about the number of required units is made by inspecting the *network* error on the training set or the behaviour generated by the weights of the connections related to a particular unit as training progresses. If in addition, data analysis techniques are applied to the output of the *hidden* units, a much better decision process can be designed.

2.5 Structure of the input space

The discussion to this point has stressed the fact that, of all the parameters available in the multi-layer perceptron network, the most important is the number of hidden units. The examples above show how an appropriate way of proceeding to find the ones required in a particular application is through the analysis of the structure of the problem domain, *i.e.* an evaluation of the placement of the points comprising the training set.

Theoretically, the most effective tools for realising this study are the ones defined in geometry as this mathematical science was conceived precisely for it. To be able to use them, all the data must be represented in numerical form. This condition is already satisfied because, the multi-layer perceptron is itself a model requiring a numerical input. Problems that include nominal data are pre-processed so that the nominal attributes might be replaced with numerical values that the training process might subsequently be able to manipulate.

The usual transformation procedure replaces each of the possible values a nominal attribute can take by a binary pattern, ensuring there is a 1 : 1 relation between the ones used and all the possible values that can be assigned to it. According to Prechelt [1994] this type of transformation guarantees that the unordered order relation between the different values of the nominal representation is maintained. For example, consider the following possible values that can be assigned to the nominal attribute **vegetable**:

original form: `vegetable = {pepper, cauliflower, onion}`
binary encoding: `bin_vegi = {100, 010, 001}`
continuous encoding: `cont_vegi = {0.33,0.66,1}`

The first one corresponds to those that can be found for it in the original dataset. According to [Prechelt 94] care must be taken if the data is transformed to a numerical representation because, the fact that there is no natural ordering in the nominal form should be reflected in the transformed version. For that reason it defines the binary encoding reflected in `bin_vegi` which, the author of **Proben1** considers is as unordered as the nominal representation. Never-

theless, regarding the values in `bin_vegi` as binary patterns, which after all is what they are, each has its numerical interpretation and an ordering can be automatically derived; therefore the **continuous encoding** associated with `cont_vegi` is equally valid. Both endow the nominal attribute `vegetable` with the same numerical qualities missing in its nominal form, something that is disputable considering the natural ordering of any existing object.

Using a continuous transformation has various advantages over the binary transformation. These are summarised in the following points:

(a) The available mechanisms for manipulating missing information. In the binary transformation this is usually dealt with by defining an additional pattern which the multi-layer perceptron training process can only treat in the same way as the other possible values. Considering that what it represents is missing information, it should really be interpreted in a different manner. With the continuous transformation a more legitimate manipulation has been implemented as it consists in the computation of the missing attribute of the incomplete patterns using the information provided by the available data. The type of calculation performed varies from one application to another, sometimes the missing information is replaced by the mean, at other times the median is the term employed, and there are examples that use other statistical parameters. Although, it is possible to generate the same type of terms for binary domains, as the computations involved are intrinsically continuous, their use is more reliable in the context of continuous-valued applications.

(b) The dimension of the input space. Using the binary transformation results in a much higher dimension space. This means more input units in the network architecture which never facilitates the subsequent network building procedure. Alternatively the continuous transformation does not require the definition of additional inputs.

(c) The geometric analysis is much more effective when working on continuous data because it involves various matrix operations which are error prone when it is very sparse, something that has a reasonable chance of occurring with the type of binary transformation described. If the transformation applied is the continuous one this risk is practically non-existent.

One of the datasets used to evaluate the network design method defined is the **soybean** problem. The initial data is comprised of 35 attributes, 16 of them are binary, 6 are continuous and, 13 are nominal. [Prechelt 94] describes a network computed to handle this task. In it the nominal attributes were transformed following the steps of the binary transformation which resulted in a 82-dimension space. This solution is part of the benchmark defined by Prechelt

Square error percentage and Mean error for the soybean problem							
network	iterations	sep_tr	sep_vl	sep_ts	merror_tr	merror_vl	merror_ts
GROPING 35-32-19	26	0.0083	0.4390	0.8465	0.0007	0.0060	0.0107
GROPING 82-8-19	5000	2.49741	2.85958	2.96034	0.05448	0.05517	0.05482
Proben1 82-16-8-19	665	0.32	0.008	0.85	0.07	1.03	0.05

Table 2.3: **different networks for the soybean data**

for evaluating the performance of learning algorithms for multi-layer perceptrons. The research described in this thesis makes use of this same benchmark and, the results achieved are compared and discussed against those published in [Prechelt 94], in chapter 6. Part of the results obtained with the **soybean** data are presented here, to provide practical evidence on the convenience of transforming nominal data to a continuous form instead of using a binary representation.

The data of this particular problem also required dealing with missing information in various attributes. In the continuous transformation encoding, each of them was replaced by the mean computed from the available data. The statistical parameter was chosen because it is very simple to calculate and, in certain experiments realised with the aim of determining whether better results could be obtained if the median was used instead of the mean, it turned out that there was no important difference when using either of them. The 82-dimension encoding used an additional binary pattern to replace all the missing instances. Appendix F contains a detailed description of the encoding method used to generate the continuous-valued representation. The procedure for 82-dimension encoding can be accessed at the **Proben1** website. Both include the exact manner in which the missing information was processed.

The analysis of the effectiveness of each of the encodings was realised through the design of two networks: one used as input the 82-dimension dataset, the other the 35-dimension continuous-valued transformation. The resulting networks, and the final measures obtained from processing the training, validation and test sets at the end of the design task, are contained in table 2.3. In it, the columns sep_tr, sep_vl, sep_ts correspond to the values registered for the **square error percentage**, the measure used by Prechelt in the **Proben1** benchmark. The columns labeled merror_tr, merror_vl, merror_ts contain the **mean square error** computed using the same data. The results presented in the third row were published in [Prechelt 94]. It is clear that the best results correspond to the 35-32-19 network built with the **GROPING** algorithm even though, the 82-8-19 network is simpler in terms of the number of hidden units it includes, but as the accompanying error measures are much improved with the continuous architecture, it is possible to conclude that the pre-processing phase used to reach this result is much more effective.

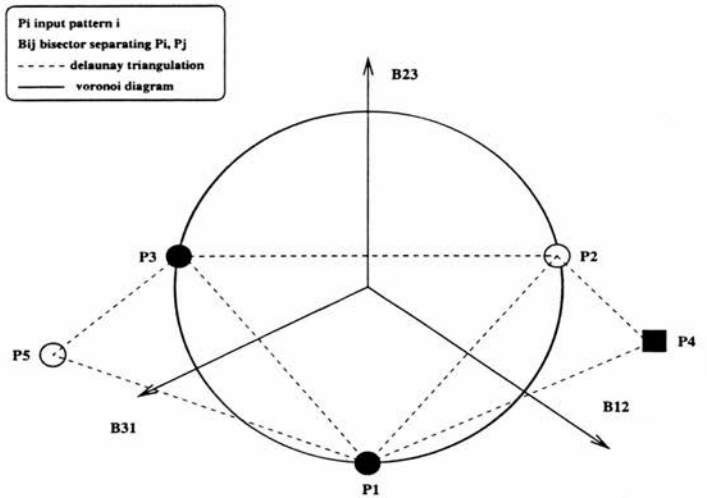


Figure 2.9: **input space partioning rendered by Voronoi diagram**

2.6 Essential idea underlying the solution built in this study

Throughout the chapter the limitations of the multi-layer perceptron have been outlined together with the most important approaches taken by different researchers mainly concerned with improving and enhancing the model's applicability. The example described in this section shows, in the simplest way possible, the main source of these limitations together with a general description of how the tools integrated in the design process built in this research can in effect aid a network developer in defining the appropriate architecture for a particular application.

Consider the kind of multi-layer perceptron network that will be required to partition the 2-dimensional space shown in figure 2.9. The space corresponds to a classification task in which each point is contained in one of three possible classes. The ones displayed in the figure, $\{P1, P2, P3, P4, P5\}$, each belongs to one of those classes. The subset of points $\{P1, P2, P3\}$ are inter connected by broken lines which spawn a triangle, *i.e.* a facet of the Delaunay triangulation that is possible to build from the complete set of points. The bold lines are the perpendicular bisectors of the edges that connect $\{P1, P2, P3\}$. Each of these bisectors contains an edge of the Voronoi diagram associated to the whole set. In figure 2.9 only a section of the geometric constructs that can be derived from the complete set of points is shown, so as to facilitate the understanding of the role their components can play in the task of designing a

multi-layer perceptron network for a particular application.

It is clear from the figure that the Delaunay triangulation is constituted by edges that join the points of the sample from which it is computed providing relevant information on the proximity of those points. At the same time, it is possible to compute the perpendicular bisector of each of these edges, *i.e.* the bold lines labeled B_{ij} which, if the related edge happens to connect sample points that belong to different classes, are clear candidates for operating as hyperplanes which will separate the space into the proper regions according to the classification task to which it is associated. This means that if they were used for initialising the weight vectors of the network designed to classify the points of the space being considered, it would require practically no training given that the weight vectors are already properly situated, at least those of the input-hidden layer.

Every point of an input set is used by the process that builds the Voronoi diagram and/or the Delaunay triangulation associated to it therefore, assuming that the computation of the geometric constructs for the number of patterns under consideration is workable, it is certainly possible to calculate the *maximum* number of hidden units and weight vectors required by a multi-layer perceptron network being designed for that same input set so that the resulting architecture will show an adequate performance. Although theoretically, it is always possible to build the Voronoi diagram and Delaunay triangulation of a set of points, there are cases in which the existing implementations of these geometric constructs encounter problems in doing so. Most of the difficulty is due to the precision available with the floating point computation of the machine because it makes certain of the elements of the set be coplanar. In such situations it is possible to conclude that the generation of an appropriate architecture will be extremely difficult, because the precision problem will apply equally to the computation of the weight vectors that are required. This means that, when encountering such a situation it is possible to apply the heuristic and conclude that there are strong enough reasons to search for alternative representation forms for that application. It might also be the case that one bisector is all that is required to correctly classify the problem domain which means that once the edges of the Voronoi diagram have been calculated they require further processing so as to eliminate those that can be deemed redundant.

From the above it is plain that the Voronoi diagram and Delaunay triangulation seem to offer effective devices to build a multi-layer perceptron network but, at the same time it raises many questions: (a) is the architecture that can be derived from them efficient?, (b) is it better than the one computed with more conventional methods?, (c) does it find a smaller architecture that still generalises well?. These issues are the ones that have been addressed in this thesis, the advantages and disadvantages of initialising multi-layer perceptron networks with this type of

geometric information, together with additional techniques, *eg.* principal component analysis, that were used to eliminate certain of the problems resulting from this type of initialisation.

2.7 Conclusions

The discussion contained in the present chapter clearly shows that, even though the representation capacities of the multi-layer perceptron have been theoretically proved, there is a lack of understanding regarding the most effective manner of building and imbuing a particular instance of the model with the knowledge of the application for which it is intended. Although many of the learning algorithms that have been defined to deal with the problem have enhanced in some way the effectiveness in using the model, only the application of *generative supervised learning* has resulted in an adequate alternative. This approach together with the incorporation of an adequate pre-processing phase to the whole network design process are the best ways of achieving a good network solution. This was supported by the set of examples presented which also helped to describe the problems that still need to be solved and, delimited some of the information that could be used towards their solution.

To provide a global picture of the matters discussed throughout this thesis a simple problem that shows how the information subsumed in the Voronoi diagram and Delaunay triangulation of a set of points might offer a reasonable basis for generating the type of network required is described pointing out the most important points that need to be discussed.

Also, a general framework providing a tool for the categorisation of the different existing methods for the development of networks based on the multi-layer perceptron model was outlined.

Overall the conclusions reached can be summarised in the following way:

1. The generative algorithms offer more appropriate methods for developing a network application given that they generally involve less *ad-hoc* decision making for deciding on the form of the network architecture and the specification of the parameters of the learning algorithm employed for the training phase.

2. There are three main types of generative algorithms: pruning methods, constructive and mixed. The pruning alternative is the least effective unless an operating network is generated before proceeding to execute the algorithm. On the other hand the constructive methods usually produce architectures that are more complex than the one required, [Fahlman & Baluja 94].

3. Any of the methods described involved an important information processing task. The

best results were obtained with those producing the correct amount and type of information. For a classification task, good information provides the algorithm with a clear description of the structure of the network's problem domain.

4. The best data for developing multi-layer perceptron applications is the one corresponding to continuous-valued attributes having little missing information. This means that it is necessary to incorporate a pre-processing phase to the network design process because the data of most real-world problems requires transformation to an appropriate continuous-valued form.

Chapter 3

Voronoi Diagrams and Delaunay Triangulations

3.1 Introduction

The Voronoi diagram of a set of points is a geometric construct that has been successfully applied to different areas, particularly in the last two decades [Okabe *et al.* 92].

The construct is one of the core concerns of computational geometry, a study area that mainly deals with the design and analysis of algorithms for geometrical problems. It has attracted enormous interest because many practically oriented fields such as computer-aided design, robotics, pattern recognition, give rise to problems that are inherently geometrical and it has been shown that it is possible to develop efficient implementations of many geometric concepts which can subsequently be put to good use in solving those problems.

The Voronoi diagram of a set of points in the plane, divides it according to the nearest-neighbour rule [Aurenhammer 91]: each point is associated with the region of the plane closest to it; creating a structure similar to the one comprised of the black lines shown in figure 3.1. In this same figure, the set of green lines that join those points whose regions share a black line, also constitute a structure that is the dual of the Voronoi diagram and is referred to as the Delaunay triangulation. The edges of the Voronoi diagram correspond to bisectors of the segments joining points of the set supplied as input therefore if the points happen to be associated to a classification task and if they belong to two different classes, the Voronoi edge is an appropriate candidate, to fulfill the role of weight vector, for a network architecture designed to implement a solution to the task. At the same time, the proportion of edges in the Delaunay triangulation connecting points of different classes provides a good indication of the level of intra-class correlation associated to the problem domain that is being manipulated, hence it

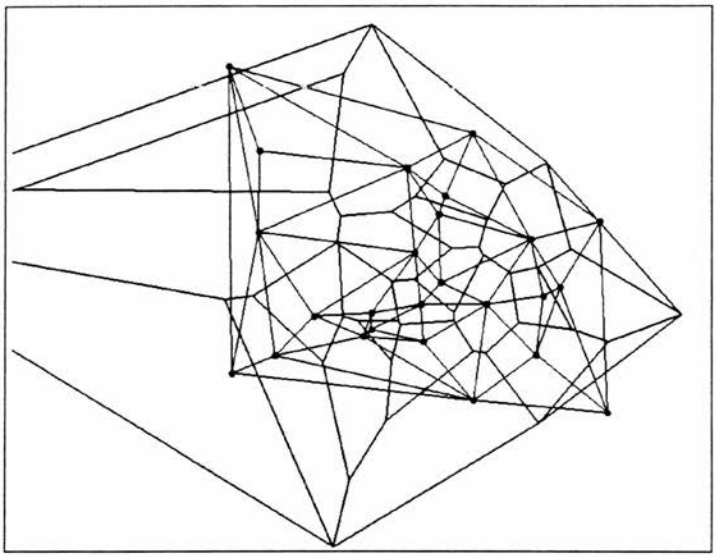


Figure 3.1: **Voronoi diagram and Delaunay triangulation of N sites**

results in an effective tool for deciding on the relative convenience of developing a network for the application of interest.

The present chapter contains a detailed description of the information subsumed in these geometric constructs and, an explanation on how it is exploited in the network design method proposed in this research. Section 3.2 includes a definition of the Voronoi diagram and Delaunay triangulation together with their basic properties. As it involves the frequent use of various geometric terms, appendix C consists of a glossary that incorporates the meaning of the relevant ones including the most important geometrical assumptions used in their definition. Section 3.3 provides details and examples of the specific ways in which it is applied to achieve an appropriate multi-layer perceptron network. Section 3.4 discusses applications for network design, developed by other authors which also exploit geometric information, comparing their effectiveness, in the pertinent cases, against the alternative application specified in this work. The most important implementations that exist for computing the relevant geometric constructs are described in section 3.5, together with an explanation supporting the implementation that was included in the network design process developed. Finally, section 3.6 summarises the points that have been brought up, linking them to the rest of the contents of this written work.

3.2 Voronoi Diagrams & Delaunay Triangulations

In the previous chapter it was indicated that the foremost input to the process for developing a multi-layer perceptron network is a set of pairs:

$$T_n = \{(x^1, o^1), (x^2, o^2), \cdots, (x^k, o^k), \cdots, (x^n, o^n)\} \tag{3.1}$$

where each x^k corresponds to an N-dimension vector, $x^k = (x_1^k, x_2^k, ..., x_N^k)$, $x^k \in R^N$. As it is necessary to evaluate the network performance, not only with respect to the elements comprising the set used for training it but also with respect to sets of elements not used for this operation, *ie.* the validation and test sets, the original source of n elements is usually divided in two groups, one of p elements which will correspond to the training set and the remaining $n - p$ pairs will be grouped into the validation and test sets. This means that the training set will have the form:

$$T_p = \{(x^1, o^1), (x^2, o^2), ..., (x^p, o^p)\} \tag{3.2}$$

with $p < n$.

Once the particular contents of each of the sets required has been specified, it is possible, to subtract all the vectors x^p from T_p and, interpret them as a set of p distinct points in the N-dimension euclidean space, *ie.* $\mathbf{X} = \{x^1, x^2, ..., x^p\}$.

Assuming that $N = 2$, given that it corresponds to the training set of a multi-layer perceptron design procedure, $\mathbf{X}$ can be the input to a partition process which labels each of its elements x^p as a site and, subsequently assigns each of the points comprising the 2-dimension euclidean space to the region corresponding to its nearest site. All those points associated to x^k will form the **Voronoi region $\mathbf{V}(x^k)$** which will be comprised of all the points at least as close to x^k as to any other site:

$$\mathbf{V}(\mathbf{x^k}) = \{x \mid \|x^k - x\| \leq \|x^m - x\|, \forall k \neq m\} \tag{3.3}$$

The definition of each $\mathbf{V}(\mathbf{x^k})$ corresponds to that of a closed set and is collectively exhaustive in the plane because every element is assigned to at least one member in $\mathbf{X}$. Those elements of the plane assigned to the set of two or more sites form the boundaries of the regions, *ie.* the bold lines in figure 3.1. Adjacent regions are overlapped only on their boundaries. Geometrically the arrangement complies with that of a *tessellation.* The particular case described corresponds, as mentioned in the introduction, to the planar ordinary Voronoi diagram of $\mathbf{X}$ and, the regions comprising the diagram constitute ordinary Voronoi polygons, [Okabe *et al.* 92].

Given that a Voronoi diagram, $\mathcal{V}(\mathbf{X})$ consists of polygons and, that a polygon can be defined in terms of half planes, it is also possible to define the Voronoi diagram in terms of half planes

in the following manner:

$$\text{a set } \mathbf{X} = \{x^1, x^2, ..., x^p\} \subseteq R^2,$$
$$\text{where } 2 \leq p < \infty \text{ and, } x^k \neq x^m, \forall k \neq m.$$

The region:

$$V(x^k) = \bigcap_{m \in [1,p]_{m \neq k}} H(x^m, x^k), \tag{3.4}$$

where

$$H(x^m, x^k) = \{x \mid \|x - x^m\| \leq \|x - x^k\|, \forall m \neq k\}$$

corresponds to the half plane that results from the partitioning generated by each of the bisectors that bound the resulting Voronoi regions.

$V(x^k)$ is the Voronoi polygon associated with x^k and,

$\mathcal{V}(\mathbf{X}) = \{V(x^1), ..., V(x^p)\}$ is the Voronoi diagram generated by $\mathbf{X}$.

The previous definition is particularly useful because it is easily extended to an N-dimensional space, therefore it is possible to describe the N-dimensional Voronoi diagram, in the following manner [Okabe *et al.* 92]:

If the set of points $\mathbf{X} = \{x^1, x^2, ..., x^p\}$, is such that,

$$x^k = (x_1^k, x_2^k, ..., x_N^k),\ x^k \in R^N,$$
$$3 \leq N < \infty, x^k \neq x^m,\ \forall m, k \in [1, p].$$

The region given by:

$$\begin{aligned} V(x^k) &= \{x \mid \|x - x^k\| \leq \|x - x^m\|, k \neq m, k, m \in [1, p]\} \\ &= \bigcap_{l \in [1,p]_{l \neq k}} H(x^k, x^l) \end{aligned} \tag{3.5}$$

comprises the N-dimensional ordinary Voronoi polyhedron associated with x^k, and the set,

$$\mathcal{V}(\mathbf{X}) = \{V(x^1), V(x^2), \cdots, V(x^p)\},$$

constitutes the N-dimensional ordinary Voronoi diagram generated by $\mathbf{X}$. The validity of this extension immediately follows from the existing equivalence between the definitions of the half-planes in the 2-dimensional space (equation 3.4) and the N-dimensional space (equation 3.5).

In the same way that a planar graph has its dual graph, a Voronoi diagram has its dual tessellation, called the Delaunay tessellation or, Delaunay triangulation [Okabe *et al.* 92]. Given that they are dual structures, each contains the same information in some sense, but represented in a different form [O'Rourke 94]. In the plane, the Delaunay triangulation of a set of points, *ie.* sites, contains a (straight-line) edge connecting two sites in the plane if and only if their Voronoi regions share a common edge, a fact that is evident by looking at the placement of the

broken lines included in figure 3.1.

In the context of multi-layer perceptron design, given the existing duality between the Voronoi diagram and the Delaunay triangulation, in the first instance, it would seem sufficient to use only one of the two constructs but, because the same information is imbued in each of them in a particular way, these two alternative organisations of similar information is precisely what can be successfully exploited when building a multi-layer perceptron network. Formally, the link between the Delaunay triangulation and the Voronoi diagram was established in 1934 by Delaunay, [O'Rourke 94], who proved that when the dual graph of a Voronoi diagram, in 2-dimension, is drawn with straight lines, it produces a planar triangulation of the Voronoi sites which is the Delaunay triangulation of those same sites.

No formal proof is included in this work regarding the effectiveness of exploiting the information contained in the geometric constructs described. Nevertheless it is possible to support this claim by looking at certain of their properties, explaining how their application aids in improving the performance of a learning algorithm for building a network architecture and also, through the empirical results included in one of the subsequent chapters.

The properties of the Voronoi diagram and the Delaunay triangulation presented, correspond to those that facilitate an understanding of the manner in which they relate to each other and, how they support the task of building a better network architecture. A proof of their validity is out of the scope of this work but details of them can be found in [Okabe *et al.* 92]. Following the approach taken in the literature consulted, [Aurenhammer 91], [Okabe *et al.* 92], [O'Rourke 94], these properties are layed out in terms of the planar Voronoi diagram and Delaunay triangulation pointing out that, for the most part, they can be readily extended to an N-dimensional domain.

The properties of the Voronoi diagram $\mathcal{V}(\mathbf{X})$ of a set of distinct sites (points, input patterns) $\mathbf{X} = \{x^1, x^2, \cdots, x^p\}$, $\mathbf{X} \subseteq R^2$, can be summarised in the following way [Okabe *et al.* 92]:

V1. The set $V(x^k)$ defined by:

$$V(x^k) = \{x \mid \|x - x^m\| \leq \|x - x^k\|, m \neq k, m, k \in [1, p]\}$$

is a non-empty convex polygon and $\mathcal{V}(\mathbf{X}) = \{V(x^1), \cdots, V(x^k), \cdots, V(x^p)\}$ satisfies:

$$\bigcup_{k=1}^{p} V(x^k) = R^2 \tag{3.6}$$

$$(V(x^k) \setminus \partial V(x^k)) \cap (V(x^m) \setminus \partial V(x^m)) = \emptyset \tag{3.7}$$

$$k \neq m, m \in [1, p],$$

where $\partial V(x^k)$ corresponds to the boundary of the Voronoi region and the Voronoi Diagram $\mathcal{V}(\mathbf{X})$ thus defined, corresponds to a unique tessellation for $\mathbf{X}$. The property also

holds in R^N if a polygon is replaced by a polyhedron and R^2 is replaced by R^N. Theoretically the Voronoi polygons cover the euclidean space, therefore at least one Voronoi polygon is infinite but, because the geometric definitions that are of interest correspond to computer implementations, the type of Voronoi diagrams used in this context are bounded constructs. **V1** guarantees the existence of a network for a particular application on the grounds that it provides a sound theoretical basis for computing a **maximal** network architecture simply by selecting all those edges that correspond to bisectors of the points, *ie.* input patterns that share a boundary and, subsequently defining a one-hidden layer network architecture with as many hidden units as there are bisectors associated to sites belonging to different classes. For a set of P points, taking the worst case, the size of this network is $O(P!)$ considering that it could be necessary to define a hidden unit for each possible pair of points. Even though theoretically it is always possible to derive the Voronoi diagram of a set of points, if a subset of them happen to be coplanar it produces precision problems that limit the possibilities of an automatic generation. For such applications, the information can be used to explain the inappropriateness of using a multi-layer perceptron as representation form because, if the precision problems involve points of different classes, it will also be extremely difficult to compute the required weight vectors, a situation that justifies the search for other modelling alternatives.

V2. For the Voronoi diagram generated by a set of distinct points, $\mathbf{X} = \{x^1, x^2, \cdots, x^p\}$, $(2 \geq p < \infty)$:

(i) the Voronoi edges are infinite straight lines if and only if $\mathbf{X}$ is collinear.

(ii) the Voronoi edge $e(x^k, x^m)(\neq \emptyset)$ is a half line if and only if $\mathbf{X}$ is non-collinear and x^k, x^m are consecutive sites of the boundary of the convex hull of $\mathbf{X}$, $(\mathbf{CH}(\mathbf{X}))$.

(iii) suppose that x^k and x^m give a Voronoi edge $e(x^k, x^m)$. Then this edge is a finite line segment if and only if $\mathbf{X}$ is non-collinear and at least one of the x^k, x^m is in the interior of $\mathbf{CH}(\mathbf{X})$.

The set of conditions that must be satisfied by the Voronoi edges ensure that, any weight vector computed on the basis of one of them will always generate surfaces producing a distinct region for each of the related sites, *ie.* training patterns which, if they happen to belong to different classes, is as should be. A training process that has to operate on a network initialised with the geometric information described is guaranteed to be successful and short, because most of its weight vectors are properly located at the start of its execution.

V3. The nearest neighbour of site x^k, generates a Voronoi edge of $V(x^k)$. This means that the set of bisectors considered to be proper candidates for computing a subset of the number

of weight vectors required, will be associated with input patterns which are closer to each other than to any other member of the training set. The creation of such weight vectors is justified because these particular ones will be appropriately fulfilling their most important function as conflict resolution devices, given that they effectively separate potential problem regions of the input space.

V4. For every Voronoi vertex q^i in a Voronoi diagram in R^2, there exists a unique empty circle C_i centred at q^i which passes through three or more sites. Under the non-degeneracy assumption [Okabe *et al.* 92], C_i passes through exactly three. In R^N the circle is replaced by a (hyper)sphere and the number of sites involved should be at least $N + 2$. This property ascertains that, for the creation of the geometric construct the complete set of sites provided as input are properly considered, therefore there is no risk any of them will escape consideration in the subsequent computation of the initial set of weight vectors which could result in a problematic training process.

V5. Let P, P_e, P_v be the number of sites, Voronoi edges and Voronoi vertices of a Voronoi diagram in R^2, respectively ($2 \leq P < \infty$). Then

$$P_v - P_e + P = 1$$

Extending the property to an N-dimension space: take P_k as the number of k-dimension Voronoi faces in an N-dimension Voronoi diagram. Then:

$$\sum_{k=0}^{N} (-1)^k P_k = (-1)^N$$

The property basically ensures a *maximum* size to the network architecture required for the application of interest.

The Voronoi diagram has additional properties but given that they are, either derived from the ones already mentioned or touch on matters which are not relevant to the problem of interest they will not be cited in this work; the interested reader can find a complete description of all of them in [Okabe *et al.* 92].

In relation to the dual of the Voronoi diagram, *ie.* the Delaunay triangulation, even though the information content in both is practically the same, the particular way in which it is organised in each of them has a specific application in the multi-layer perceptron design task, therefore it is also necessary to elaborate on the properties of the Delaunay triangulation clearly depicting how it can be employed in designing a network. It is worth mentioning that given the rules underlying their building process is very similar and, the existing relation between them, the generation of one or both of them involves the same computational costs. This means there is no disadvantage in using both constructs notwithstanding the underlying redundancy.

If $\mathcal{V}(\mathbf{X})$ is the Voronoi diagram generated by a set of n distinct points, $\mathbf{X} = \{x^1, x^2, \cdots, x^n\} \subseteq$

$R^2, 3 \leq n < \infty$ that satisfies the non-collinearity assumption; $\mathbf{Q} = \{q^1, q^2, \cdots, q^{n_v}\}$ is the set of Voronoi vertices in $\mathcal{V}(\mathbf{X})$ and, $\{x^{i_1}, x^{i_2}, \cdots, x^{i_{k_i}}\}$ are the location vectors of the sites whose Voronoi polygons share vertex q^i. It is possible to define the following sets:

$$T_i = \{x \mid x = \sum_{j=1}^{k_i} \lambda_j x_j^i, \sum_{j=1}^{k_i} \lambda_j = 1, \lambda_j \geq 0, j \in [1, k_i]\} \tag{3.8}$$

and

$$\mathcal{D} = \{T_1, T_2, \cdots, T_{n_v}\}$$

such that, if $k_i = 3, \forall i \in [1, n_v]$, the set $\mathcal{D}$ is the Delaunay triangulation of the $\mathbf{CH}(\mathbf{X})$ spanning $\mathbf{X}$. If $\mathcal{D}$ includes T_i with $k_i \geq 4$, the tessellation is called a Delaunay pretriangulation and requires an additional stage of processing that reduces all the T_i to triangles. As $\mathcal{D}(\mathbf{X})$ is the dual of $\mathcal{V}(\mathbf{X})$ the properties underlying both constructs reflect this duality. The ones that are satisfied by the Delaunay triangulation and, relevant in the context of the multi-layer perceptron network design are the following [Okabe *et al.* 92]:

D1. the set T_i defined by equation 3.8 is a unique non-empty polygon, and the set $\mathcal{D}(\mathbf{X}) = \{T_1, \cdots, T_{n_v}\}$ satisfies:

$$\bigcup_{i=1}^{n_v} T_i = \mathbf{CH}(\mathbf{X})$$
$$(T_i \setminus \partial T_i) \cap (T_j \setminus \partial T_j) = \emptyset$$
$$j \neq i, i, j \in [1, n_v]$$

hence $\mathcal{D}(\mathbf{X})$ forms a unique tessellation spanning $\mathbf{X}$. As in the previous case this property guarantees that the generation of the geometric construct considers all the points comprising the input set, therefore when designing a network it is possible to conclude that all the relevant information subsumed in the training set plays some role in generating the network architecture.

D2: The Delaunay triangulation of a set of points $\mathbf{X}$ in the plane, $\mathcal{D}(\mathbf{X})$ is precisely the projection to the 2-dimensional space of the lower convex hull of the transformed points in three dimensions, transformed by mapping upwards to the paraboloid $x_3 = x_1^2 + x_2^2$. In [Okabe *et al.* 92] there is an extensive discussion on how it is possible to generate the desired geometric construct from the transformation mentioned. It should be noted that this property holds beyond the plane, *ie.* the Delaunay triangulation for a set of N-dimensional points is the projection of the lower hull of points in (N+1)-dimensions. The importance of this result in the network context derives from the fact that the

weight vectors comprising the input-hidden layer of a network must be defined in an (N+1)-dimension space, assuming the input domain is N-dimensional, given that it is necessary to provide weights for the connections coming out of the threshold unit apart from the ones linking each of the input units and specified hidden units. Computing a set of weight vectors, making use of the information subsumed in the convex hull derived from the points comprising the training set, after they have been extended to a higher dimension provides an alternative network initialisation. One that is even more effective than the one described previously which is restricted to the Voronoi edges, given that these are only defined in a space that has the same dimensionality as the input.

D3: For the Voronoi diagram $\mathcal{V}(\mathbf{X})$ generated by $\mathbf{X}$, and the Delaunay triangulation spanning $\mathbf{X}$, let Q and Q_d be, respectively, the set of the Voronoi vertices and that of the Delaunay vertices; E and E_d the set of Voronoi edges and that of the Delaunay edges; C_d the set of circumcentres of the Delaunay triangles. Then:

(i) $Q_d = \mathbf{X}$.

(ii) $C_d = Q$.

(iii) $|E_d| \geq |E|$; $|E_d| = |E|$ if and only if $\mathcal{V}(\mathbf{X})$ is non degenerate, $|E|$ is the number of elements in the set E.

The conditions satisfied by the elements that comprise the basic building blocks of the Delaunay triangulation ensure the appropriateness of using them for deciding which patterns of the training set are responsible for causing an inefficient learning process. The fact that E_d has at least, as many elements as E and, as each element of E is associated to Voronoi regions, *eg.* $V(\boldsymbol{x}^k)$, $V(\boldsymbol{x}^m)$, that share a boundary, the elements in E_d guarantee the existence of enough geometric constituents to manage conflicting input patterns. Given that the Voronoi edges correspond to boundaries of the existing Voronoi regions and, that sites sharing a boundary are nearest neighbours, being connected by an element of E_d can indicate that there might be a problem in generating the required network architecture if the endpoints of the element in question happen to belong to different classes. Accepting as valid the geometric interpretation of the multilayer perceptron model, [Hertz *et al.* 91], [Hallam & Ross 93], the existence of a surface connecting elements of the training set is practical evidence of the possibility of building a network of this type to solve the classification task of interest granting that the surfaces described can be used to compute the weight vectors associated to each of the required hidden units. Once the Delaunay tessellation has been computed it is possible to build the necessary network calculating the surfaces that intersect the Delaunay instances connecting points of the training set associated to different classes.

D4: For the Delaunay triangulation $\mathcal{D}(\mathbf{X})$ spanning $\mathbf{X}$, which satisfies the non-circularity assumption [Okabe *et al.* 92], let n_e be the number of Voronoi or Delaunay edges, n_v the number of the triangles in $\mathcal{D}(\mathbf{X})$ and n_c the number of vertices on the boundary of $\mathbf{CH}(\mathbf{X})$. Then the following equations hold:

$$n_v = 2 * n - n_c - 2 \tag{3.9}$$

$$n_e = 3 * n - n_c - 3 \tag{3.10}$$

These numeric relations guarantee the existence of a finite network architecture that will be able to support the application of interest and will take into consideration the complete training set.

Finally it is necessary to reflect on the relevance the set of assumptions associated to the concepts described have, at the moment of building the Delaunay triangulation and/or Voronoi diagram of a set of points for subsequently shaping a multi-layer perceptron network. Given that the most effective way of computing any of them is through the generation of the convex hull spanning the set provided as input, the issue can be viewed in terms of how the violation of any of the underlying theoretical assumptions is treated by the algorithm applied for their computation. The one used in this research follows the policy of merging those facets that are comprised of problematic points, *eg.* collinear or co-circular points. This implies that unless the set of points grossly violate the relevant set of assumptions, it is capable of building the geometric constructs associated to the set provided as input. The algorithm always indicates how difficult it was to process the input set, together with a list of the points that created difficulties, such information can be used to explain the appropriateness of using the multi-layer perceptron model for a particular application. Therefore this type of approach, in the design of a multi-layer perceptron network, will always provide useful information and a reliable direction to pursue when building an architecture for a particular application, no matter the form of the input space, as it is able to provide pertinent insights on this most important issue.

3.3 Computational Geometry & Multi-Layer Perceptrons

As the material described throughout the chapter is, for the most part, set in a 2-dimension space, the examples included in this section also correspond to 2-dimension problems. The ones discussed, which are plotted in figure 3.2.a and 3.2.b, constitute good examples for showing the effectiveness of applying the information subtracted from the geometric constructs defined in the preceding section, to the following network design operations:

a.- Evaluation of the convenience of using the model for the application of interest.

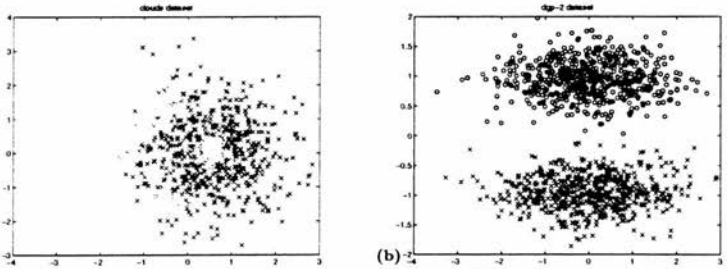


Figure 3.2: **Form of discussed input spaces: (a) clouds data, (b) dgp-2 data**

b.- Initialisation of the maximal network architecture required by the problem.

The plot in Figure 3.2.a represents a badly structured set for which it will be more convenient to find alternative representation forms other than the multi-layer perceptron. By contrast, the structure of the data plotted in Figure 3.2.b corresponds to a problem for which an application based on the multi-layer perceptron model will be highly convenient. The difference between them lies in the easiness with which it is possible to identify the regions associated to the different classes that comprise each of the problems. In the data of the cloud problem, although there are proper clusters of points, each belonging to one class, there is a lot of overlap which makes it extremely difficult to properly compute the number and value, of the necessary hyperplanes that the network architecture must include to operate as a reasonable classifier. On the other hand, the points of the 2-class problem plotted in Figure 3.2.b are structured in two well discerned and non-overlapping clusters. This last type of data is one for which the multi-layer perceptron is a particularly convenient representation form.

3.3.1 Measuring convenience of building a network implementation

The Delaunay triangulation of a set of points in the plane consists of a number of edges that must satisfy an important criteria. Each of them must connect two points whose Voronoi regions have a common boundary. This fact implies that these points are closer to each other than to any other point in the set, excepting those with whom it shares other boundaries which, by virtue of being in a two dimension space, cannot be greater than three.

The preceding description elucidates a possible strategy that can be pursued at the moment of having to decide on the adequacy of designing a mutli-layer perceptron network for a particular application. Building the Delaunay triangulation of the set of points associated to it, subsequently counting the number of edges connecting those points, so as to compute the

percentage of them having endpoints that belong to different classes, *i.e.* mixed edges, and, also determining the proportion of edges joining points that belong to the same class. An analysis of these two values provides an insight on the level of complexity that the required network must have and, help decide on the adequacy of employing the multi-layer perceptron as representation tool. Even though, there seems to be a strong similarity between this type of computation and the nearest neighbour technique, [Bishop 95], it is worth noting that, while there are various classifier methods founded on the nearest neighbour theory, the percentages described simply provide an understanding of the structure of the problem domain of interest and are not being used as main components of a classifier, the objects enumerated to compute them are the ones that constitute important components of the classifier being defined.

In the Delaunay triangulation of a given set of points it is possible to calculate different degrees of class-intersection. Considering the basic components of this geometric construct, the types of intersection that can be subtracted from it are:

(a) facets intersection: divide the points in as many groups as there are classes so that, the points in any group all belong to the same class. Build the Delaunay triangulation of each of these groups. As these geometric constructs are comprised of a set of facets, it is possible to calculate how many of the ones comprising any of them, intersect with those of each of the Delaunay triangulations generated from the other groups. For the two datasets described at the beginning of the section, the results of this particular computation are outlined in the summary of results presented at the end of this section.

(b) edges intersection: build the Delaunay triangulation of all the points that comprise the training set. Count the number of mixed edges, *i.e.* those edges that have endpoints that belong to different classes, and compute the percentage of the total set that is mixed. Table 3.1 contains the values calculated for this kind of intersection on the cloud and dgp-2 datasets.

The results on the examples presented were generated for different sizes of the same type of data so as to determine how sensitive the measures were to the number of elements in the sample. The values in the table and, those outlined at the end of the section, show that the intersection measure described is not affected by the size of the sample therefore, it can be confidently used to draw conclusions on the most appropriate manner of processing the problem domain from which they were derived. It has a similar behaviour to that of other measures of the same sort, *e.g.* the sample mean, it is more reliable the larger the sample.

Of the intersection measures defined, the one obtained from the edges is the more important of the two because, it provides more manifest evidence of how interlaced the different classes in

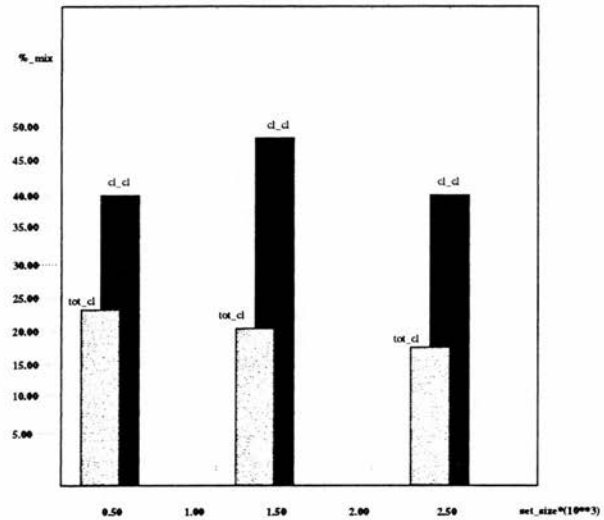


Figure 3.3: **Intersection degree in Clouds dataset**

the problem are. The bar plots included in figures 3.3, 3.4 were produced from the intersection information. Plot 3.3 was built using the outcome of the intersection analysis performed on the clouds data and, plot 3.4 was built using the information subtracted from the dgp-2 set. The plots clearly show a much higher level of intersection in the clouds set which, agrees with the type of structure shown in the clusters of points contained in figure 3.2. The analysis was performed for the same type of data using samples of 500, 1500, 2500 elements, and, calculating the values of two types of intersection for each of them. The results correspond to the 6 bars comprising the plots in figures 3.3, 3.4. The ones labeled $tot\_ < nomset >$ reflect the edge intersection degree ($< nomset >= cl$ for the clouds problem, $< nomset >= dgp$ in the dgp-2 example). Those labeled $cl\_ < nomset >$, following the same rule as in the $tot\_ < nomset >$ bars, reflect the percentage of intersection between the facets of the geometric structures which are being analysed. The most salient feature is the relative difference between the intersection measures calculated in each of the samples. It is more pronounced in the dgp-2 dataset which, together with the fact that the percentage of mixed edges in all sample's sizes is quite small clearly indicates that the underlying structure of the problem is quite appropriate to be modeled with the multi-layer perceptron.

The main difficulty regarding these measures stems from the computational costs involved in building the geometric structures from which they are deduced because, for high dimension spaces generating them is very computationally intensive. Therefore, it could be concluded that

its use for evaluating the convenience of using the multi-layer perceptron model is impractical given that most of the applications that make use of the model tend to be high dimensional ones. Yet, taking into account:

(a) the existence of very efficient implementations for building the Delaunay triangulation of a set of points in low dimension spaces and,

(b) that the projection transformation affects the position of the points but not the class they belong to, given that the percentages computed measure the existing degree of class intersection there is in the problem domain, it is possible to guarantee a similar conclusion for both, the original data and its reduced version.

The above considerations are a clear indication of the practicality of performing the structural analysis that has been described. The fact that, the geometric constructs evaluated are the same ones subsequently used to compute the initial network's weight vectors, if the analysis indicates that it is an appropriate representation tool, further enhances the utility of this initial step in the multi-layer perceptron's design process.

Additionally, the implementations used in this research for computing the Delaunay triangulation of the training set generate information describing it, showing how appropriate the data provided as input is. This information indicates whether the points satisfy or not the set of conditions that need to be complied with to proceed with the construction of the **convex hull** from which the Delaunay triangulation and Voronoi diagram of the set of input points is derived. This initial analysis in itself clearly indicates how difficult, or otherwise, the task of designing a network is going to be. The use of this information is of course, conditioned to those operating spaces which are originally low-dimension given that for the high dimension ones the geometric analysis only takes place in a projection of the input and, this particular type of transformation can only result in spaces which originally might be very simple.

```
Intersection on the clouds data
size 500; avr_prt = 40.326;            size 1500; avr_prt = 49.8949;
     num_c1 = 249; num_c2 = 231;            num_v1 = 383; num_v2 = 367;
     num_v1 = 129; num_v2 = 121;            num_c1 = 753; num_c2 = 721;
     Class (1 into 2) -> num_inter = 100    Class (1 into 2) -> num_inter = 376
     Class (2 into 1) -> num_inter = 93     Class (2 into 1) -> num_inter = 360

size 2500;  avr_prt = 41.5711;
     num_v1 = 636; num_v2 = 614;
     num_c1 = 1258; num_c2 = 1214;
     Class (1 into 2 -> num_inter = 523
     Class (2 into 1)-> num_inter = 505
```

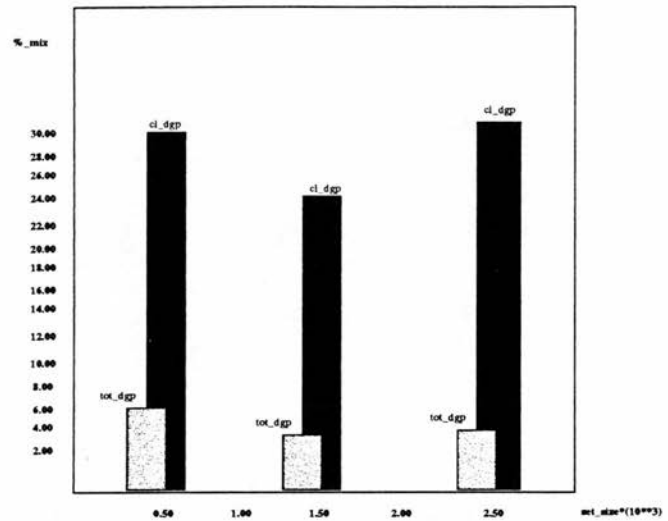


Figure 3.4: **Intersection degree in DGP-2 dataset**

```
Intersection DGP-2

size 500  avr_prt =28.523;
     num_c1 = 249; num_c2 = 227;
     num_v1 = 130; num_v2 = 120;
     Class (1 into 2) -> num_inter = 71
     Class (2 into 1) -> num_inter = 65

size 1500 avr_prt = 22.02;
     num_c1 = 719; num_c2 = 751;
     num_v1 = 367; num_v2 = 383;
     Class (1 into 2) -> num_inter = 158
     Class (2 into 1) -> num_inter = 165

size 2500  avr_prt = 29.4485
     num_c1 = 1238; num_c2 = 1236;
     num_v1 = 626; num_v2 = 624;
     Class (1 into 2) -> num_inter = 365
     Class (2 into 1) -> num_inter = 364
```

set size	tot_num_edges	num_mix	% mix
clouds dataset			
500	977	226	23.132
1500	2965	593	20.00
2500	4922	852	17.310
dgp-2 dataset			
500	989	35	3.54
1500	2976	45	1.512
2500	4931	46	0.933

Table 3.1: Geometric Analysis on cloud/DGP-2 data; Results complete set

3.3.2 Specifying network architecture through geometric information

Table 3.2 contains the results of the training processes realised to adjust the maximal networks that were subtracted from the geometric constructs of the clouds and dgp-2 problems described in the preceding section. The guideline obeyed to setup these training processes, the manner in which the data was divided into, training, validation and, test sets, the error measures registered, etc, it all complies with the benchmark rules followed to evaluate the performance of the learning algorithm defined in this research and, are clearly depicted in section 6.3.

The values included in the table correspond to the different error measures that can be calculated when training a multi-layer perceptron; there are three such evaluations associated to each of the problems, one for every sample size used for the geometric analysis discussed. In the results obtained, there is a clear difference between the training process that can be realised on well-structured data, such as the one associated to the dgp-2 problem, and that which is performed in a less properly structured sample such as the clouds data. In both cases low values were registered when classifying the training set. This means there is a high degree of correct classification for these sets but, looking at the values of the errors registered for the validation and test sets, it is evident that the network architecture that was designed by exploiting the geometric information subtracted from the less adequately structured sample is not the most appropriate solution because, high values for this particular type of error implies an inappropriate network generaliser. What is understood in this research as an appropriate generaliser is described in detail in chapter 6, although the discussions presented up to now make reasonably clear what this study understands as such.

The difficulty found when trying to generate an adequate network generaliser through the application of geometric information might suggest that it is an inadequate source and that it is better to consider an alternative approach. An appropriate way for deciding on this matter consists in evaluating the possibilities of finding a better solution than the one built using the geometry of the input space. Although there are no formal means for proving that such a possibility is practically non-existent, it is possible to argue that, given the evident complexity of the data structure, the amount of effort that needs to be incurred, to build such a solution, definitely labels the multi-layer perceptron as inadequate for the type of application. The model itself can be distinctly interpreted in geometric terms, the kind of function exercised by each of the hidden units incorporated to a particular network is of a distinct geometric nature, therefore it is very reasonable to conclude that no great improvement will be achieved by looking towards other sources when building the network for a problem which can be labeled as geometrically inadequate. Such problems are better managed with other data models.

Computing the number of hidden units required for the network of a particular application and the values of the weight vectors associated to each of these units, as it uses the same sort of information as the one applied when evaluating the structure of the input space, there is evidence of the same kind of problem, *i.e.* high computational costs for high dimension spaces. To deal with the difficulty, an approach similar to the one pursued in the input structure analysis was taken. The procedure defined to build the initial maximal architecture is comprised of the following steps:

(a) to reduce the computational costs involved in generating the geometric constructs utilised in this research, the data in the training set is projected to the 3-dimension space spanned by the three most informative dimensions of the input as determined by *PCA* which, theoretically correspond to the three eigenvectors associated to the highest three eigenvalues of the data's covariance matrix, [Jolliffe 86]. The size of the basis is restricted to three vectors because, as the aim of the projection operation is to take advantage of the very efficient implementations that exist for computing the necessary geometric constructs in low dimension spaces, these can only be at most 3-dimensional. Also, (1) for many problems a large percentage of its information variance can be explained with very few of its eigenvectors and, (2) the very reasonable results finally achieved with the learning algorithm.

(b) compute the Delaunay triangulation of the projected data. This information produces a set of edges that connect all the elements of the input. There are geometric instruments for computing the bisectors of all these edges and, it is precisely this information that is used to determine the values of the required weight vectors. It is not possible to calculate fewer weight vectors than the ones really required in the original input space because, a projected version of a high dimension space will never be simpler than its original form, given that the same information is pushed into fewer degrees of freedom. For this reason, it is possible to affirm that the network finally computed is maximal. It is probable that the correct and precise architecture required will not be computed but, as the subsequent learning algorithm is of the pruning-generative type, it incorporates the mechanisms to eliminate the superfluous hidden units.

(c) the bisectors derived from the 3-dimension space need to be expanded to the original input space before they might be used as potential weight vectors. To calculate the required expansions it is possible to make use of the conditions these weight vectors must satisfy, look into the kind of values each of its components must have so that, every element of the training set might be correctly classified by the network. It could be argued that, given the effort invested in computing the three most informative dimensions of the problem

domain and, in the subsequent projection, the final network design could be realised in this reduced version of the space but as the complete set of weight vectors still needs to be determined and, the specific network training operation is based on an optimisation method that can be very effective if the initial conditions on which it starts operating are appropriate, there are strong enough reasons to consider that a more effective, final multi-layer perceptron is produced in the original high-dimension input space.

Efficient use is made of the most appropriate type of information for designing an application for a problem based in the multi-layer perceptron. It is not possible to prove that the architecture that can be derived from this information might be considered the best solution but the results obtained for the experiments described in chapter 6 show that, compared to the solutions obtained using other learning methods for the same kind of model, the one based on the geometric constructs described at this point is very adequate.

problem	iterations	sep_train	sep_validation	sep_test	me_train	me_validation	me_test
Errors for clouds data							
clouds-500	1480	0.02803	15.57889	19.25158	0.00548	0.17398	0.2145
clouds-1500	5004	1.67860	10.24369	14.53699	0.04149	0.12131	0.15914
clouds-2500	1251	6.43904	8.48186	10.24552	0.11167	0.13046	0.14757
Errors for dgp-2 data							
dgp2-500	10	0.06343	0.06314	0.07454	0.02348	0.02352	0.02516
dgp2-1500	12	0.16598	0.61853	0.969564	0.03322	0.04328	0.053225
dgp2-2500	14	0.00208	0.04193	0.00563	0.00043	0.00142	0.00057

Table 3.2: **Results published in training maximal network**

3.4 Other geometric-based network applications

The literature of computational geometry, specially in the last few years, has shown substantial interest in the geometric structures previously described, in particular in the Voronoi diagram, an interest which can be automatically extended to the Delaunay triangulation as one is the dual of the other. According to Aurenhammer [1991] there are three main reasons for this state of affairs. First, Voronoi diagrams arise in nature in various situations given that there are many natural processes that can be used to define particular classes of such geometric constructs. Second, the interesting and surprising mathematical properties they have which links them to various other well-known geometrical structures and, mathematical-statistical theories. Finally, the Voronoi diagrams have proved to be a powerful tool in solving various computational problems which explains its wide present applicability in computer science.

These causes encompass an enormous variety of possible applications which have as common ground in the use each makes of the Voronoi diagram and/or Delaunay triangulation to support

their development, the exploitation of the relation satisfied by the edges that comprise such structures, *ie.* the nearest-neighbour rule. Information of this type is crucial for the successful realisation of various tasks which easily explains the multitude of uses that have been found for it. Although it would be interesting to dwell on these applications, as the main concern of this research is the neural network model the rest of the section will be restricted to a discussion of the ways in which network development has been and might be improved by the incorporation of the information subsumed in both geometric structures, a detailed bibliographical review on further applications can be found in [Aurenhammer 91].

Of the three main reasons depicted by Aurenhammer [1991], there are two which can be linked to the network models and which help explain the applicability computational geometry has in this particular context. In the first place there are certain network applications whose main objective is to recreate different natural phenomenon, *eg.* visual perception, and it has been shown that in Nature the underlying structure of many systems can be reflected with a Voronoi diagram, a fact that could be exploited in the neural network models designed to recreate this type of phenomena. Alternatively, in many of the applications based in this type of model, the achievement of a successful result greatly depends in properly capturing the structure of the domains involved in the problem of interest and, it is usually the case that either, only a sample of such domains is accessible or, the time factor is an important component of the whole problem. In both of these situations the application of statistics is generally the most reliable manner for obtaining the type of information required. As an important area of work on the statistical properties of the Voronoi diagrams has been developed and is still being developed, see for example the research described in [Gilbert 61], [Conway & Sloane 82], there seems to be a lot of potential in the integration of network models, statistics and computational geometry for the development of reliable and powerful learning systems.

In terms of existing applications, Delaunay triangulations and/or Voronoi diagrams have been used as the primary knowledge structures of algorithms aimed at function learning and, also as supporting components in the design of neural network models, these last include a method aimed at building multi-layer perceptron architectures which constitutes an alternative to the kind of process described in this work.

The most important of these applications corresponds to the method described in [Bose & Garga 93] which uses the Voronoi diagram of a set of points, *ie.* the training set, to build a multi-layer perceptron network. Its main advantage rests in the possibility it offers of defining an architecture following a series of steps, all of them clearly depicted and not requiring any *ad-hoc* decision making. Given the enormous number of design procedures that have been defined since the network model and its first learning algorithm came into existence, all comprised of

some form of arbitrary decision making, *eg.* the appropriate initialisation of some convenient parameter, having access to a method that does not incorporate any form of arbitrariness in the first instance seems particularly attractive. Nevertheless, when considering that in present days, the model is mostly used for implementing tasks whose domains are not completely well known, contrary to those like the parity problem for example and, that it is more important to achieve a multi-layer perceptron network that performs better as a generaliser, it might seem reasonable to provide some form of uncertainty in the building procedure that will result in more flexible architectures, *ie.* in better learners.

The fact that Bose and Garga, in the results described in [Bose & Garga 93], make no mention on the generalisation capacity of the architectures obtained by the application of their process, opens the possibility of doubting the generalisation capacity of such networks. Although in this research their method was not implemented to provide empirical proof of this drawback, there are other points which also diminish its initial potential usefulness:

- The rules provided for deciding on the generation of additional units and/or layers, will usually result in more complex architectures than the one required for the problem under consideration. This turns out to be the case because it is not uncommon to have to operate on an input space in which the intersection between the points belonging to different classes is not empty, therefore more often than not, the method will generate a network architecture with two layers of hidden units.

- Their architecture only uses units with a linear activation function as that is a reliable manner of implementing hyperplanes, their main device for solving conflicts in the classification process. It is a valid alternative given the type of solution they offer but most of the history of the multi-layer perceptron model shows that networks built with sigmoid activation function units have greater representational capacity, therefore at the end of the day more efficient architectures for solving the same kind of problem can be achieved with sigmoid units.

- They mention the potential of extending their solution to higher-dimension problems given the possibility of applying fast algorithms for computing the Voronoi diagram of a set of points. This, at least at present is not possible, given that the existing algorithms, at least the ones reviewed, are exponential on the dimension and size of the input space, although very fast implementations have been created for problems in 2 and 3 dimension domains.

After reviewing Bose and Garga's work it seems reasonable to conclude there is no great advantage in using geometric information for improving a neural network design process, given

the computational costs involved when operating on high-dimensional domains. Yet the generative learning algorithm described in this work clearly shows that it is possible to exploit the very useful information contained in the geometric constructs mentioned but without being affected by a high computational cost.

Bernd Fritzke in [1993] describes a network model that incorporates two variants. The first performs unsupervised learning and includes the ability to automatically find a suitable network structure and size through a controlled growth process based on geometric information. The second variant of the model is a supervised learning method which results from the combination of the self-structuring network mentioned, together with the radial basis function approach. The objective pursued when designing a self-structuring network based on the model is to generate a mapping from the original input space R^N onto a discrete k-dimensional topological structure. The mapping generated must satisfy the similarity relations in the original input. This is achieved by ensuring that when executing the network's learning process the structure underlying it satisfies the properties associated with the Voronoi diagram of the points comprising the input space. The importance of the model stems from the fact that it is providing a very different manner of applying Voronoi diagrams to neural networks. Given the properties associated with the Voronoi diagram, it is possible to conclude that the idea of generating network structures that continuously evolve so that they comply with the characteristics of the geometric construct under consideration, will certainly guarantee the preservation of the similarity relations. What is not so obvious is how the method is an improvement in network development given that already there are various algorithms for computing Voronoi diagrams and, as the main idea of the algorithm presented can be viewed not only as an innovative learning algorithm but, also as a novel way of computing the Voronoi diagram of a set of points, to definitely evaluate its usefulness, its performance should be compared against other methods defined to produce such geometric constructs.

At the beginning of the section the close relation that exists between the geometric constructs considered in this research and many natural processes is indicated as one of the reasons for their present popularity. The association can be viewed from a function approximation perspective. In this particular context the relation is engendered through the numerous motor and perceptual systems studied in areas such as robotics, vision, speech, graphics, all of which result in interesting emerging techniques. One such technique are the geometric learning algorithms developed by Omohundro, [1989a], [1989b]. The underlying aim is to learn smooth non-linear mappings from sets of random input output pairs in high dimension spaces, this is achieved by initially organising the input points in triangles and subsequently applying linear interpolation within each triangle. To achieve the best results it is necessary to choose appropriate triangles

and given their properties the Delaunay triangulation is an optimal choice, something that is formally proved in [Omohundro 89a]. As the number of triangles that must be checked at the time of having to compute the output associated with a particular input point grows considerably with the size and dimension of the data available, to reduce the time spent in searching for the proper triangle a support structure is defined, an N-d (N-dimensional) tree, *ie* a binary tree in which each node contains information identifying the region of the input space covered by the nodes below it, an ordering of the Delaunay triangles that guarantees a logarithmic search of any triangle when approximating an input pattern. A classification example is described in [Omohundro 89b] to compare the learning model proposed against the network model and, although theirs is more efficient in terms of the number of computations that must be performed to correctly classify an input, this is due to the supporting search structure, *ie*. the N-d tree, because restricting the comparison to their method without the tree against the network model, there is not much difference given the number of computations that must be performed for detecting the appropriate triangle and the subsequent linear interpolation procedure. Even though the overall learning system seems to be more efficient in terms of computations, when measuring operational costs in terms of the amount of space required by the whole system there are a number of more efficient alternatives, [Bose & Garga 93], [Fritzke 93], the multi-layer perceptron network described in this work.

Although it is not possible to ascertain that the above discussion comprises all the relevant existing research in which geometric information is integrated to learning systems to improve the overall learning process, a thorough search of the area was performed and, the cases described correspond to what is considered to be the most important work developed in the same line as the one pursued in this research.

3.5 Existing implementations for computing the relevant geometric constructs

As it is possible to associate a geometric interpretation to many of the problems different research areas have to deal with, a lot of effort has been invested in trying to reduce the computational costs that must be incurred when using many of the existing implementations.

For the case of the Voronoi diagram and the Delaunay triangulation all the most relevant ones exploit the relation that exists between these concepts and, the convex hull generated from the same set of points. Suggested by Edelsbrunner, Siedel in [1986], for the 2-dimension case it was summarised in the following theorem:

- **Theorem:** The Delaunay triangulation of a set of points in two dimensions is precisely the projection to the xy-plane of the lower convex hull of the transformed points in three dimensions, transformed by mapping upwards to the paraboloid $z = x^2 + y^2$.

As the relationship holds in higher dimensions, it is possible to state that the Delaunay triangulation for a set of N-dimension points is the projection of the lower hull of points in (N + 1)-dimensions [O'Rourke 94].

This relationship automatically suggests a means for implementing a process for the generation of the Voronoi diagram, Delaunay triangulation, of a set of points and, is the underlying principle of most of the existing methods for computing these geometric constructs.

The available implementations for computing the convex hull of a set of points can be classified according to the following criteria:

(**1**) the arithmetic employed in the implementation which can be, exact or floating point. This characteristic induces a second level of classification if the algorithm supports floating point arithmetic because the amount of precision allowed plays an important role on the type of solutions that can be generated.

(**2**) if it supports N-dimension spaces or only works for sets of points in 2 and/or 3 dimensions.

The majority of the implementations are restricted to exact arithmetic in 2 and 3-dimension spaces, because managing floating point precision can be extremely difficult if the input is not good, *eg.* processing coplanar points requires additional special treatment to support such things as division by a number considered to be approximately zero because of the machine's precision. The possibility of classifying the existing implementations according to the dimension of the space they are able to process, stems from the fact that generally geometric operations are exponential on the dimension of the input space, and only in 2 and 3-dimension spaces has it been possible to produce really efficient solutions. The existence of many more useful applications of geometry in low-dimension spaces has also played its part in bringing about this state of affairs.

Many types of algorithms have been defined for computing the convex hull of a set of points, each with the objective of providing a more efficient solution than the already existing alternatives. The most efficient of these implementations was first provided by R. Graham [1972], who managed to define an $O(n \log(n))$ algorithm for building the convex hull of a set of points in 2 dimensions. Given that, one of the principal operations underlying these algorithms is the organisation of the points into a specific order, because this is the main condition that must

be satisfied by the vertices of a polyhedra and, a convex hull is a polyhedra, furthermore it has already been proved that it is not possible to compute a sorting operation in less time than $O(n\log(n))$, the bound automatically applies to the process of generating the convex hull of a set of points given that its generation requires a sorting operation.

In a 3-dimension space the first algorithm that was shown to be $O(n\log(n))$ was the incremental algorithm. Described in [O'Rourke 94] it works adding the points one at a time, at each step constructing the hull of the first k points, and using the hull to incorporate the next point. The advantage underlying the philosophy of this algorithm is that it opened the possibility of building algorithms for generating the convex hulls of points set in higher dimensions, even though, up to this moment it has not been possible to produce one showing a $O(n\log(n))$ performance in a high N-dimension space, $N > 3$.

To avoid the high costs involved in computing the Voronoi diagram and Delaunay triangulation of a set of points set in a high dimension space, which is the type of space that is usually the input to a multi-layer perceptron network, it was necessary to include additional processing of the input data. This processing consists in projecting the training set to a 3-dimension space, computing the Delaunay triangulation of the projected points and, subsequently calculating the (N+1)-dimension weight vectors that will be assigned to the maximal network architecture, using the information of the geometric construct generated from the projected points. The only disadvantage of this process is that it will probably be the case that more weight vectors than the ones really required will be defined but, as the subsequent phase of the design process involves a pruning operation, this is no major drawback considering the gain in costs achieved by limiting the generation of the geometric constructs used to low-dimension points.

In 2 and 3-dimension spaces, the fastest algorithms that have been defined to compute the Delaunay triangulation of a set of points are the randomised algorithms, [Motwani & Raghavan 95], although the conditions that the input must satisfy to ensure that the algorithm will perform efficiently greatly restrict its applicability. Such is the case of Chew's algorithm that computes the Delaunay triangulation of a set of points in expected time $O(n)$ if the points are the vertices of a convex polygon and are fed to the program in the same order as they appear in the hull.

A more applicable alternative described in [Clarkson *et al.* 92], corresponds to an incremental algorithm which incorporates randomisation to the selection process of the input points when incorporating them to the convex hull. Its main drawback concerns the amount of memory it uses and, the fact that the existing implementations only operate with exact arithmetic.

The quickhull algorithm described in [Barber *et al.* 95] is also based on the incremental algorithm but, instead of selecting points in a random manner it selects them according to a

specific order based on the distance the points that have not yet been processed are from the convex hull that has already been built. Although this ordering makes the implementation more computationally intensive than its randomised version, it uses less space therefore it is possible to conclude that both implementations, *ie.* randomised, quickhull, are equally effective.

Quickhull has the additional advantage that it provides good support for floating point arithmetic ensuring the generation of the convex hull of any input. If it be the case that the points provided as input generate degenerate facets quickhull effectively deals with the difficulty by merging these problematic facets with a neighbouring element.

Providing an efficient 3-dimension implementation of the Delaunay triangulation of a set of points, and an adequate manipulation of floating point arithmetic were the main reasons for incorporating Barber's quickhull implementation to the network design process. The documentation and source code of the program can be accessed through the WWW at the following site: `http://www.geom.umn.edu/software/qhull/` .

3.6 Conclusions

This chapter describes a novel manner of approaching the design process of a multi-layer perceptron network for a classification problem. The examples presented are reliable evidence of the geometric nature of the network model which explains the reason for the effectiveness of using this type of information at the time of building solutions based on the model.

Geometry includes many concepts that have a wide applicability in other research areas. The ones that are useful in the development of multi-layer perceptrons, *i.e.* the Voronoi diagram and Delaunay triangulation, are discussed in detail. Particular emphasis is put on the properties they have that make them specially useful and, the manner in which they are derived given the high computational costs involved when applying them to high dimension spaces which, is the general rule in domains for which network applications are of common use.

The examples described also showed that it is possible to measure the value of using the multi-layer perceptron by inspecting the arrangement of the points in the geometric structures that are generated. There are no formal proofs regarding the validity of this measure but, the type of results described, the properties associated to the geometric constructs used and, the fact that each theoretical device is particularly useful for certain kinds of problems, should be evidence enough regarding the type of solution that is being offered in this study.

Chapter 4

Principal Components Analysis (*PCA*)

4.1 Introduction

The central idea of principal components analysis (*PCA*), is to reduce the dimensionality of a data set which consists of a large number of interrelated variables, while retaining as much as possible of the variation present in it. This is achieved by transforming the data to a new set of variables, the principal components, which are uncorrelated and ordered so that the first few retain most of the variation present in all of the original variables [Jolliffe 86].

Considering the preceding definition, the most evident manner of applying *PCA* to the area of multi-layer perceptrons, would be in a pre-processing phase of the network's inputs that will produce a reduced version of the data which the learning algorithm responsible of training the network architecture will find easier to manipulate [Bishop 95]. Although this approach might result in a simpler architecture, the loss of information that invariably takes place because of the dimensionality reduction, could result in a harder training process because, the algorithm would be operating on a less manageable input space harder to classify as, it is not taking into consideration all the information concerning the form of its input space.

Nevertheless, as *PCA* is a relevant technique in statistics for data analysis and, considering that a successful multi-layer perceptron network design process is very dependent on the availability of detailed information about the structure of the network's input space, it is reasonable to look at other ways of applying this technique for the development of this type of network. This chapter comprises the description of such an alternative, presenting arguments in support of the kind of application developed in this study, at least when compared against those described by other authors. Section 4.2 provides details of the concepts associated to *PCA* that

were used in this research together with a description of the way in which they were applied for the development of multi-layer perceptron networks. Section 4.3 discusses the alternative applications of this statistical technique that have been developed in this area. Section 4.4 provides information on the computational costs of the implementation process and, section 4.5 summarises the contents of the chapter.

4.2 Applying *PCA* to the design of Multi-layer Perceptron networks

Making the most effective use of any technique towards the achievement of a specific goal, involves two basic steps: understanding the technique, its components and how to proceed on its application and, having a clear idea of the goal that wants to be achieved, or rather, of how the technique under consideration fits into the process that is being defined. For this reason the present section can be further divided in two parts, the first one containing details on the theoretical elements that underlie the statistical technique known as *PCA*. The second part describes how it is used in the design method that is the outcome of this research.

4.2.1 Theoretical framework

The training set required by any learning algorithm employed for building a multi-layer perceptron network, that is to be applied to a classification task, has the form:

$$\mathbf{T_P} = \{(x^1, o^1), (x^2, o^2), ..., (x^P, o^P)\}$$

where each x^i corresponds to an $n - dimensional$ input vector and, o^i is its $m - dimensional$ expected output. Subtracting all the available inputs from $\mathbf{T_P}$ it is possible to create the following sample set:

$$\mathbf{S_P} = \{\vec{x}^1, \vec{x}^2, \cdots, \vec{x}^i, \cdots, \vec{x}^P\} \tag{4.1}$$

where each $\vec{x}^i$ corresponds to an independent observation of the $n - dimensional$ random vectors referred to in $\mathbf{T_P}$.

A reduction in the dimensionality of the elements contained in the sample set $\mathbf{S_P}$, will be accompanied by a loss of some of the information which discriminates between different classes (or, more specifically, which determines the expected target values o^i). The goal in dimensionality reduction, must therefore be to preserve as much of the discriminatory information as is possible. The manner *PCA* has been designed so as to achieve this kind of dimensionality reduction, comprises the application of a particular type of linear transformation.

The goal in the process is to map vectors $\vec{x}^i$ in an $n-dimensional$ space $(x_1^i, x_2^i, ..., x_j^i, ..., x_n^i)$, onto vectors $\vec{z}^i$ in an $l-dimensional$ space $(z_1^i, z_2^i, ..., z_l^i)$ where $l << n$. The first step therein, involves representing each vector $\vec{x}^i$ as a linear combination of a set of n orthonormal vectors, $\vec{u}_j$

$$\vec{x}^i = \sum_{j=1}^{n} z_j^i \vec{u}_j \tag{4.2}$$

where the vectors $\vec{u}_j$ satisfy the orthonormality relation,

$$\vec{u}_k^T * \vec{u}_j = \delta_{kj} \tag{4.3}$$

in which δ_{kj} denotes the usual Kronecker delta symbol, [Bishop 95],

$$\delta_{kj} = \begin{cases} 1 & : \quad k = j \\ 0 & : \quad otherwise \end{cases}$$

Explicit expressions for the coefficients z_j^i in 4.2 can be found by using 4.3 to give:

$$z_j^i = \vec{u}_j^T * \vec{x}^i \tag{4.4}$$

which can be regarded as a simple rotation of the coordinate system from the original $\vec{x}^i s$ to a new set of coordinates given by the $\vec{z}^i s$. If only a subset $l << n$ of the basis vectors $\vec{u}_j$ is retained, so that only l of the coefficients z_j are used, then the remaining coefficients are replaced by constants b_j so that each original vector $\vec{x}^i$ is approximated by an expression of the form

$$\tilde{x}^i = \sum_{j=1}^{l} z_j^i \vec{u}_j + \sum_{j=l+1}^{n} b_j \vec{u}_j \tag{4.5}$$

which represents a form of dimensionality reduction since the original vector $\vec{x}^i$ which contained n degrees of freedom must now be approximated by a new vector $\vec{z}^i$ which has $l << n$ degrees of freedom. Considering the whole sample set, $\mathbf{S_p}$, in PCA the basis vectors $\vec{u}_j$ and the coefficients b_j must be chosen so that the approximation given by 4.5, with the values of z_j^i determined by 4.4 give the best approximation to the original vector $\vec{x}^i$ on average for the whole data set. The error in the vector $\vec{x}^i$ introduced by the dimensionality reduction is given by

$$\vec{x}^i - \tilde{x}^i = \sum_{j=l+1}^{n} (z_j^i - b_j) * \vec{u}_j \tag{4.6}$$

In PCA the best approximation is taken to be that which minimises the sum of the square errors over the whole data set. Therefore the expression that is minimised corresponds to:

$$E_l = \frac{1}{2} * (\sum_{i=1}^{P} \|\vec{x}^i - \tilde{x}^i\|^2) = \frac{1}{2} \sum_{i=1}^{P} \sum_{j=l+1}^{n} (z_j^i - b_j)^2 \tag{4.7}$$

where the orthonormality relation of 4.3 holds. Setting the derivative of E_l with respect to b_j to *zero* it turns out that

$$b_j = \frac{1}{P}\sum_{i=1}^{P} z_j^i = \vec{u}_j^T \overline{\mathbf{x}} \tag{4.8}$$

where the mean vector $\overline{\mathbf{x}}$ is computed by

$$\overline{\mathbf{x}} = \frac{1}{P} * (\sum_{i=1}^{P} x_1^i, \sum_{i=1}^{P} x_2^i, \cdots, \sum_{i=1}^{P} x_n^P) \tag{4.9}$$

using 4.4 and 4.8 it is possible to write the error sum of squares 4.7 as

$$E_l = \frac{1}{2}\sum_{j=l+1}^{n}\sum_{i=1}^{P} \vec{u}_j^T (\vec{x}^i - \overline{\mathbf{x}})^2 = \frac{1}{2}\sum_{j=l+1}^{n} \vec{u}_j^T \Sigma \vec{u}_j \tag{4.10}$$

where Σ is the covariance matrix of the sample set $\mathbf{S_p}$ and is given by:

$$\Sigma = \sum_{i=1}^{P} (\vec{x}^i - \overline{\mathbf{x}})(\vec{x}^i - \overline{\mathbf{x}})^T \tag{4.11}$$

The final step in the *PCA* processing involves the task of minimising E_l with respect to the choice of basis vectors $\vec{\mathbf{u}}_\mathbf{i}$. It is shown in [Bishop 95] that the *minimum* occurs when the basis vectors satisfy:

$$\Sigma \vec{u}_i = \lambda_i \vec{u}_i \tag{4.12}$$

so that they are the eigenvectors of the covariance matrix. Substituting 4.12 into 4.10, and making use of the orthonormality relation 4.3, the value of the error criterion at its *minimum* has the form:

$$E_l = \frac{1}{2}\sum_{i=l+1}^{n} \lambda_i \tag{4.13}$$

Thus, the *minimum* is obtained by choosing the $n - l$ smallest eigenvalues, and their corresponding eigenvectors, as the ones to discard.

The linear dimensionality reduction procedure derived above is called the *Karhunen-Loéve transformation* or *PCA* and is discussed at length in [Jolliffe 86]. Each of the eigenvectors $\vec{u}_j$ is called a *principal component.* The technique is illustrated graphically in Figure 4.1 for the case of a sample of data points in two dimensions.

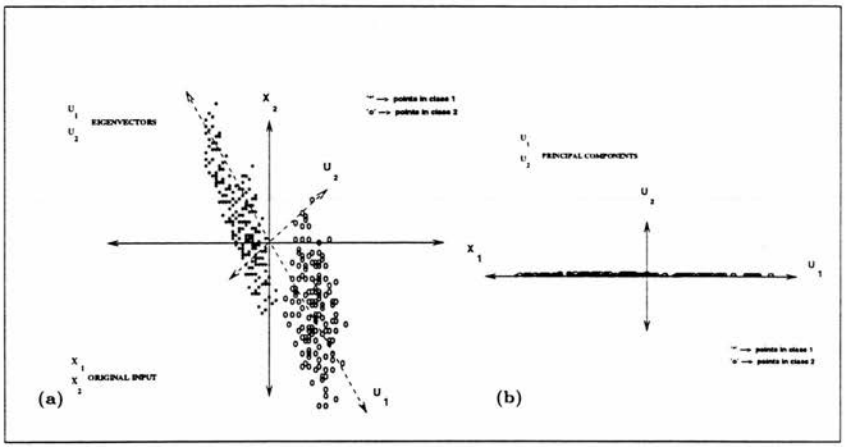


Figure 4.1: **Inappropriate data structure for *PCA* processing**

The first step in the algorithm defined to implement this technique, involves the calculation of the mean and covariance of the sample set that is provided as input. The set of eigenvectors and eigenvalues subsequently derived from the covariance matrix, automatically satisfy the orthonormality condition mentioned before given that the matrix from which they are obtained is symmetric and real. The last step involves the selection of the l eigenvectors associated to the l largest eigenvalues and, projecting the vectors $\vec{x}^i$ of the sample provided as input onto the space spanned by them . Thus in Figure 4.1.a each 2-dimensional data point is transformed to a single variable z_1 representing the projection of the data point onto the eigenvector $\vec{u}_1$ as shown in 4.1.b.

The example depicted in figure 4.1 reflects the kind of problem that can result if *PCA* is applied in a pre-processing step of the complete network design process for the multi-layer perceptron with the aim of reducing the dimension of the input space. The sample plotted is comprised of points belonging to two classes, each associated with one of the two clusters of 4.1.a which, because of the relative positions they have, reflect a mapping that would be easily grasped by a perceptron network, *ie.* no hidden units are required in the architecture. In complete contrast, the space spanned by the highest eigenvector shows a cloud of points in which the elements of the two classes involved are definitely not linearly separable therefore, designing a network architecture to grasp the form of the projection of the initial input space would require a hidden layer which means that the pre-processing step instead of facilitating the design of an adequate network architecture, has hampered it.

Nevertheless, it is possible to exploit the information provided by *PCA* in ways that will be completely effective, in the sense that no matter the form of the problem at hand, its application will always produce a more satisfactory network design process and a more appropriate network architecture.

4.2.2 How *PCA* is used in this study

When designing a multi-layer perceptron network it is possible to identify four spaces all of which are integrated in the network architecture in a meaningful manner, in that it provides a means to solve a task through a simulation process which makes use of the contents of the network. The spaces are: the one comprising the elements of the input space, the one spanned by the weight vectors of the network's connections, the output space and, for those cases requiring a hidden layer, the space spanned by the output of the hidden units.

Many devices have been defined with the objective of handling, the well known problem associated to the task of deciding on the number of units for a network's hidden layer. In the present study, one of the applications that makes use of part of the processing elements comprising *PCA*, is aimed at tackling this same problem. Given that the eigenvalues of the covariance matrix of the input to a *PCA* indicate how relevant the dimensions of such an input are, the same information , subtracted from the space spanned by the output of the hidden units during training, could be used to conclude on whether there are redundant hidden units that could be pruned from the architecture, as this would be reflected in the smallest, *ie.* least important eigenvalues of the covariance matrix.

The important condition that must be satisfied, so that it might be possible to arrive at the type of conclusion described, is that the output of the hidden units that has been registered, be subtracted from a training process that has converged, otherwise the information that would result from a *PCA* on that space would not be very reliable.

The other step of the network design method that incorporates elements associated with *PCA* involves pre-processing of the training set aiming for an efficient execution of the subsequent geometric analysis. The specific application consists in projecting the initial input space to the 3-dimensional space spanned by the eigenvectors associated to the three highest eigenvalues of the covariance matrix. This transformation enables the application of a very efficient implementation of the relevant geometric concepts that exists for low-dimension spaces.

The main operation in the previous description is a projection, therefore it is probable that the subsequent geometric analysis will provide information supporting a more complex network architecture than the one really required. This situation is generated because, the projected

version of an Euclidean space usually has a more complex structure than that of the space from which it was derived. Greater complexity in the input space is dealt with through the definition of a larger number of hidden units in the network architecture. The superfluous elements of the hidden layer are detected and deleted in an additional processing step also based on *PCA* and which was described above. At first instance, it might seem to be the case that this final extra processing should have been avoided but, the considerable gain achieved in the computational costs resulting from the execution of the efficient implementations of the geometric concepts employed, more than justifies their inclusion in the overall multi-layer perceptron design process.

4.3 Alternative applications of *PCA* and Multi-layer Perceptrons

PCA has been used in other ways in conjunction with the multi-layer perceptron model. Each of these alternative applications can be assigned to one of two categories. The first category consists of those techniques defined for computing the principal components of an input space through the application of *NN*. The other is comprised of additional ways of building the architecture of a multi-layer perceptron through the execution of a *PCA* step.

The homogeneous neural network is a *NN* model developed by Oja and his group [1992], incorporating a particular learning algorithm described in [Oja *et al.* 92b]. The main aim behind the model is the parallel computation of the principal components of the sample set provided as input. Even though, the authors solve the asymptotically stable limits of the algorithm, showing that they correspond to the eigenvectors of the data covariance matrix, and argue that their implementation is specially attractive as it opens the possibility of a fast computation of the principal components of an input data set given its potential parallelism, the fact that this input must support a linear model so that the algorithm when training such a network might converge to a practical solution, [Oja *et al.* 92a], [Oja 92], greatly restricts its potential applicability. Moreover there is little advantage to this type of implementation when compared against the standard numerical analysis methods available for performing the same task.

The beginning of the chapter mentions a possible data pre-processing stage for the development of multi-layer perceptron networks. In it, a high-dimension space is reduced to a smaller one that is assumed will be more effectively manipulated by a multi-layer perceptron network, thereby justifying the inclusion of *PCA* in the task of network development. In this context, certain research has been aimed at defining a method for training multi-layer networks so that

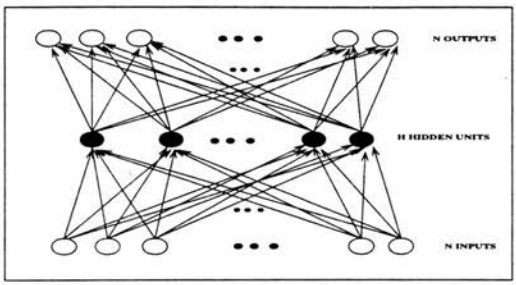


Figure 4.2: **Autoassociative network for implementing** *PCA*

they perform a dimensionality reduction at the hidden layer level, once the training process has converged. The first type of multi-layer perceptron designed for this particular problem has the form shown in Figure 4.2, and corresponds to a network with n inputs, a layer of h hidden units and n output units [Rumelhart *et al.* 86b]. The targets used to train the network are the input vectors themselves, therefore the network is been trained to perform an autoassociative mapping [Hertz *et al.* 91]. For the case of linear hidden units the error function has a unique *global minimum*, and at this *minimum* the network performs a projection onto the $H - dimensional$ space spanned by the first H principal components of the data which, in 4.2, correspond to the weight vectors going into the hidden units, [Bishop 95]. Further studies of the auto-associative network tried to exploit the representational powers of the model using non-linear hidden units. This work was based on the assumption that such an extension to the network model should provide a tool for implementing a non-linear *PCA* but in [Bourlard & Kamp 88], it was proved that when the *minimum* is reached by the learning algorithm training the particular non-linear autoassociative network, the resulting weight vectors once again correspond to the H most important principal components. Given that this alternative implementation of, a dimensionality reduction of the input space generates the same kind of output as the conventional *PCA*, it suffers from the same limitation, *ie.* that the operation might produce a space in which the discriminatory information has been lost, therefore it is not a technique that will invariably produce an improvement in the development of multi-layer perceptron networks.

Another example applying *PCA* to the same area, that belongs to the second of the categories defined at the start of the section, can be found in [Georgiou & C 92]. It describes a learning algorithm developed as an extension of *BP*, named *two-phase BP*. In the first phase the directions of the weight vectors of a network with one layer of hidden units are constrained to remain in the same direction as, either that of the linear discriminants or, the principal components of the training data. In the second phase the restrictions are removed and standard *BP* takes

over to further minimise the error. The handicap of this alternative is that there is no mention in the written work explaining the manner in which the important dimensions of the principal components were selected or of how the bias term of the network model is chosen, something which will be impossible to find if the training data used for building the network is not linear.

Each of the limitations detected in the applications described were carefully taken into consideration at the time of evaluating the possibility of applying *PCA* as part of the network design method defined in the study discussed in this dissertation. Therefore the manner in which it was finally employed can only be considered as an improvement in comparison with the rest of the work developed with the same objective.

4.4 Computational Costs

The first thing that should be noted at this point is that, although it is possible to compute the principal components of a problem domain with a neural network [Oja *et al.* 92a], this alternative method was not considered appropriate in the context of the design process defined in this research for the following reasons:

(a) there are other implementations that are more efficient, *eg.* the one provided in the MATLAB software.

(b) the networks described in [Oja *et al.* 92a], [Oja *et al.* 92b], correspond to variants of perceptrons with no hidden layers. This means that such networks will in effect generate the required eigenvectors if, the problem domain in which they are operating corresponds to a linearly separable one, otherwise no solution will be found.

The implementation of *PCA* used in this research, corresponds to the $EIG(X)$ operator incorporated in MATLAB, the matrix calculator environment developed by MathWorks.

$EIG(X)$ is comprised of a series of routines providing an efficient alternative for generating the eigenvalues and/or eigenvectors of a real general matrix, whilst minimising the possibility of roundoff errors that can result from the computer's calculations. The set of routines included as part of the operator are:

balanc: this receives as input a square matrix of dimension N and replaces it by a balanced matrix with identical eigenvalues. If the input corresponds to a symmetric matrix, as it is already balanced, it is unaffected by this procedure. The procedure is of order N^2 operations.

balbak: the procedure is a continuation on **balanc** that is performed in those cases requiring the computation of the eigenvectors in addition to the eigenvalues of the matrix provided as input to the overall process. It receives the output generated with **balanc** together with a vector D with information on the permutations and scalings performed therein for the problem at hand. With the information it generates the corresponding eigenvectors. It is also an order N^2 routine.

orthes: converts a real general matrix to Hessenberg form using orthogonal similarity transformations. This step produces a reduced matrix from which it is much cheaper to generate the associated eigenvalues therefore resulting in an important gain in efficiency for the overall solution of the eigensystem. The operation count for the routine is in the order of $\frac{5}{6}N^3$.

ortran: builds the matrix of accumulated transformations that were performed whilst executing **orthes** so that the information might be available for the subsequent computation.

hqr2: this routine computes the eigenvalues and eigenvectors of a real Hessenberg matrix using the QR method. As the typical average number of iterations per eigenvalue is ≈ 1.8, the total operation count for all the eigenvalues is $\approx 3 * N^3$.

This set of routines belong to the $EISPACK$ package. The package itself is part of the reusable computer software available to scientists and engineers, for use in mathematical modeling and statistical analysis. Extensive documentation on the particular group of routines employed can be found in [Wilkinson & Reinsch 71] and [Press *et al.* 86].

One thing to note is that, every time the process described is used as part of the design method of multilayer perceptron networks, the input it is provided with, corresponds to the covariance matrix of a sample set. This particular matrix is symmetric, therefore it is already normalised which means that when $EIG(X)$ is executed, it does not perform the first two steps. It is used twice in the whole design method proposed in this research: (a) for reducing the high dimensional input space to a more manageable form on which the subsequent geometrical analysis will work in a more efficient manner and, (b) during the analysis of the outputs of the hidden units that are part of the network architecture. The first application requires the computation of the eigenvectors and the eigenvalues but, as for the second only the eigenvalues are required, its application results in a faster computation.

It is also convenient to mention that, overall the implementation is of order N^3, N the dimension of the input space

As both of the steps in the method proposed, make use of *PCA* to process the covariance matrix of a sample set and, as the matrices are only computed for the purpose of performing the analysis, it was considered reasonable to include the computational costs involved in the generation of this information at this point. The procedure used for calculating the covariance matrix of a particular set of data, is the one underlying the $COV(X)$ operator that is provided in MATLAB. The implementation is based on algorithm 2 and is of order $N^2 * M$, N the dimension of the sample set and M its number of elements.

Algorithm 1 Covariance(Tset,C)

Require: *sample Tset contains inputs of the training set* {there is one such input per row}
Outputs: C covariance matrix of elements in Tset ;
  $[m, n] \leftarrow size(Tset)$;
  if $(min(size(x)) == 1)$ **then**
    $Tset \leftarrow Tset(:)$;
  end if
  $[m, n] \leftarrow size(Tset)$;
  $Tset \leftarrow Tset - ones(m, 1)(\frac{sum(Tset)}{m})$; {ones(m,1) an m-dimensional unitary vector}
  if $(m == 1)$ **then**
    $X\_def \leftarrow 0$;
  else
    $X\_def \leftarrow \frac{Tset' \otimes Tset}{(m-1)}$;
  end if
  $C \leftarrow X\_def$ {**C the computed covariance of the sample Tset**}

4.5 Conclusions

The present chapter discusses the manner in which, the principal components of the training set, that is available for building a multilayer perceptron network, is used in the learning algorithm that resulted from this research. Therein it is possible to identify two ways:

use1: to project the original input space to the space spanned by the three eigenvectors associated to the three largest eigenvalues of the set's covariance matrix.

use2: to evaluate how many of the hidden units that comprise a particular architecture are really useful. This is done by selecting only the number of hidden units that reflect at least 90% of the variance of that space. The output of the hidden units that is analysed is registered once the learning algorithm acting on the network has converged.

The implementation used is of order $O(N^3)$ and corresponds to the fastest algorithm available in the different existing mathematical software libraries. Although, the computation of the covariance matrix is also part of the analysis described and, the procedure associated to this specific operation is $O(N^2 * M)$, with M size of the sample set, the overall cost is still $O(N^3)$

because the existing relation between the square and cubic functions requires an extreme difference between the dimension and size of the training set to vary the $O(N^3)$ behaviour of the implementation more towards a $O(N^2 * M)$ type of performance.

Comparing the manner in which *PCA* is applied to those designed by other authors it is clear that, the one depicted herein is much more effective because, it relies heavily on the existing efficient mathematical *PCA* implementations and, it is not dependent on any kind of ad-hoc decision as, for example, the application associated to the *2-phase BP* process. However, *PCA* is still just a linear transformation and, given the existence of non-linear versions of *PCA* a future extension of this work could be to evaluate how much improvement can be achieved in the network design process if a non-linear version is applied in place of the one used at this moment.

Chapter 5

GROPING

Geometric Reasoning bearing On the Production of Initial Network Geometry

5.1 Introduction

The previous chapters describe the main weaknesses and/or strengths of the multi-layer perceptron model together with those of the most relevant supervised learning algorithms that have been defined to build such networks. Subsequently an analysis of different theoretical sources that provide tools for surmounting the most relevant of those failings is established, including detailed explanations supporting the set that was finally selected to improve the design process of a multi-layer perceptron network.

In the present chapter all those useful devices are integrated in a coherent framework and linked with one of the existing learning algorithms, *ie.* rpropagation [Riedmiller & Braun 93], resulting in a very effective generative supervised learning method, *ie.* the **G**eometric **R**easoning bearing **O**n the **P**roduction of **I**nitial **N**etwork **G**eometry (**GROPING**) process. Given that it is comprised of a series of steps involving different types of operations, the ensuing description has been organised so that it might be clearly understood, how each of them enhances the process of constructing a multi-layer perceptron network.

Section 5.2 includes a general outline of the **GROPING** algorithm describing its main components and how they are integrated so that the whole process shows a coherent behaviour at the moment of executing it to design a network for a particular application. Section 5.3 explains how **GROPING** in effect solves some of the main problems associated with the task of designing a network architecture. Section 5.4 is comprised of an analysis of the computational costs of the whole procedure and Section 5.5 summarises the main points delineated in the

chapter.

5.2 Outline of GROPING

Certain of the conclusions reached in chapter 2, after discussing the different approaches that have been taken in the multi-layer perceptron research area to solve the problems encountered using it, referred to the advantages of applying generative supervised learning, over conventional supervised learning. These advantages can be summarised in the fact that for the first type of algorithms it is not necessary to define a reasonable initial structure because, part of the processing involves constructing the network's architecture to satisfy the requirements of the application of interest.

GROPING was designed in accordance with the main principles of that type of algorithm which, as noted in chapter 2, can operate in one of two ways: begin with a minimal network, *ie.* one comprised only of the units corresponding to the inputs and outputs and subsequently growing it until the overall performance is satisfactory; alternatively start with an architecture too large for the problem under consideration and the design process will then involve trimming the network to a reasonable *minimum* size. The difficulty with the second approach is that it is necessary to provide as input an initial **maximal** network solution, *ie.* a network architecture that is too large for the application of interest, a task which in itself can be quite complicated. Nevertheless, if it is at all possible to find one, the design process of the required architecture is greatly simplified because, there is reliable information confirming the hypothesis that it is possible to use the network model to build a solution for the problem of interest. The complex architecture that is provided as input to the training component of the network design process, is evidence of the feasibility of achieving a solution, something that cannot be ascertained in a similar manner with the method that starts from a minimal architecture. In chapter 3 it is shown how the Delaunay triangulation and the Voronoi diagram of the points comprising the training set are sound sources for the process of creating an initial maximal network therefore, having found an effective manner to satisfy the main constraint associated with the pruning-generative supervised learning methods, it was possible to design **GROPING** as an algorithm of this kind.

Once the theoretical framework underlying the algorithm that is going to be designed has been clearly specified, the next step requiring careful consideration is the way in which the available data about the problem is going to be organised so that it might be used in the most effective manner. In conventional supervised learning it is usually divided in two sets, *ie.* the training and test sets. The first one is employed for updating the weights of the

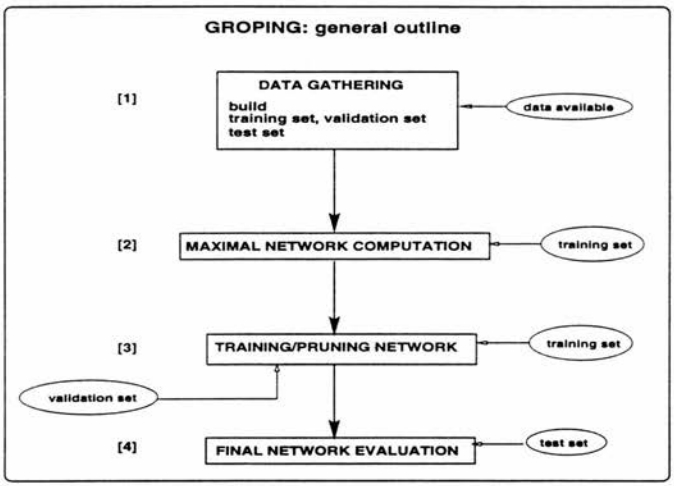


Figure 5.1: **General outline of the GROPING algorithm**

connections of the architecture during the execution of the algorithm. The other one, *ie.* the test set, is used for measuring the level of generalisation achieved with the network once the training process is completed. This manner of structuring the data available to build a network was modified to avoid the overfitting (overtraining) phenomenon: for two networks trained on the same problem, the one with larger training set error may actually be better, since the other has concentrated on peculiarities of the training set at the cost of losing much of the regularities needed for good generalisation [Prechelt 94]. For this reason, another alternative way of partitioning the available data was created, one which included an additional division of the training set providing the whole network design process with three types of element against which to measure the appropriateness of the network being created. These are: the training set, the validation set and the test set. The first one, as in the previous case, is used to compute the amount by which each of the network's connections should be updated to reduce the error; the validation set is provided to allow control over the overtraining phenomenon and, the test set is used to check on the generalisation level achieved by the final network.

The **GROPING** algorithm was developed to operate on a training, validation and, test set. All the available data is partitioned so that 50% comprises the training set, 25% is used for the validation set and the last 25% is assigned as part of the test set. The three sets are used in a manner similar to the one described above, the main difference lies in the additional processing that the training set undergoes so as to build the initial maximal network architecture. Figure

5.1 presents a high level description of the whole design process marking which dataset is used as input for each of the main operations. Throughout, it is clear that the most relevant of the sets is the training set but, as will be subsequently shown, the information extracted from it is carefully controlled so that the final network architecture will not be over-accommodated to its peculiarities. The amount of available data associated with each of the sets required for constructing the network, was assigned according to the indications contained in the **Proben1** benchmark which corresponds to the method followed in this study to evaluate the network algorithm defined and, given that the results obtained with **GROPING** were compared against those presented in **Proben1**, it was necessary to proceed with exactly the same datasets.

Another relevant issue affecting the overall performance of a multi-layer perceptron's network training process consists of the correct definition of the different parameters related to the training component of the design process. In chapter 2, the examples presented showing how different values of the learning rate (η) and the momentum term (μ), affect the evolution of the network training task are a clear indication of how important it is, for achieving a good performance, *ie.* a good level of generalisation with a moderate architecture trained in a reasonable number of iterations, to correctly tune these parameters. In view of their relevance, the one required to ensure a correct performance of the training algorithm are defined at the moment of creating the network and no pre-processing needs to take place in advance of its application. This is a major enhancement as it means that the elements responsible for producing an *ad-hoc* procedure have been successfully dealt with.

The particularities of **GROPING** will be discussed following the outline depicted in the diagram displayed in figure 5.1, which shows that the method is comprised of four major operations:

(1) corresponds to the organisation of the data available about the problem. The operation consists in ordering it in a form that is acceptable to the input's specifications of a network model. The elements of the different problem's samples must be organised in pairs: (x^p, o^p), where $x^p \in R^N$ and, $o^p \in B^m, B^m = \{0,1\}^m$. The form of B^m corresponds to that of the binary patterns that reflect a classification task and, each x^p is an instance of the problem space that must be associated to one of the possible classes related to it. The second part of the operation consists of assigning each of the (x^p, o^p) pairs to one of the sets needed for the task of designing the multi-layer peceptron network. Care must be taken to clearly mark any missing values and also to avoid duplicates as they both have a negative effect on the process of developing a network. For example, at the moment of computing the weight vectors of the initial architecture the operations involved require that all values of the components of the training set be clearly specified, because they

have not been designed to manipulate missing information.

(**2**) consists of the steps which result in the initial maximal network architecture required to solve the problem. It uses geometrical and statistical information, for deciding on the number of nodes and connections that should constitute the network architecture and, the initial values of the weights associated with those connections. Additionally it is comprised of steps to deal with missing information because, for the particular network model under consideration the input to the learning algorithm must be complete. Finally it includes a step for transforming the data to zero mean and unitary variance because various experiments performed at the beginning of the research have shown that the learning process generally produces better results when dealing with the transformed data and, there is a well specified mechanism to perform this type of transformation.

(**3**) this is probably the most relevant component of the whole design method, even though it is certainly the case that, its successful completion is very dependent on the output produced by the previous step because, if an appropriate maximal network is not available it is futile to proceed with its execution. The task is a combination of a multi-layer perceptron learning algorithm, rpropagation and, the statistical method principal component analysis. This last procedure provides information to decide on the number of superfluous hidden units. The learning algorithm uses the training set to compute the quantities by which each of the weight's connections must be updated and, the validation set to decide at the end of the training process which is the most convenient architecture for the problem at hand.

(**4**) this operation consists of computing the error made by the final network architecture generated by the previous step when presented with the elements of the test set. It can be considered to be mostly a simple performance measure that enables the network developer to come to a final decision on the success of the whole training process.

Given that operations two and three involve a series of steps requiring further explanation, each of them will be discussed in greater detail in the ensuing sections.

5.2.1 Building the maximal network architecture

This operation is mainly a data processing task which extracts the information from the training set that is needed to perform a successful definition of the multi-layer perceptron network required to solve the problem from which the set was obtained. The specific components of the problem's solution that are built with the information extracted are the following:

(**1**) the mean vector and covariance matrix that will be used for the transformation of the data to mean zero and unitary variance (covariance in an N-dimension space, N > 1).

(**2**) the number of nodes comprising the architecture.

(**3**) the values for the weight vectors of the input-hidden connections of the initial maximal network.

Algorithm 2 DataStandardisation(Tset,P,N,StandardTset)

Require: $Tset$ {elements of the training set}
Require: $P = size(Tset)$
Require: $N = dimension(Tset)$;
Outputs: Normalised data in StandardTset;
$Cts \leftarrow covariance(Tset, P, N)$; {compute sample's covariance matrix }
$[RtsDts] \leftarrow eigen(Cts)$; {Rts: eigenvectors of Cts; Dts: eigenvalues of Cts }
$Mts \leftarrow mean(Tset)$;
$V \leftarrow Tset - ones(P, 1) * Mts$; {extract mean from original input}
$V1 \leftarrow V * Rts * inverse(sqrt(Dts))$;
rotate using the eigenvectors of the covariance
matrix scale by the inverse of the standard deviation;
$StandardTset \leftarrow V1$;
{conditions satisfied by StandardTset};
$mean(SttSet) == zeros(N)$;
$covariance(StandardTset) == identity(N)$

Assuming that the training set has the form:

$$T_P = \{(x^1, o^1), (x^2, o^2), \ldots, (x^i, o^i), \ldots, (x^P, o^P))\} \tag{5.1}$$

where, i indexes the occurrence of x^i in the training set, a pair (x^i, o^i) is comprised of an input pattern x^i and its expected output o^i and, the pattern $x^i \in R^N$ is described by the tuple:

$$\mathbf{x}^{(i)} = (x_1^i, x_2^i, \ldots x_N^i) \tag{5.2}$$

the mean vector and covariance matrix needed for standardising the data are estimated in the following manner:

$$\overline{\mathbf{X}} = \overline{x}_1, \overline{x}_2, \ldots, \overline{x}_N \tag{5.3}$$

$$\overline{x}_j = \frac{\sum_{i=1}^{P} x_j^i}{P} \tag{5.4}$$

$$c_{jk} = \frac{\sum_{i=1}^{P} (x_j^i - \overline{x}_j)(x_k^i - \overline{x}_k)}{(P-1)} \tag{5.5}$$

Equation 5.5 corresponds to the expression used for computing each of the terms of the training set's covariance matrix and, $\overline{\mathbf{X}}$ is the N-dimension mean vector, each of whose terms is calculated using equation 5.4. Both terms are subsequently used in algorithm 2 to standardise the data of the training set. As the transformed data is the one that is provided as input to

Algorithm 3 BuildInitialNetwork(Tset,P,N,DefWeights)

Require: $Tset$
Require: $P = size(Tset)$
Require: $N = dimension(Tset)$;
Outputs: Number of hidden units, weights in DefWeights;
(1) $M \leftarrow zeros(N)$;
(2) $C \leftarrow identity(N)$; {initialise structures for storing Tset's mean M, covariance C}
(3) $[R, D] \leftarrow eigen(C)$; {compute eigenvectors, eigenvalues of C}
(4) $Pset \leftarrow project(Tset, R(:, 1 : 3))$; {project total space to 3-dimensional subspace spanned} {by three highest eigenvectors of C}
(5) $SetWeights \leftarrow \emptyset$; {initial set of weights is empty}
(6) $DT \leftarrow BuildDelTri(Pset)$; {build Delaunay triangulation of projected 3-dim set}
(7) $SetWeights \leftarrow ComputeWeights(DT)$;
use geometric construct in (N+1)-dimension to build weight vectors
(8) $DefWeights \leftarrow SubtractParallel(SetWeights)$; {extract the sufficiently parallel vectors}

the neural network, the computation of the initial maximal network architecture is realised through a geometric analysis of this version of the training set.

This analysis produces the network architecture which will be subsequently trained and pruned and will eventually result in the final network that reflects the mapping underlying the classification problem of interest. Algorithm 3 corresponds to a high level description of the most important steps associated with this task. The reasoning underlying each of them runs as follows:

(1) Given that the Delaunay triangulation is generated with a geometric algorithm, as was discussed in chapter 3, it shows exponential behaviour on the dimension of the space when, dimension > 3. To avoid these costs and take advantage of the efficient implementations that exist for 2-dimension and 3-dimension spaces, it is necessary to adjust the weight initialisation process so that it would exploit the information subsumed in the Delaunay triangulation of the set of points calculated by one of these efficient implementations. The structure of a space is never improved for purposes of my method when it is projected because, as this type of transformation involves squeezing the data into a smaller number of dimensions, there are points in the original input space that will map to the same image or, to points that will be far closer to each other than their counterparts in the original space, increasing the difficulty of discriminating between them. This could be considered a drawback given that the multi-layer perceptron network is being designed to operate as a discriminator but, as the aim at this point is to build a maximal network architecture and, a geometric analysis of this more complex space can only result in the generation of more hidden units than the number really required for the classification task in the original input space, it seemed reasonable to reduce the input to the 3-dimension

space spanned by the three eigenvectors of the data's covariance matrix and, compute the Delaunay triangulation of the projected dataset applying one of the very efficient algorithms that have been defined to process low dimension spaces. The eigenvectors associated with the three highest eigenvalues of the covariance matrix, which correspond to the three most important principal components of the data, was judged to be the most appropriate basis because, it retains the highest amount of information regarding the peculiarities of the space from which it was computed. Steps **3**, **4** of algorithm 3 produce the projected data set on which the geometric analysis will subsequently operate.

(**2**) The execution of steps **6**, **7** in the network building procedure initialise a set structure which will be instantiated with the Delaunay triangulation derived from the points of the projected space. The geometric process is performed using the qhull system developed by Barber and his research group, [Barber *et al.* 95], and which is an $O(N \log(N))$ implementation of the process for generating the Delaunay triangulation and Voronoi diagram of points in 2 and 3 dimension spaces. The reasons for selecting this particular implementation were given in chapter 3.

(**3**) The final operation indicated in the algorithm, corresponds to the extraction of the weight vectors of the initial network architecture from the Delaunay triangulation generated from the projected set of points of the training set. It is based in the solution of a set of (N+1)-dimension linear systems, one for each of the weight vectors considered to be necessary in the input-hidden layer of the maximal network that is being computed. As it is the central task of the building process, the steps comprising it are expanded in the ensuing discussion.

Algorithm 4 contains the operations designed to compute the set of weight vectors required for the initialisation of the maximal network architecture. Most of them are basic operations easily interpreted once the rationale of the algorithm is understood. The main loop traverses the facets of the convex hull associated with the Delaunay triangulation obtained from the projected points of the training set. Given that the input to the qhull algorithm, in this particular case, is always in 3-dimensions, the structure that the weight generation process receives is usually a convex hull created in a 4-dimension space. This means that each of the facets that will be traversed in the main loop of the weight generation process will be comprised of a group of interconnected points organised and ordered according to the definition of a convex polyhedra, [O'Rourke 94].

The derivation of the weights for the input-hidden layer of a network operating in the original N-dimension input space, $N > 3$, requires the computation of a set of (N+1) dimension vectors were the additional dimension corresponds to the threshold associated with each hidden unit.

Algorithm 4 WeightVectorsComputation(Dt3D,N,W,NumW)

Require: 3-dimensional data's DELAUNAY TRIANGULATION Dt3D;
Require: N: dimension of the original space;
Outputs: W set of weights;
Outputs: NumWts number of vectors generated;

```
{variables required in the procedure}
M is a matrix(N + 1, N + 1); B is a vector(1, N + 1);
Wtmp is a vector(1, N + 1);
{variable initialisation}
M ← zeros(N + 1, N + 1);
NumEq ← 0; Wtmp ← zeros(1, N + 1); NumW ← 0; F ← FirstFacet(Dt3D);
while F ≠ NULL do
  F ← NextFacet(Dt3D); {set of points included in facet F}
  if ¬(all(P) ∈ SameClass) then
    not all points in F belong to same class; might require hidden unit
    for edges(F) do
      look for edges with endpoints in different classes
      if ¬(endpoints(e) ∈ SameClass) then
        (p1, p2) ← endpoints(e);
        if ¬(p1 ∈ M) then
          NumEq + +; M(NumEq, :) ← p1; o ← ExpectedOutput(p1);
          if ¬(o) then
            B(NumEq, 1) ← log(9);
          else
            B(NumEq, 1) ← − log(9);
          end if
        end if
        if NumEq == (N + 1) then
          break;
        end if
        if ¬(p2 ∈ M) then
          num_eq + +; M(num_eq, :) ← p2; o ← ExpectedOutput(p2);
          if ¬(o) then
            B(num_eq, 1) ← log(9);
          else
            B(NumEq, 1) ← − log(9);
          end if
        end if
        if NumEq == (N + 1) then
          break;
        end if
        e ← NextEdge(F);
        if e==NULL && NumEq ¡ (N+1) then
          LookNeighbours(F, M, NumEq, N + 1);
        end if
      end if
    end for
    if NumEq == (N+1) then
      Wtmp ← LinearSystem(M, B);
      if ¬infinity(Wtmp) then
        NumW ← NumW + 1; W(NumW, :) ← Wtmp; M ← zeros(N + 1, N + 1);
      end if
      F− > process ← TRUE; B ← zeros(1, N + 1); num_eq = 0;
    end if
    F ← NextFacet(Dt3D);
  end if
  closes if not(all(P) in SameClass)
end while
closes while (F != NULL)
```

These are derived from the data structure generated by qhull to store the Delaunay triangulation of the 3-dimension points it received as input. It consists of vertices, ridges and facets. Each vertex corresponds to one of the 3-dimension points obtained from the projection of the N-dimension training set. A ridge is a set of 3 vertices and two neighbouring facets and, a facet is a set of ridges, a set of neighbouring facets, a set of incident vertices, and a hyperplane equation. Of all these components the ones considered most relevant to the computation of the required set of weight vectors, corresponds to the set of facets F because, they provide the most adequate source to obtain the information on the adjacency relations which is the key element in the conception of the necessary linear systems. These systems are computed in the following manner:

(1) If the facet F is comprised of points corresponding to projections of their counterparts which, in the original N-dimension space have different expected outputs o^i then, given that they are connected because they are closer to each other than to any other of the elements in the training set, it is reasonable to assume that a hidden unit will be required to generate two regions, one for each of these conflicting points.

(2) The edge connecting these conflicting points and, the facet F to which they belong, provide useful information for computing the hyperplane that will generate the regions desired. It can be calculated by solving a linear system that is defined using the set of conditions the related points should satisfy. The points are those belonging to the facet F under analysis and also to those included in the neighbouring facets which have to be taken into account because the system must be defined in the original dimension and this is always greater than the one associated with the convex hull on which the procedure is at present operating. The conditions are based on the outputs the network should compute when presented with any of the points under consideration. Taking into account that:

   (a) the activation function of any unit, apart from the inputs, is the sigmoid,

   (b) the outputs can only have one of two possible values, $\{0, 1\}$, in the case of the points in the training set,

   (c) the midpoint of any edge connecting two input points having different expected outputs, can be computed as $\frac{(x^i+x^j)}{2}$ and, if presented to the network it should generate as output an approximate value of 0.5 because of its position in the convex hull,

   Only three types of equations can be added to the linear system that is being built. These will reflect a set of conditions that should be satisfied by the following variables:

(a) the weight vector that requires calculation:
$W^{num\_eq} = (w_1^{num\_eq}, w_2^{num\_eq}, \ldots, w_{(N+1)}^{num\_eq})$,

(b) the threshold value $w_{num\_eq}^{(N+1)} = -1.00$, tolerance $= 0.1$,

(c) the input vector $x^i = (x_1^i, x_2^i, \ldots, x_N^i)$

which means they will have one of the following forms:

$$\sum_{j=0}^{N+1} w_j^{num\_eq} x_j^i = -\ln(9), \quad if \quad (o^i = 0) \tag{5.6}$$

$$\sum_{j=0}^{N+1} w_j^{num\_eq} x_j^i = \ln(9), \quad if \quad (o^i = 1) \tag{5.7}$$

$$\sum_{j=0}^{N+1} w_j^{num\_eq} x_j^i = 0.0, \quad if \quad (o^i = 0.5) \tag{5.8}$$

(3) Each of the vectors obtained through the solution of the linear systems construed in the main loop of the algorithm, constitutes a weight vector of the input-hidden layer and will require the incorporation of an additional unit in the hidden layer of the network architecture.

The most fundamental piece of information subsumed in the convex hull provided as input corresponds to the neighbouring relation that exists between the different geometric components that comprise it because, this is the information used for deciding whether or not to incorporate an equation to the linear systems that are being computed. In the output of the qhull implementation used it is possible to detect this type of relation between facets and between ridges and facets. Each time the vertices comprising one of the hull's components are incorporated into a system that results in a valid weight vector, the component is marked as used so that it will not be selected for the computation of any other weight vector. This constitutes a first step towards the control of the creation of redundant units in the hidden layer.

A final control mechanism is realised at the end of this process, whereby all the weight vectors that have been computed are taken in pairs to produce subsets of them that are sufficiently parallel so that only one vector, *ie.* a group representative, is incorporated in the final network architecture. Various examples, some of which are summarised in appendix E, were performed to provide an upper limit that should be satisfied by the angle θ computed from the weight vectors w_i, w_j that would render them sufficiently parallel and, as in most cases, it appeared reasonable, in computational terms, to conclude that any two vectors were parallel enough if $\theta < \frac{\pi}{4}$, this is the limit used in **GROPING** for deciding on this matter. Finally those weight vectors considered to be sufficiently parallel are subjected to an additional analysis because it might be the case that, even though the angle between the two vectors renders them

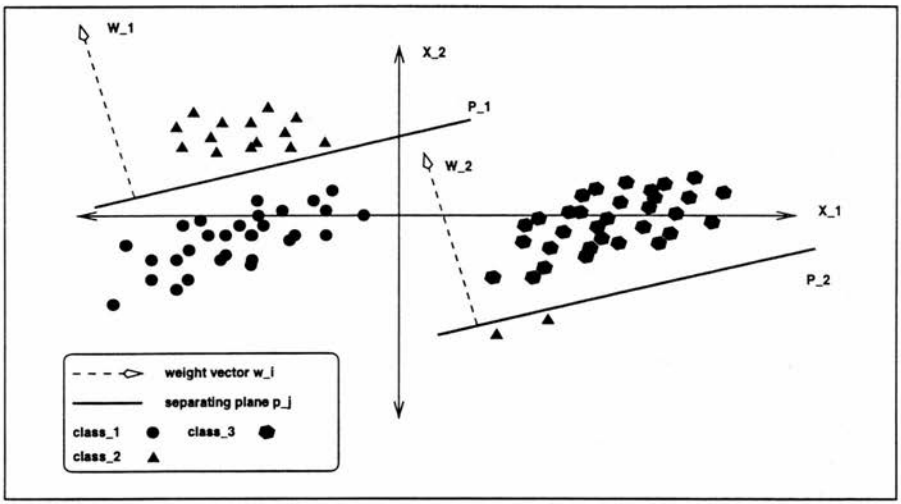


Figure 5.2: **Example of sufficiently parallel and useful weight vectors**

sufficiently parallel, as they are located in completely different regions of the input space and, as each is generating useful discriminatory information in the region in which it is set, neither should be eliminated from the final pool of weight vectors. This proximity criteria is realised by computing the distance between the two weight vectors and checking whether it is larger than $(\frac{2}{3})\,MAXIMUM\_DISTANCE(train\_set)$; if the answer is positive then, even though the angle between them is low, neither of them is extracted from the final network architecture. The upper limit on the distance parameter also corresponds to an empirical boundary obtained from a pool of experiments performed on different training sets using different values for this measure. The training set and weight vectors displayed in figure 5.2 exemplify the problem; w_1 and w_2 are parallel enough but if either of them is eliminated, the input space ceases to be correctly partitioned.

The elements of the initial maximal network architecture completely specified at the end of this process are:

(a) The units of the input and output layer;

(b) whether or not a hidden layer is necessary, if one is indeed required the number of hidden units that should be included and;

(c) the complete set of the weight vectors associated with the connections of the input-hidden layer. These weight vectors are all scaled to a unitary length. This additional

processing ensures low values at the start of the training process which will guarantee a properly behaved learning because, it greatly reduces the possibility of getting stuck in a local *minimum* at the start of the training. The weighted sum of inputs, $\sum_{j=0}^{N} w_{ij}\, x_j^p$, that is the sigmoid's input, will not have very large values given that the training set is standardised and the weight vectors have unitary length.

Before proceeding with the next stage of the **GROPING** method it is necessary to calculate the weight vectors of the connections of the hidden-output layer so that the maximal network is completely specified. Given that, more than half of the architecture at this point has been decided upon only taking into account the structure of the training set, to reduce the over-training phenomena, thereby easing the subsequent pruning procedure, the weight vectors of the hidden-output layer were randomly initialised with values in the $[-1.00, 1.00]$ interval.

5.2.2 Evolving-pruning the network of interest

Even though, all the steps depicted in figure 5.1 play a relevant and necessary role in the task of achieving an adequate multi-layer perceptron network within the framework of pruning-generative learning, in the literature reviewed the discussions regarding this type of learning are restricted to the pruning operation because, most of them assume that the learning begins with an already available maximal network architecture.

This restricted view of what constitutes pruning-generative learning should be revised because, the task of building the initial maximal architecture is equally, and probably harder, than the operation of trimming, reducing its size. Nevertheless, in the context of the **GROPING** algorithm, the activities that perform the pruning task have been labeled as the learning component of the complete design process so as to comply with the general, although incorrect view, associated with the type of solution that is being proposed.

The diagram in figure 5.3 contains the sequence of pertinent processing involved. The algorithm is controlled by a main loop that consists of training a particular architecture until it achieves complete recognition of the training set subject to an allowed error limit. At the end of this process, the output of the hidden units is computed using the final set of weight values generated in the training step. These vectors become the input to a principal component analysis, the operation that has been incorporated for deciding on the network's superfluous hidden units. Subsequently the units found to be redundant are eliminated and, the task of learning to recognise the training set is re-initiated but, with the new reduced network architecture. The process stops the moment the network being trained fails to recognise the complete training set after it has been working on it for a maximum number of iterations. Details of each of

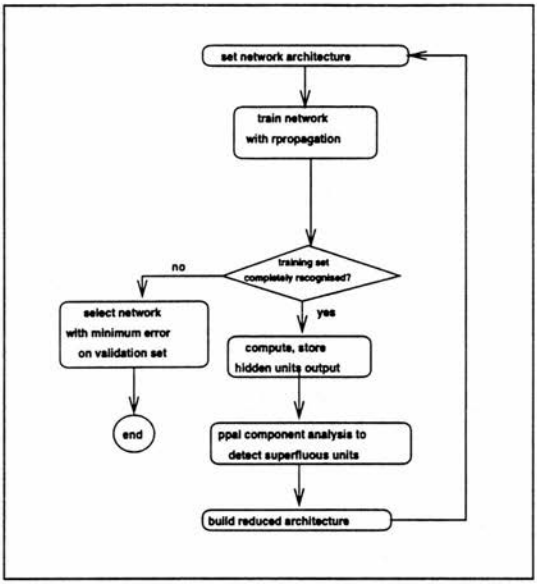


Figure 5.3: **Flow chart of learning-pruning component**

these major tasks are provided in the ensuing discussion.

Training a particular network architecture

For the final adjustment of the weights of the connections in the network architecture under construction, the **GROPING** algorithm includes one of the existing supervised learning algorithms that have been developed for training multi-layer perceptron networks.

The implementation is based on an algorithm described in [Riedmiller & Braun 93]. It is the **R**esilient back**prop**agation, a local adaptive learning scheme, performing supervised batch learning in multi-layer perceptrons, designed to overcome the difficulties found in many applications when using this network model and the well-known learning algorithm, backpropagation. The implementation used in **GROPING** corresponds to the one found in the network simulator **rbp** developed by Donald Tveter [1996] which can be retrieved from the website:

`http://www.mcs.com/~drt/home.html` .

In the context of a generative learning algorithm the execution of **rprop** produces a network architecture that captures the input-output mapping of all the elements in the training set. As

training proceeds two errors are registered, the one associated with the training set and, the one associated with the validation set, this last corresponds to the information employed to control the over-training phenomena.

The training process halts for one of two reasons:

(1) the network architecture has completely captured the input-output mapping of the training set.

(2) a **maximum** number of iterations have been executed but it is not possible to capture the input-output mapping of the training set with the present network architecture. Given that this specific operation is taking place within a generative-pruning process, when this situation is encountered it is possible to conclude that a complete learning is not possible because the architecture is not large enough, that is, it does not have enough hidden units to deal with the task.

If the training process halts because all the patterns have been completely recognised, it is reasonable to assume that the network architecture is still too large and can be further reduced. As the pruning task is achieved by analysing the outputs of the hidden units, the last step of the training procedure consists in presenting all the training patterns to the final architecture that **rprop** has computed, registering the output of all the hidden units when the network is presented with each of them. This operation produces a set of P (size of the training set) H-dimensional (H number of hidden units in the present architecture) vectors which will be subsequently used in the principal component analysis process to decide how many of those units can be considered to be superfluous and therefore pruned from the overall architecture.

If the learning is stopped because it has been operating for the **maximum** number of iterations and has not yet achieved a correct classification of all the elements of the training set, in the **GROPING** context it provides reasons to conclude that **rprop** is trying to train an architecture that is not large enough to deal with the present application. The main reason for reaching this conclusion rests in the fact that **rprop** has arrived at the upper limit of the number of iterations. This upper limit, set at 5000 iterations, was defined following the results obtained with a set of the experiments performed in this research.

It corresponds to a study that was realised to evaluate the behaviour shown by the training process when operating on network architectures built applying the geometric analysis described in section 5.2.1. Each network was construed to capture the mapping of one of 28 artificial datasets generated using the DGP-2 software, developed by Powell Benedict, available at the website: `http://www.cogs.susx.ac.uk/users/christ/UCI_repository/dgp-2` .

set_id	max_feature_value	range	percentage
1	2	1	72
2	2	1	67
3	100	15	70
4	100	10	80
5	4	0.5	90
6	1	0.6	75
7	1	0.6	60
8	1	0.1	67
9	38	5.7	80
10	4	1.2	65
11	2	1	50
12	9	1.8	50
13	9	1.8	50
14	2	0.2	80
15	10	1	85
16	10	1	85
17	5	0.4	66
18	1.1650	0.1398	98
19	0.0931	0.014	18
20	150.0254	19.5033	35
21	607.8	81.8178	24
22	1.1650	0.1398	96
23	90.23	6.3161	81
24	0.0529	0.0069	67
25	31670048	2.5336e+06	86
26	4.4602	0.0892	81
27	95.56	13.3784	95
28	67.8865	18.6939	38.35

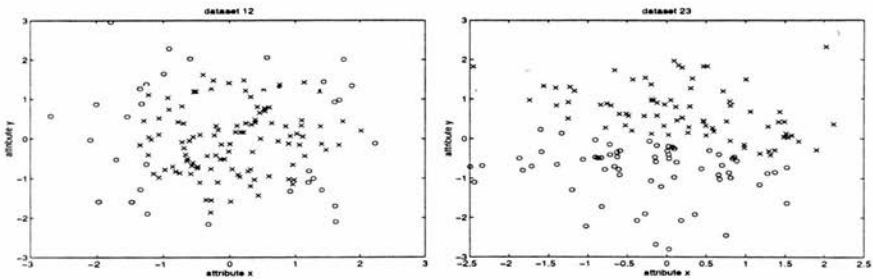
Table 5.1: **Main properties of sets used in hypothesis testing**

The selection of the sample was based on the need to generate a sufficiently large one that would allow a sound statistical inference process, having a manageable degree of complexity and, corresponding to a group of problems for which the multi-layer perceptron model could be considered a legitimate representation form. The characteristics common to all the sets are:

(**1**) the dimensionality and size. All are subsets of 200 points, extracted from the 2-dimension continuous space.

(**2**) All the datasets are examples of a boolean type classification task. This means that the expected output associated with each input instance, corresponds to a one dimension binary vector that indicates if the instance is positive or negative, *i.e.* whether or not, it is an example of the target concept underlying the dataset to which the instance belongs.

The main properties of DGP-2, particular to each set, are presented in table 5.1. Column one is a unique number used for identifying each of the sets. Column two is the max_feature_value and



(a) (b)

Figure 5.4: **Form of two problem domains: (a) dataset 12, (b) dataset 23**

it is a parameter that tells DGP-2 which is the maximum value that might be assigned to any of the features of the dataset, all of which will be generated in the range $[0, max\_feature\_value]$. The contents of the third column, *i.e.* the range, specifies a range around the centroids, within which instances will be positive. This range along with percentage (number of positive instances requested), determines the standard deviation value in the normal distribution function. The fourth column contains the values assigned to the percentage parameter in each of the datasets. Varying these two parameters it is possible to produce data that will be easier(harder) to process with a network. A larger range with respect to max_feature_value and a greater number of centroids induces a more complex architecture because, the task of deciding if an element is positive or negative becomes more ambiguous.

Figure 5.4 shows the manner in which the points of two of the artificial datasets generated for this particular analysis are positioned. Plot 5.4.a corresponds to the training patterns of the data included in set 12 and, plot 5.4.b shows those that comprise the training component of set 23. The organisation of the points included in dataset 12 induced a harder training task because the boundaries between the positive and negative instances are less defined, something that is evident in the results exhibited in table 5.2. On the other hand, it was relatively easy to train a network to model the form of dataset 23, something that is not surprising given the clear delimitation there is between its positive and negative instances.

The choice of approach was made considering that at this point of the research, the only means to reach reliable conclusions concerning a new method of network initialisation, was through a procedure of experimental investigation. To this end, 30 training processes were performed on each of the problems built with DGP-2. In all the problems, each of the training processes was realised on a network architecture randomly initialised in the interval $[-1.00, 1.00]$. The size of the network complied with the size which, according to the geometric analysis, was needed to solve the task represented by the training set being processed.

set_id	percent classified at number of iterations					percent growth_level			
erations	500	1000	5000	10000	15000	500-1000	1000-5000	5000-10000	10000-15000
1	87	92	96	97	97	5	4	1	0
2	100	100	100	100	100	0	0	0	0
3	89	94	96	96	96	5	2	0	0
4	95	95	95	95	96	0	0	0	1
5	100	100	100	100	100	0	0	0	0
6	98	98	98	98	98	0	0	0	0
7	94	95	96	96	96	1	1	0	0
8	79	88	95	95	96	9	7	0	1
9	100	100	100	100	100	0	0	0	0
10	100	100	100	100	100	0	0	0	0
11	95	96	97	97	97	1	1	0	0
12	79	86	93	94	94	7	7	1	0
13	98	98	98	98	98	0	0	0	0
14	100	100	100	100	100	0	0	0	0
15	100	100	100	100	100	0	0	0	0
16	98	98	98	98	98	0	0	0	0
17	100	100	100	100	100	0	0	0	0
18	100	100	100	100	100	0	0	0	0
19	100	100	100	100	100	0	0	0	0
20	97	97	98	98	98	0	1	0	0
21	97	97	97	97	97	0	0	0	0
22	98	98	98	98	98	0	0	0	0
23	100	100	100	100	100	0	0	0	0
24	100	100	100	100	100	0	0	0	0
25	96	96	97	96	96	0	1	-1	0
26	100	100	100	100	100	0	0	0	0
27	98	98	98	98	98	0	0	0	0
28	92	94	94	94	94	2	0	0	0

Table 5.2: **Results in hypothesis testing concerning number of iterations**

The statistical inference used was the hypothesis testing method [Kleinbaum *et al.* 88]. It was centered mainly on the **minimum mean square error** because being an average over a reasonably large set, the central limit theorem holds. Hence, it was possible to perform a process of statistical inference comparing the error achieved by a training process operating on a network initialised with the geometric information against the error achieved by the same training process, operating on the same network architecture but, with weights randomly initialised. Appendix D contains a description of the analysis performed together with the results achieved concerning the error measure which showed, overall, that the training process is greatly improved when it is operating on the network initialised with the geometric information. Given that what is being discussed here concerns the choice in the value assigned to the **maximum** number of iterations, the contents of table 5.2 provide information supporting the one employed in **GROPING**. Each of the rows in 5.2 corresponds to one of the datasets built with DGP-2. The first five numbers in row i are the percentage of elements in the training

set of example i, correctly classified by the training process at different stages of the learning cycle, *i.e.* at 500, 1000, 5000, 10000 and 15000 iterations respectively. The other four columns contain the difference between the percentage level achieved in two subsequent intervals. For example, the values associated with the 10th dataset, indicate that the training set has been completely recognised by iteration 500 therefore, between 500-1000, 1000-5000, 5000-10000, 10000-15000, there is no change in the portion of the training set correctly classified by the learning algorithm. The number of zeroes on the last four columns of table 5.2 clearly show that it is useless to proceed with a training process over 5000 iterations if the training set has not been completely captured by the network at that stage. Furthermore, the training could be stopped at 500 iterations, but as there are a few results that go over 500 iterations it was finally decided to set the upper limit at 5000.

Pruning a trained network architecture

The main task in the pruning process is the analysis of the space spanned by the vectors of the outputs of the hidden units registered at the end of the training process. Various additional operations have to be executed in order to achieve a reasonable, reduced network architecture. Algorithm 5 contains an overview of the main steps that constitute the complete task.

The analysis of the space spanned by the outputs of the hidden units is performed through the computation of the eigenvalues and eigenvectors of the underlying covariance matrix; these items are calculated using the same routines as the ones employed in the previous section for reducing the input space to 3 dimensions. The number of redundant units is reflected in the percentage of eigenvalues which are close to zero. Deciding what is meant by being redundant or rather, close to zero, can be difficult when inspecting a set of numbers, most having a value under 0.001. In principal component analysis, the issue of deciding the number of eigenvalues which are sufficiently close to zero, to render the associated dimensions non-informative, is in itself an important topic of research [Jolliffe 86]. In **GROPING** the problem has been dealt with in the following manner:

(**1**) count the number of zeroes to the right of the decimal point of the smallest eigenvalue, set NumZero to this value.

(**2**) go through the list of ordered eigenvalues counting, for each of them, how many zeros there are to the right of the decimal point. If it is equal to NumZero proceed to the next eigenvalue in the list. If the number of zeros for this case is less than NumZero, consider that the number of eigenvalues which have as many zeroes to the right of the decimal point as the smallest of them, correspond to the number of redundant hidden units that

Algorithm 5 NetworkPruning(OutputHidden, FinalWeights, MaxArchW, H, NumIn, NumOut)

Require: $OutputHidden$ {vectors of outputs of hidden units}
Require: $FinalWeights$ {weight vectors of input-hidden layer registered at end of network training}
Require: $MaxArchW$ {weight vectors of maximal architecture}
Require: H, $NumIn$, $NumOut$ {number of hidden units, input dimension, output dimension}
Outputs: ReducedNetworkArchitecture;
$C \leftarrow covariance(OutputHidden)$;
$[RD] \leftarrow eigen(C)$; {R eigenvectors; D eigenvalues of covariance matrix of hidden units outputs}
for $i = 1$ to H **do**
$T(i,1) \leftarrow D(i,i)$;
end for{put eigenvalues into a one-dimension vector}
$sT \leftarrow sort(T)$; {sort the eigenvalues in ascending order}
$NumRedundant \leftarrow SelectRedundant(sT)$;
select number of units to prune checking on eigenvalues stored in sT
for $i = 1$ to H **do**
for $j = 1$ to $(NumIn + 1)$ **do**
$Ew(i,j) \leftarrow FinalWeights((i-1) * (NumIn + 1) + j, 1)$;
end for
end for
$INFOw \leftarrow \emptyset$; {Temporary variable to store information on weight vectors}
for $i = 1$ to H **do**
$INFOw(i,1) \leftarrow \sqrt{(Ew(i,:) * Ew(i,:)')}$; {compute length of final weights}
$INFOw(i,2) \leftarrow i$;
end for
$TmpW \leftarrow InfoW$; $InfoW \leftarrow sort(TmpW)$; {sort the lengths in increasing order to select largest weight vectors}
$num \leftarrow (H - NumRedundant)$;
for $i = 1$ to num **do**
$NewPw(i,:) \leftarrow MaxArchW(InfoW(i + num, 2), :)$;
end for
{select the weight vectors of initial maximal architecture they correspond to the heaviest final weight vectors};
$CreateFile(NewPw, num, NumOut)$; {create file of weight vectors for reduced architecture}

can be extracted from the network architecture. In this context, redundancy is defined in terms of the difference in the relative sizes of the eigenvalues, it is this information the one that indicates which are the important dimensions.

The philosophy underlying the `SelectRedundant()` routine was defined to reduce the costs of computing the final network architecture and, at the same time minimise the possibility of taking away too many hidden units in one step. Once NumRedundant has been computed it is necessary to select the weight vectors of the input-hidden layer of the new architecture. To select the most convenient set of weight vectors, the ones calculated at the end of the training process are sorted in increasing order according to their length. The information of the largest weight vectors is subsequently employed to extract the ones that will be used, to initialise the input-hidden layer of weights in the new reduced network architecture. This new set of weights is extracted from the complete set of vectors that were used to initialise the maximal network.

The manner of proceeding in the generation of the reduced set of weight vectors, was defined with the aim of taking advantage of the additional information produced during a network training process, implicit in the weight vectors of the resulting architecture. Those with a greater length, are associated with hidden units that have been more active in the overall training process, therefore the original weight vector from which each has been derived is probably relevant in the correct partitioning of the input space. There is no formal demonstration that such is the case but, performing a pruning process without a careful selection of the weight vectors usually produced lengthier training-pruning processes.

The weight vectors of the hidden-output layer in the reduced architecture are randomly generated numbers in the $[-1, 1]$ interval.

5.2.3 Final network evaluation

During the execution of the preceding operation, it is possible to generate a file with information concerning the evolution of the error measure during the various training processes that have to be executed. There are two sets of such measures for each of the intermediate architectures that had to be tuned, one corresponds to the error the network makes when presented with the training set and, the other is the error computed when the input provided corresponds to the elements of the validation set.

In order to produce a final network architecture that might be considered a good generaliser, the complete set of errors computed for each of the networks trained during the evolving/pruning process is analysed and, the architecture that produced the smallest error when processing the problem's validation set is selected as the final version that will be included in the application

of interest.

The choice is made to produce a good generaliser. This is certainly the case considering that it is based on the error each network that has been trained makes when processing, what can be considered to be novel data as the information subsumed in the validation set has not been relevant in any way during the training phase of the network design process.

To check the reliability of this procedure, the error on the test set was also measured for all the intermediate architectures and as, in both cases, the one producing the smallest error was the same, when building **GROPING**, the final network evaluation is restricted to using the error generated on the validation set.

5.3 Managing the weaknesses of the Multi-layer perceptron model

The causes which make the task of creating a multi-layer perceptron network hard can be summarised with the following statement: the problem of building an optimal network using the multi-layer perceptron for a particular application is **NP-complete**, [Hertz *et al.* 91]. Many of the inconveniences different authors have found with the model, and subsequently tried to solve in various ways were discussed in chapter 2, therefore the material included in this section simply describes the manner in which most of them have been dealt with in **GROPING**.

The main aim in this research has been to provide a learning algorithm that will improve the design process of a network built using the components provided with the multi-layer perceptron model. This has been achieved by reducing the amount of uncertainty related to the selection of the values the different parameters of the model should take when applied to a particular task. All the mechanisms defined to reduce this uncertainty were integrated in the algorithm described in the foregoing section and, its effectiveness is shown in the results depicted in chapter 6.

The different improvements to the network design process are summarised in the following list:

(**1**) the complexity measure which provides a reliable source of information for deciding on the value of using the multi-layer perceptron for knowledge representation. It is the case that its computation can be very costly for high dimension spaces but given the kind of information obtained for some cases it might be reasonable to proceed with its calculation.

(**2**) the initial maximal architecture which given the importance the hidden units have in the

overall process constitutes a significant improvement. Also the possibility of assigning a specific and appropriate set of values to certain of the weight vectors contained in that initial network.

(**3**) the **maximum** number of iterations that have to be performed by the training process. This **maximum** is a reasonable amount, as the results presented in table 5.2 show, given that it was generated from the execution of a concise statistical method. Knowing there is an adequate upper limit on the number of iterations over which a learning process should extend, greatly simplifies the task of training a multi-layer perceptron network.

(**4**) regarding the training component of the network design method, the one included is **rprop**, therefore there is no need to provide in advance values for the learning rate (η) or the momentum term (μ) and, it is possible to affirm that the execution of the learning algorithm is tailored to the application at hand which guarantees faster training.

(**5**) the mechanism used to control the over-fitting phenomenon, a common problem in generative-pruning algorithms [Reed 93], given that they start operating on an initial maximal architecture which means that the initial solution provided by the design process is over-trained. As it was possible with the geometric analysis to provide a maximal network that solves the task of interest it was necessary to design a proper method for reducing this over-trained structure. The manner in which the redundant units are eliminated, together with the fact that the final network architecture finally selected, corresponds to the one showing the smallest error when presented with the validation set, are proper mechanisms to deal with the main causes of the over-fitting phenomenon, because: (a) the improvement is in effect produced by reducing the network through the elimination of units from the hidden layer and, (b) the selection of the final architecture is performed using criteria based on data completely novel to the network, insofar as the training process did not use the data in any way to update the weight values of the different connections.

(**6**) the fact that the algorithm's processing is centered around a network that effectively solves the problem of interest is an overall advantage because it is always easier to improve on an existing large solution than to attempt on building an appropriate one from a minimal available architecture which is concluded to be less than minimal given its inability to correctly process the elements of the training set.

The overall gain achieved with the improvements sketched is the creation of a tool with which it is possible to: (a) reliably decide on whether or not to employ the multi-layer perceptron;

(b) in the affirmative case, investing an appropriate amount of computational effort create a network which is a reasonable generaliser.

5.4 GROPING: its computational costs

To determine the computational costs involved in the process of building multi-layer perceptron networks using the method defined, given its complexity, it is convenient to proceed by evaluating the costs incurred in executing each of its main steps, outlined in figure 5.1, given that **GROPING** will inherit the costs of the most expensive of them. This is applicable because the analysis realised corresponds to the worst case complexity as specified by the order-of-magnitude notation. Thus, the algorithm could presumably run for much less on certain inputs, perhaps on most inputs but, it is certain that it will never run for more than say, k times the expression governing the order associated to the algorithm being described [Harel 92].

Of the four main tasks mentioned in the process chart shown in figure 5.1 it is clear, that for an input of size N, the computational burden is shared between the second and third of them which correspond, respectively, to the construction of the maximal network architecture and, the training/pruning task. The other two are straightforward input-output operations linear, $O(N)$, in the size of the input sets and, as will be made clear in the ensuing discussion, do not come to bear in the computational costs incurred in executing **GROPING** because other tasks are more computationally intensive.

The hierarchical diagrams contained in figures 5.5 and 5.6, show respectively, the most important sub-processes that comprise the tasks of building the maximal network architecture and, pruning/evolving the network. The analysis of the computational costs incurred when executing either of them, described in the following subsections, is organised according to the contents of those diagrams.

5.4.1 Costs involved in building the maximal architecture

The input to this task is the same type of training set provided as input to a conventional supervised learning algorithm to train a multi-layer perceptron network. This means, that it has the form described in equations 5.1 and 5.2:

$$T_P = \{(x^1, o^1), (x^2, o^2), \ldots, (x^i, o^i), \ldots, (x^P, o^P))\}$$

where each pair (x^i, o^i) is comprised of an input pattern $x^i \in R^N$ and its expected output o^i. To maintain a consistent notation throughout, when referring to a linear algorithm, the term

will appear as $O(P)$ instead of the conventional $O(N)$ because P is the size of the training set and, N is already being used to denote the dimension of the input vectors comprising the training set; o^i corresponds to a C-dimension binary pattern, assuming that the classification task, of which T_P is a sample, involves C classes.

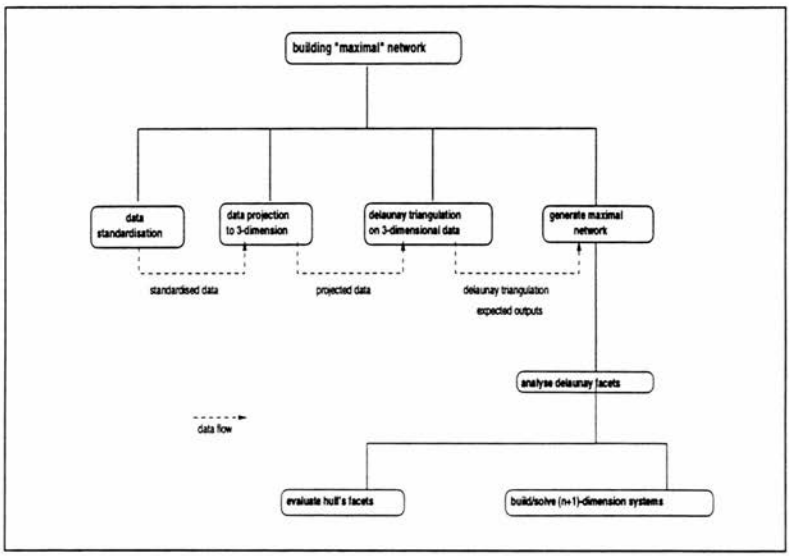


Figure 5.5: **hierarchical diagram: building maximal architecture**

Figure 5.5 shows the various subprocesses that comprise the task of building the maximal network. Given that the analysis aims at resolving the costs of the worst-case, in the context of this particular task, it can be calculated by selecting the highest cost incurred by any of the sibling subprocesses depicted in its process diagram. These costs correspond to:

(1) Data standardisation: this subprocess only involves matrix manipulations. The computation of the mean and covariance of the dataset $\mathbf{X} = \{x^1, x^2, \ldots, x^P\}$ according to the expressions specified in equations 5.4, 5.5; the subsequent calculation of the eigenvectors, eigenvalues of the covariance matrix and, the transformation of all the input patterns. The costs involved in each of the operations runs as follows:

Calculate mean($\mathbf{X}$), covariance($\mathbf{X}$): $O(P^2)$,

Compute eigenvectors, eigenvalues of N-dimension positive, symmetric matrix: $O(N^3)$

Data transformation: subtracting mean; rotating, scaling using eigenvectors: $O(P\,N^2)$

Looking at the costs associated with each of the operations, it is clear that this subprocess

is $O(P^2)$.

(2) Data projection: this is also a matrix manipulation operation. It involves the transformation, projection, of all the input patterns in $\mathbf{X}$ to the 3-dimension space spanned by the three strongest eigenvectors of the data's covariance matrix. It is $O(P\,N^2)$.

(3) Delaunay triangulation: this operation consists of a call to the convex hull implementation described in [Barber *et al.* 95] which computes the Delaunay triangulation of a set of 3-dimension points in $O(P\,\log(P))$. A detailed discussion on the computational costs of this implementation can be found in [Barber *et al.* 95].

(4) Network generation: the main structure that is input to this subprocess is the set of facets of the Delaunay triangulation computed in the previous step and, as it consists of a main loop that checks, whether each of the facets contained therein satisfy certain conditions, the costs involved in its execution are dominated by its size. For each facet F, in the worst case, this algorithm builds and solves a linear system of $N + 1$ equations using a method of QR decomposition. The implementation used for computing the Delaunay triangulation in the previous step, consists of a main loop over the set of facets it builds therefore, given that it is $O(P\,\log(P))$, it is possible to assume, in the worst case, that the size of the set of facets is approximately the same as the computational costs incurred in the execution of the geometric operation. Thus, the generation of the initial network has similar computational costs because: (a) the execution of the operation for solving the $(N+1)$ linear systems that are created, is just $O((N+1)^3)$ and, as the dimension of the input patterns is generally much smaller than the size of the complete set it can be taken as a constant term; (b) the main loop in this subprocess is over the set of facets created by the Delaunay triangulation. Therefore a,b guarantee that this subprocess is $O(P\,\log(P))$.

The previous analysis describes the first and third subprocesses as the ones involving the highest amount of computation in the global task that is being considered at this point. Of these two, the first one being $O(P^2)$ is the one whose complexity governs that of the whole operation. Therefore, it is possible to conclude that the task of building the maximal network architecture is $O(P^2)$.

5.4.2 Costs involved in pruning/evolving the network

This task, as the flow chart contained in figure 5.3 shows, is an iterative process over the number of intermediate network architectures that have to be built throughout. Assuming H units in the hidden layer of the initial maximal network, the worst case in this algorithm

would be if the architecture were reduced on a unitary basis, *i.e.* each time the number of redundant units is calculated for a particular network instance, only one unit is pruned and, the resulting architecture at the end of the reduction procedure has no hidden layer. Thus, the overall costs of executing one instance of the iterative process multiplied by H will correspond to the complexity of the pruning/evolving task.

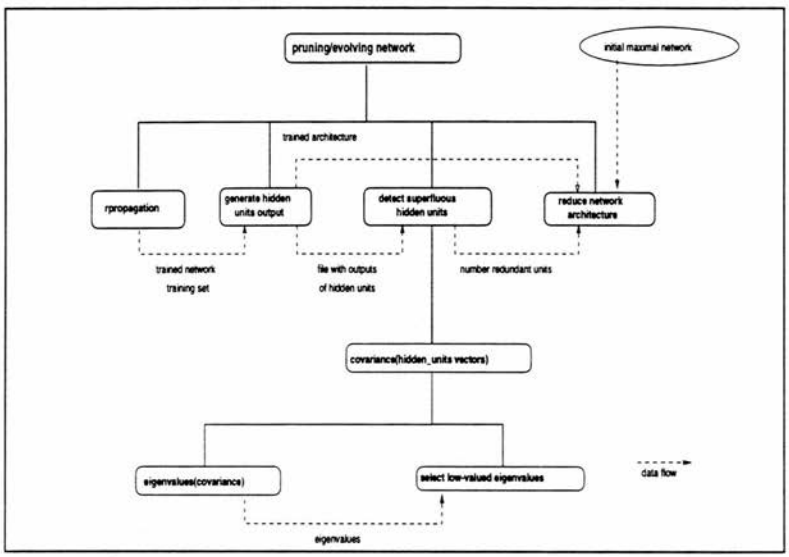


Figure 5.6: **hierarchical diagram: network pruning/evolving**

The hierarchy of subprocesses in figure 5.6 which, correspond to the ones executed to complete one instance of the iterative loop in the pruning/evolving task, can be used as a framework to estimate the overall costs it involves. The complexity of each of them is estimated in the following way:

(**1**) Rpropagation (**rprop**): as the main aim of this subprocess is to achieve complete recognition of the training set, with the architecture it is updating, it seems natural to define its computational complexity in terms of the size of this structure. Assuming that W represents the size of the architecture, measured by counting the number of connections and, that $NumIter$ corresponds to the **maximum** number of iterations **rprop** may perform updating the weight values of those connections, its complexity can be explained by considering the main steps that comprise it which are depicted in algorithm 6. Executing the main set of steps is O(W). As they are contained in a **while** loop that runs over the

total training set, it is possible to conclude that **rprop** is $O(P \times W)$. In the worst-case, NumIter would be held at a constant value of 5000 therefore, it only influences the constant term implicit in the **O-notation** and does not affect the overall complexity of this step.

Algorithm 6 Rpropagation

$i \leftarrow 0$;
while $(i \leq NumIter \;\&\&\; \neg(train(W)))$ **do**
  if $\neg(train(W))$ **then**
    must update network
    compute $\partial(ErrorMeasure)$;
    compute ΔW_{ij}; $W_{ij}^{new} \leftarrow W_{ij} + \Delta W_{ij}$; {both steps are O(2*W)}
  end if
  $Update(Error)$; $i++$;
end while{loop over number of iterations}

(2) Build output of hidden units: this is achieved by presenting the training set to the network and, storing the output computed by each of the units in the hidden layer. It is an $O(P \times W)$ operation.

(3) Calculate redundant units: the input to this operation is a $P \times H$ structure. As the process consists in performing a principal component analysis on P elements of an H-dimension space, *i.e.* the hu_vectors, the costs involved are:

Calculate covariance(hu_vectors): $O(P^2 \times H)$,

Compute eigenvalues of H-dimension positive, symmetric matrix: $O(H^3)$

Considering the worst-case scenario, the number of hidden units and the size of the training set are approximately the same therefore, the complexity of this operation is $O(P^3)$.

(4) Reduce network architecture: two network instances are used at this stage: $MaxW$ the weight vectors of the initial maximal architecture, W_{end} the weight vectors of the network generated by **rprop**. The complexity of this task is dominated by the sorting operation of the weight vectors comprising W_{end}, based on the size of each of them, *i.e.* the vector's length. H weight vectors must be sorted, therefore the complexity of this task is $O(H \log(H))$. The second part of this operation which consists in selecting the $(H - NumRedundant_{units})$ weight vectors from $MaxW$ which have the largest lengths in W_{end}, is a trivial operation in computational terms, given the information available.

There are two important complexity expressions in the steps of the pruning task, $O(P^3)$, $O(P \times W)$. Given that, the network used in the worst case has only one hidden layer, taking N as the number of inputs and, C as the number of outputs, the term W can be rewritten and,

subsequently replaced in the $O(P \times W)$ expression through the following set of substitutions:

$$W = (N+1) \times H + (H+1) \times C$$
$$O(P \times W) = O(P \times (H \times (N+C+1)+C))$$
$$O(P \times (H \times (N+C+1)+C)) \simeq O(P \times H \times N)$$

considering small-valued C even in the worst case because, it is associated with the number of classes of a classification task. Thus the complexity term $O(P \times W)$ has been reduced to $O(P \times H \times N)$ which eases the comparison against the $O(P^3)$ complexity. In a worst-case scenario it is also viable to assume, $H >> N$ and, $H \simeq P$, therefore the complexity of the pruning task has an $O(P^3)$ complexity.

5.4.3 Summing up GROPING's complexity

Given that the final operation of the complete design method involves selecting the best generaliser, its complexity can be measured assuming that, H network architectures were built. Given that, a **minimum** error was calculated whilst training each of them during the pruning task, the selection procedure has a $O(H)$ complexity because, it simply involves finding the smallest of these errors. The architecture associated with that **smallest** error will be the best generaliser.

The analysis throughout this section has produced three complexity measures, one for each of **GROPING**'s main tasks:

Building maximal architecture: $O(P \log(P))$
Pruning/evolving final network: $O(P^3)$
Selecting best generaliser: $O(H)$

and, under the worst-case assumptions, given that the function $f(x) = x^3$ grows at a much faster rate than $g(x) = x \log(x)$, it is reasonable to conclude that **GROPING** is an $O(P^3)$ algorithm.

5.5 Conclusions

The material presented in this chapter reports a pruning generative learning algorithm that provides a reliable alternative for designing multi-layer perceptron networks. Its main advantage rests in the coherent manner the different parameters involved in the design process are manipulated. For the most part they need not be specified by the application developer because they are instantiated and appropriately updated during execution. The manner in which

the instantiation proceeds is absolutely dependent on the attributes shown by the underlying problem domain. This information is extracted through the application of two types of methods: a) Generation of a geometric construct, *i.e.* the Delaunay triangulation of the training set which provides information concerning the proximity of the points included therein; b) principal component analysis, which calculates its most important dimensions.

The reasons for the effectiveness of the method defined are provided by examining the manner in which the multi-layer perceptron model functions. Its operating mode is of a geometric nature given that it processes the inputs provided to it by assigning them to regions of the problem domain; regions which result from the hidden units comprising the network architecture. Therefore the best way to decide on how many of these units are needed and, on the form the weight vectors associated to each of them should have, is by analysing the geometric nature of the domain, and **GROPING** is based on one such method.

Most of the high computational costs which might be incurred when trying to generate the type of information that **GROPING** employs have been dealt with reducing the complexity of its application. Nevertheless being a $O(P^3)$ algorithm means it is particularly sensitive to the size of the training set but, as it is a worst-case result, implying a polynomial-time algorithm it is possible to conclude that **GROPING** is a reasonable alternative and, that much can be gained by its application.

Chapter 6

Benchmarking and experimental results

6.1 Introduction

Generating the optimal network architecture for a particular application is a NP-complete problem. For this reason, when presenting a new learning algorithm, the best way of showing its effectiveness, is by comparing its performance against that of other existing learning algorithms which have been defined to operate on the same type of network model. To this end it is necessary to clearly state the objects involved and the set of assumptions underlying the solution proposed because there are many variants in the context of generative supervised learning algorithms which have an important influence on the interpretation of the results associated with a particular training process.

For example, the characteristics of the input and output domains associated with the mapping that wants to be reflected with a network architecture or, the type of measure the learning algorithm uses during a training process to decide on whether the network it is manipulating has achieved a reasonable degree of recognition. In the first case it is fundamental to clearly describe the domains that are supported by the type of network model and learning algorithm defined to ensure a level of success at the start of the development process of a network application and, also to provide an initial mechanism for deciding on the appropriateness of using the network model for the application of interest. In relation to the measure used by the algorithm during training, given that different types might be used and also that it might be computed in relation to different datasets, *eg.* the square error of the training set or the cross-entropy [Bishop 95] of the test set, it is necessary to depict the type of measure computed, what type of data is provided as input and how it affects the decision process underlying the whole training task.

The assumptions underlying **GROPING** have been described in Chapter 5 as it is comprised of a detailed discussion of the whole algorithm. This means it is not necessary to elaborate on them in this chapter but, some of them will be reviewed when discussing the difference in performance shown by a network design process executed using **GROPING**, when compared against the performance registered at the time of applying other learning algorithms to the same problem. To ensure an unbiased and reliable comparison, it is necessary to describe a framework that will ensure that the set of conditions influencing a network training process are properly controlled otherwise the realisation of a proper evaluation becomes difficult. To this end, in the area of neural networks different researchers have developed benchmarks which correspond to a series of problems that comply to a particular format and organisation together with a set of error measures. These, in conjunction with the complexity of the network architecture generated by a learning algorithm, permit the evaluation as to which of them is the most adequate.

The present chapter is comprised of the following parts: Section 6.2 includes a description of the different existing benchmarks considered, together with an enumeration of their advantages and/or disadvantages and, a set of arguments in support of the framework pursued in this study. Section 6.3 provides an outline of the problems finally selected for evaluating **GROPING**. Section 6.4 contains a description and discussion of the results obtained together with a comparison of its performance against that of other learning algorithms applied to the same set of problems. Section 6.5 summarises the main points of the chapter.

6.2 Benchmarks

In [Prechelt 94], a quantitative study of experimental evaluations of neural network learning algorithms is described, showing an interesting and not very satisfactory state of affairs in relation to the percentage of publications produced in the research area which include a proper, detailed and varied experimental evaluation of the learning algorithm described. In the conclusions it is observed that, in the material published in some of the most important journals in neural networks during the period 1993-1994, 29% of them did not include one learning example, only 8% of the articles analysed presented results for more than one problem and, one third of the complete set of publications considered did not present any quantitative comparison with a previously known algorithm.

This is probably the main reason for the efforts invested in the last couple of years aimed at developing a suitable benchmark environment because, it is clear that this is the most effective way of performing a proper evaluation of any learning algorithm, given that there

are no formal tools available to compare one against another. Due to the possibilities of worldwide communication provided by present technology, *eg.* the **W**orld **W**ide **W**eb, there is an enormous amount of data associated with different types of problems that can be used for designing the experiments to perform the evaluation task. The real difficulty is reflected in the set of conditions under which these experiments are performed because it is precisely, through their proper manipulation that many different results might be achieved, and this greatly restricts the possibility of comparing different existing alternatives.

The most important source of information on the **WWW** concerned with benchmarks for learning algorithms which might be applied to neural networks is the site:

`http://wwwipd.ira.uka.de/~prechelt/NIPS_bench.html`

which, at least at this time, seems to be the only one of its type available. From it, there is access to three types of sources that might be useful in the endevour of executing a benchmark:

(**1**) The main neural network benchmarking data collections which correspond to repositories of data specifically tailored for evaluating learning theories not restricted to the network model. There are five main sources of this type:

- (**a**) Carnegie-Mellon university benchmark.
- (**b**) UCI machine learning databases archive.
- (**c**) **Proben1** (Germany / USA).
- (**d**) Statlog data.
- (**e**) ELENA (**E**nhanced **L**earning for **E**volutive **N**eural **A**rchitectures) repository.

(**2**) **DELVE** (**D**ata for **E**valuating **L**earning in **V**alid **E**xperiments): theoretically, it is an environment comprising a collection of datasets from many sources, which provides a set of conditions that will ease the process of using the data to assess the performance of methods for learning relationships from the data. It also includes a repository for the results of such assessments. The project is developed by G. Hinton and E. Rasmussen at the university of Toronto. It seems to provide a good set of guidelines and software facilities to ensure the proper execution of the evaluation of a learning algorithm. Its main problem is rooted in the fact that its success is very dependent on the use the research community in general makes of it, as its effectiveness can only be measured in terms of the number of datasets and learning methods that different researchers might provide and which might be accessed by other members of the community. At the time of developing this research, only the first release of the tool was available and, considering

the difficulties involved in manipulating an experimental tool in addition to the set of difficulties always associated with the final stages of a research project, it was ultimately decided not to make use of this environment, nevertheless as a continuation of the research described herein it seems to have great possibilities.

(3) This group includes all the data sources which have been created in other areas such as Statistics which also require repositories of information to evaluate their methods. In addition, it comprises collections of sets of different problems that have been created by different research groups all over the world. The main problem with this type of source is that generally the data requires quite a lot of pre-processing before it might be considered a proper dataset for the task of training a neural network and, registered results generated by other learning algorithms against which to compare the results of your alternative are nonexistent. Additionally, given the number of data sets available which have been adequately processed and prepared for use in the evaluation of a neural network learning algorithm, there was no immediate need to download any of the sets included in this group.

From the previous categorisation it is clear that the type of repositories, that could be considered relevant to this study are those that comprise the first group. It is comprised of 5 collections of datasets, not all disjoint and some providing a better framework for evaluating the performance of a particular learning algorithm and, comparing it against that of other algorithms of the same type. The basic reason for considering that any two collections are not completely disjoint is because both have some datasets in common, *ie.* they both include samples associated to the same problem. The specific cause that renders them particularly useful for the study described herein, is that many of them have been used by other researchers for evaluating their learning algorithms. This means they provide an appropriate context for analysing the performance achieved with a particular network design algorithm as there are other results against which to measure its performance level.

The datasets of interest all correspond to classification tasks which, according to many of the existing publications dealing with the benchmark problem,[Aviles-Cruz *et al.* 95], [Prechelt 94], [Rasmussen *et al.* 96], [Prechelt 96], can be categorised on the basis of the means employed to generate the associated samples. Even though each offers a particular typification of the datasets that are available for building a benchmark, they generally agree on the existence of three basic types which, according to [Prechelt 96] might be defined in the following way:

(1) **Artificial data:** corresponds to data that has been artificially generated , *ie.* synthetically, based on some simple logic or arithmetic formula, such is the case of the

encoder/decoder problem or the datasets of the parity problem. The advantage of this kind of data is that its behaviour is very well understood therefore it facilitates the process of labeling the learning algorithm under consideration as it being appropriate or not, for data complying with certain characteristics.

(2) **Realistic data:** this also corresponds to synthetic data, but data generated by a model with properties similar to those that can be found in real problems. At the present time there are only three procedures available for generating data that might be termed realistic, [Prechelt 94]: a.- a procedure that uses a complex and realistic mathematical model of a physical system such as robot kinematics; b.- the generation of data by chaotic mathematical problems such as the Mackey-Glass equation [M.C. Mackey 77]; and, c.- data produced by stochastic processes such as mixture of Gaussian random variables. Together with the artificial data, the realistic type offers the possibility of characterising the kind of problem for which a learning algorithm would be particularly appropriate because the sources of the data are clearly understood. Given that this type of data generators allows to build samples requiring a network architecture for which it makes sense to measure its generalisation capacity it is considered to be comprised of a more interesting type of datasets for measuring the performance of a learning algorithm.

(3) **Real Data:** is that data that represents actual observations of phenomena in the physical world. Such data tends to contain some amount of error and noise, also in many cases it is not completely known, *ie.* it is not well specified. The main disadvantage is the degree of uncertainty. If it is too high it renders the multi-layer perceptron network model an impractical tool for building a solution to the problem because, a considerable amount of reliable information is required, so that it might be possible to design an appropriate architecture. Nevertheless, in many cases it is properly specified and, given that for them the multi-layer perceptron model is a very appropriate alternative to build an application, they constitute the most important type of datasets at the time of defining a benchmark.

All the data collections mentioned in the benchmarking webpage, at least those considered relevant in this research, are comprised of the three types of data. The most applicable of them for evaluating a learning algorithm for multi-layer perceptron networks is the **Proben1** repository because, it is specifically designed for the analysis of this type of learning theory, therefore it includes the most adequate set of guidelines that, if correctly pursued, result in a proper evaluation. In addition, it provides a set of results obtained with another learning algorithm for multi-layer perceptron networks against which to compare the performance level of the **GROPING** learning algorithm. Details of the **Proben1** repository will be described through the rest of the chapter as the results produced in this research are based on the

guidelines described therein. The rest of this section contains an outline of each of the relevant collections.

6.2.1 Carnegie-Mellon University (CMU) repository

The neural network benchmark contained in this repository can be accessed through two sites:

a.- The main directory of the CMU Artificial Intelligence re pository

`http://www.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/html/intro.html`

b.- Its own http address:

`http://www-cgi.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/html/intro.html`

The main CMU directory provides information on many important topics of Artificial Intelligence apart from neural networks. Browsing through the contents of all the subdirectories it contains, shown in table 6.1, it is reasonable to conclude that it constitutes an ideal starting point in the search for material on Artificial Intelligence. The fact that it does not provide some general benchmark, a set of problems and, an analysis on the behaviour exhibited by the solutions to these problems built using different AI tools, each designed on the basis of a particular AI paradigm, *eg.* production rules and neural networks, could be considered a drawback but, being set on a dynamic media like the **WWW** it is hoped that such a study might be included in the short term.

The CMU neural network site is organised so that there are two files for each of the problems comprising it: a text file describing the problem and showing tables of results generated with different multi-layer perceptrons architectures and, the proper data file that in three of the examples correspond to programs developed in C, which can be run to generate instances of the associated problem. There is no homogeneity in relation to the tests performed on each of the datasets, *eg.* for each problem the weights are initialised with random values in an interval that varies from problem to problem for no particular reason, some include results obtained by training the network with cascade correlation while others use backpropagation. Even though this deficiency in uniformity across the handling of the different problems could render this benchmark useless, the fact that they all include the same performance measure, *ie.* the percentage of correctly classified patterns from the training and test sets and, that there are a couple of interesting real-world problems, resulted in supporting a decision in favour of including part of its results in this study. The datasets stored are:

(1) Nettalk, it is an updated and corrected version of the data used by Sejnowski and Rosenberg in their study of speech generation using a neural network
[Sejnowski & Rosenberg 87]. It requires a considerable amount of pre-processing before a

top level in CMU repository	
directory_name	**general description**
areas	AI software packages
dod	documentation relating to FTP repositories
lang	AI programming languages
util	compression/archiving software

areas directory	
directory_name	**general description**
Agents	Agent Architectures
alife	Artificial Life and Complex Adaptive Systems
anneal	Simulated Annealing
areas	AI Newsgroup Archives
blackbrd	Blackboard Architectures
bookcode	Machine readable parts of various AI textbooks.
ca	Cellular Automata
classics	Classical AI Programs
constrnt	Constraint Processing
dai	Distributed AI
discover	Machine Discovery and Data-Mining
doc	Documentation and other AI-related notes.
edu	Educational Tools
expert	Expert Systems and Production Systems
faq	AI: FAQ postings for comp.ai
fuzzy	Fuzzy Logic
games	Game Playing
genetic	Genetic Algorithms and Genetic Programming
icot	IFS: ICOT Free Software
kr	Knowledge Representation
learning	Machine Learning
misc	Miscellaneous Unclassified AI Packages
music	Music
neural	Neural Networks, Connectionist and Neural Systems
nlp	Natural Language Processing (NLU, NLG, Parsing,...)
planning	Planning and Plan Recognition
reasonng	Reasoning
robotics	Robotics
search	Search
speech	Speech Recognition and Synthesis
testbeds	Testbeds, Simulators, and Micro-Worlds for Planning, Agents, and Robotics
virtual	Virtual Reality
vision	Computer Vision

Table 6.1: **CMU Artificial Intelligence Repository**

network can be trained to deal with it. Although the accompanying document describes the dataset, *ie.* the type/size of the inputs and outputs, and includes a detailed discussion on the processing the data has to undergo so that, it might be in a form that can be provided as input to a multi-layer perceptron network, given that not many results were registered and that the pre-processing needs to be completely implemented because in the repository only the raw data is available, it was decided not to include it as part of the benchmark for evaluating **GROPING**.

(**2**) Parity, is an example of an artificial benchmark problem. The task is to train a network to produce the sum, mod 2, of N binary inputs, otherwise known as computing the odd parity function. It is the **XOR** problem extended to an N-dimensional domain. Having a binary input domain means that all its elements can be clearly specified therefore the *generalisation* capacity of the network is not an essential matter, even though many studies have been based on the analysis of the minimum requirements network architectures designed for this task must satisfy, [Tesauro & Janssens 88]. Given the reasons discussed in section 2.3 it did not seem useful to include this particular set in the benchmark for assessing **GROPING**.

(**3**) Protein, is the data set contributed to the benchmark collection by Terry Sejnowski who developed it, in collaboration with Ning Qian, to study the effectiveness of using a neural network to predict the secondary structure of certain globular protein, [Sejnowski & Qian 88]. As it has the same problem the Nettalk data has regarding the extensive preprocessing required before it might be possible to build an application based on the multi-layer perceptron model, it was not considered useful in this research.

(**4**) Sonar, corresponds to another dataset donated by Sejnowski, used in a study he realised in conjunction with Gorman, [1988b], to determine the effectiveness of developing neural network applications for the task of classifying sonar signals. This is one of the real-world problems included in the **GROPING** benchmark, therefore further details on its characteristics will be provided in the following section.

(**5**) Vowel, corresponds to the other real-world problem in the CMU neural benchmark that was incorporated into the **GROPING** benchmark. It was donated to Carnegie-Mellon by Tony Robinson and, corresponds to data subtracted from a study on the possibility of achieving speaker independent recognition of the eleven steady state vowels of British English using a specified training set built using a number of measurements which are described in [Robinson 89]. Further details are provided in the next section.

(**6**) Two-spirals, **XOR**, both two very well-known artificial problems which have already being discussed at length in previous chapters.

This particular repository although not very extensive, does include a reasonable variety of problems and, as the performance measures registered for all cases is already computed by the network software developed, a positive use can be made of some of its contents. Finally it is worth noting that as an extension to the repository it includes a link to the **Proben1** benchmark, the one providing the framework on which the results generated in this study are based.

6.2.2 ELENA Project

This repository can be accessed at:

`http://www.dice.ucl.ac.be/neural-nets/ELENA/ELENA.html`

The datasets contained in it are the result of a project developed with the joint effort of different members of the European Community. Its principal aim was the evaluation of various neural network models designed to perform classification tasks. The choice of the databases was guided by various parameters, such as availability of published results concerning conventional classification algorithms, size of the database, number of attributes, number of classes, overlapping between classes and non-linearities of the borders.

6.2.3 Proben1 - Set of neural network benchmark problems and rules

The website of this benchmark software and data source is:

`http://wwwipd.ira.uka.de/~prechelt/NIPS_bench.html`

It consists mainly of a collection of problems designed to build neural network applications for two types of tasks: classification and function approximation. It contains 15 datasets from 12 different domains. As **GROPING**'s applicability is restricted to the development of classification networks only the datasets associated with this type of task were subtracted from the site and used to generate the results described in a subsequent section.

Together with the datasets, **Proben1** includes a set of rules regarding the manner a benchmark should be documented and conducted. These guidelines touch on the following objects associated to any benchmark procedure:

(**1**) Type of training, test set that should be used.

(**2**) Input and output representations of the datasets.

(**3**) Training algorithm.

(**4**) Error measures.

(**5**) Procedure for building the network architectures.

(**6**) Method for registering training and test results.

As it is considered a very complete proposal of guidelines, the analysis performed in this research is partly based on it. Details on the exact structure of the elements mentioned, will be provided in the section that includes a detailed description of the benchmark procedure followed.

6.2.4 Statlog data

The site of the repository is:

`ftp://ftp.ncc.up.pt/pub/Statlog/.`

It is the outcome of the joint effort of various researchers, aimed at the development of proper devices for comparing the performance of different learning algorithms and statistical tools. It provides quite a large number of results which include a great variety of learning and statistical methods. A detailed description of the complete project which resulted in the Statlog repository can be found in [Michie *et al.* 94]. Even though the published results include more than twenty datasets, only the following are available for public use:

(a) Australian credit.

(b) Diabetes of pima-indians.

(c) DNA sequence.

(d) German credit.

(e) Heart disease.

(f) Letter recognition.

(g) Image segmentation.

(h) Shuttle control.

(i) Landsat satellite image.

(j) Vehicle recognition using silhouettes.

All the datasets can be accessed at ftp.ncc.up.pt (192.26.239.52). The repository's webpage has links to information on the original sources and structure of the datasets together with a summary of the results obtained in the Statlog project. A table of results has been generated for each of the datasets listing information on time, memory and, error rates. These values were obtained for the training and test sets used in the process of capturing the form underlying the relevant dataset with each of the classifiers that were analysed. They used two methods for computing the error rates: train-and-test, cross-validation. In the first method the whole dataset is divided in two, 70% of it comprises the training set and the rest is contained in the test set; the total absolute error is computed for the training set and subsequently on the test set and finally used for calculating the error rate listed in the table. Given that cross-validation

involves dividing the sample into a number of subsamples, at the end of its application there will be an error rate for each of them, therefore in the table the error rate listed corresponds to an average over the subsamples. Given the number and diversity of classification devices that were evaluated in this study, in relation to the multi-layer perceptron they only worked with networks with one layer of hidden units and sigmoid activation function. Basic backpropagation was the learning algorithm used to train the network in all cases. The analysis also includes cascade correlation to provide an example of a classification mechanism designed to improve the performance of the multi-layer perceptron model based on the generative learning algorithm paradigm. This analysis has the following drawbacks:

(**1**) Given the enormous amount of literature providing proof of the advantage of using an enhanced version of backpropagation instead of the basic algorithm, it seems reasonable to consider that the multi-layer perceptron model working in conjunction with quick-propagation or rpropagation would have produced better results than the ones provided and, as the aim was to measure the effectiveness of the multi-layer perceptron model as a classifier, it would have been more appropriate to evaluate it with a more effective learning algorithm.

(**2**) In [Michie *et al.* 94] the cascade correlation algorithm is presented as an example of a model developed for improving the drawbacks of the multi-layer perceptron and back-propagation, nevertheless when looking at the tables of results only one third of them list results for the cascade algorithm and there are no explanations of why this is the case. Given that in general, constructive learning algorithms have allowed a wider applicability of the multi-layer perceptron model to classification tasks, more efforts should have been invested on evaluating cascade correlation over all the datasets just as it was done on the other classifiers included in the study.

(**3**) There is no information on the specific network architecture used for each of the examples, therefore it was difficult to measure the resulting Statlog multi-layer perceptron network in terms of architecture efficiency against the one built with **GROPING** for the same dataset. The same problem arises when looking at the few results registered with the cascade correlation algorithm, *ie.* they do not provide information on the final architecture computed by the algorithm. The large diversity of classifiers being compared and the differences in the data structures used by each of them deemed it necessary to define a very general measure for comparing the classifier's efficiency in terms of amount of space employed. This could justify the memory measure listed in all the tables of results nevertheless, given that the network architecture is a very relevant piece of information from the multi-layer perceptron model perspective it should have been included as part

of the results so as to make the benchmark more widely applicable. The same argument applies to the other classifiers and the structures generated by each of them in each of the datasets.

These drawbacks explain why, at least in terms of benchmarks, the one associated to this repository did not rank as the most important at the time of deciding on the best way of evaluating the capacity of the design method described in this study but, as it does provide the opportunity of looking at performance against that of a completely different type of classifiers, some results were compiled using a few of the datasets provided in this site. These will be described in the following section.

6.2.5 UCI machine learning databases archive

This is a repository of databases, domain theories and data generators used by the machine learning community for the empirical analysis of machine learning algorithms. The site of the repository in the **WWW** is:

`http://www.ics.uci.edu/AI/ML/MLDBRepository.html`

At this moment it contains over one hundred entries and is in continuous expansion, thanks to the joint efforts of the different members of the scientific community. Many of the benchmarks previously described have used part of the datasets included in their evaluation from this repository. It is probably the largest repository of databases, domain theories and data generators in existence. Apart from the datasets, it contains links to the most important machine learning sites including programs and publications in the area.

Given that the main interest in this study was to find datasets already used to evaluate other learning theories, the ones contained in this repository that were deemed interesting, were those for which some form of learning result that could be compared against the performance measure computed with **GROPING** had been registered.

6.3 Benchmarking problems and rules

The contents of this section supply information that facilitates the interpretation of the set of results obtained. The same can be classified in two basic groups:

(1) The rules specified in **Proben1** that were applied in **GROPING**'s evaluation.

(2) Descriptions of the datasets for which a multi-layer perceptron network was built applying **GROPING**.

6.3.1 Benchmarking rules

Proben1 defines a set of rules to ensure that, published results on its benchmark problems can be effectively compared and reproduced by other researchers. Its importance in the context of this study, stems from the possibility it provides for comparing results between different learning algorithms given that, it has been defined to evaluate neural network learning in the realm of pattern classification.

It is necessary to note a difference in the measures that were registered during a training phase in **GROPING** when compared to those computed for the same problem in **Proben1**. Both use the same basic definition of the measures registered in each to reflect the behaviour of a training process. For a particular problem, **Proben1** initialises at random various networks of the same size and subsequently trains all these architectures, the final measures published, offering evidence of the effectiveness of the training process, correspond to averages over the results obtained from the random initialisations. The approach pursued when applying **GROPING** on the same problem, also involves the creation of various architectures, given that it is a generative learning algorithm but, the difference between all these networks is of a structural deterministic nature, therefore the concept of averages is not practical. For this reason, as the basic aim when designing **GROPING** was to build the best generaliser, the final output of any application is one network, *i.e.* the architecture showing the smallest error when processing the validation set and, the measures registered were obtained during the training process of that architecture. In order to produce a reliable evaluation, the ones computed correspond to those for which the result of comparing an absolute value against an average is considered valid.

The guidelines outlined in [Prechelt 94], that should be unambiguously specified at the moment of evaluating a learning algorithm, comply with the following characteristics for the analysis of the algorithm described in this study:

(1) The training, validation and test sets in each problem, are the same as those provided in the benchmark's website. However, there is an important difference between the structure of the input to a network instance designed to solve the problem in **GROPING** and, that of the input to the network associated with the same problem used in **Proben1**. It is caused by the fact that the first step in **GROPING** requires transforming the data to zero mean and unitary variance therefore, every network is designed on the basis of a standardised version of the same dataset that, in its original form, is the input to the training process in **Proben1**. The covariance and mean matrix in every problem, needed to realise the normalisation, were computed from the training set. Some authors might claim the comparison invalid due to the difference in the inputs but, it could be argued

that a learning algorithm operating on a multi-layer perceptron network working with a sigmoid activation function always behaves better when processing normalised data, therefore this type of pre-processing should always be part of a design method working on this kind of model. Also, there is a reliable set of steps that must be followed and, the initial data source is the same. Using the information subsumed in the training set in a more effective manner, is not a strong enough reason to invalidate a better result, labeling it as incomparable. Finally, to measure how sensitive the learning algorithm is, to the ordering and specific instances of the patterns associated to a classification task, three permutations of the same data are associated to each of the problems comprising **Proben1** and, the set of results published were obtained from building network architectures using each of them. The same approach was pursued in this benchmark and, the permutations used were subtracted from **Proben1** when applicable.

(2) All the input data is handled as real-valued otherwise it is very easy to run into problems at the moment of executing the geometric analysis. The output data has the same structure as the one provided in **Proben1**.

(3) The other important difference regarding the input structure, concerns the manner in which the nominal attributes were manipulated. In **Proben1** they are replaced by binary patterns, in **GROPING** they are substituted by a set of values in the interval $[0, 1]$. If a nominal attribute can take one of four possible values, then the interval is divided by four producing the set $\{0.25, 0.5, 0.75, 1.00\}$ and each of the valid outcomes for the nominal attribute is associated to one of those numbers. The usefulness of this type of data transformation when working with the multi-layer perceptron model was discussed in chapter 2.

(4) In **Proben1** missing data is explicitly reflected in the input by using an additional attribute that is 1, only if the attribute value is missing. Given the type of data transformation that has been defined in **GROPING**, the strategy for dealing with missing values consists in subtracting the indicator attribute from the original set and, instantiating the missing elements with the mean calculated from the problem's patterns for which that particular attribute contains a value.

(5) The training algorithm is **rprop**, the version used has the characteristics described in chapter 2.

(6) Two error measures are computed throughout: the mean absolute error and, the squared error percentage. The expressions used for each of them, assuming a classification problem

with C classes and, a training set T_p such that:

$$T_p = \{(x^1, o^1), (x^2, o^2), \ldots, (x^i, o^i), \ldots, (x^P, o^P)\} \tag{6.1}$$

$$x^i \in R^N; o^i \in \{0,1\}^C \tag{6.2}$$

have the following form:

The **mean absolute error**:

$$\mathbf{ME} = \frac{1}{C\,P}\left(\sum_{i=1}^{P}\sum_{j=1}^{C}|o_j^i - c_j^i|\right) \tag{6.3}$$

The **square error percentage**:

$$\mathbf{SEP} = 100\left(\frac{1}{(o^{max} - o^{min})^2\,P\,C}\right)\sum_{i=1}^{P}\sum_{j=1}^{C}(o_j^i - c_j^i)^2 \tag{6.4}$$

The difference between the **ME** and **SEP** rests in the dependence the first of them has on the range of the output values and, the fact that **SEP**, in equation 6.4, is a squared percentage. Aside from that, the terms required to compute both quantities are the same. The main reason for incorporating the new error measure into **GROPING** lies in the fact that the results published in [Prechelt 94] for the most part refer to **SEP**, which means that only through its calculation could the benchmark be adequately used. As the network's weight values are updated in accordance with the periodic approach, all the patterns of the training set are presented to the network before updating any weight. This means that it is possible to compute the error measures described at different stages of the training process, after completing an epoch.

(7) There are two important assumptions underlying **GROPING**'s theoretical framework which clearly delimit the search space that has to be browsed in search of a network solution for a particular problem, resulting in a much faster procedure:

(a) The validity of Cybenko's results, concerning the capacities of a one-hidden layer perceptron with sigmoid activation function having enough hidden units which, in [Cybenko 88], were shown to extend over the complete set of continuous functions.

(b) The possibility of reflecting the structure of a classification problem through an adequate mapping involving the relevant inputs and expected outputs.

Those assumptions imply that, in the process of searching for the appropriate network architecture to solve a particular problem, the space of possible network solutions can be restricted to one-hidden layer perceptrons, a space which is reduced even more, by the results obtained from the geometric analysis realised on the elements of the training set in the first phase of the design process.

Data availability					
dataset/repository	CMU	ELENA	Proben1	Statlog	UCI
cancer	N	N	Y	N	Y
card	N	N	Y	N	Y
diabetes	N	N	Y	Y	Y
gene	N	N	Y	N	N
glass	N	N	Y	N	Y
heart	N	N	Y	Y	Y
mushroom	N	N	Y	N	Y
satimage	N	N	N	Y	N
soybean	N	N	Y	N	Y
thyroid	N	N	Y	N	Y
vowel	Y	N	N	N	N

Table 6.2: **Principal datasets vs Repositories**

(8) **GROPING** is a generative-pruning algorithm. The results presented were registered at the end of the training process performed on the various networks built during the pruning phase in the search for the optimal network. For each of the problems included in this benchmark the following results are published in this written work:

(a) a list of all the architectures that had to be trained in the search for the optimal one.

(b) the mean error, **ME**, the squared error percentage **SEP**, registered at the end of the training process for each of the architectures. The two error measures are computed for the three relevant datasets: the training, validation and, test set.

(c) the number of epochs through which the training process had to iterate before each of the architectures achieved complete recognition of the training set.

(d) the results associated to the optimal network, *i.e.* that architecture with which the smallest error was achieved when presenting the validation set. This solution is compared against the average solutions published in **Proben1**.

(9) The completion of an execution of the **GROPING** algorithm occurs when, during the architecture reduction phase, it is processing a particular network that after having been trained for the **maximum** allowed number of iterations does not achieve complete recognition of the elements of the training set.

6.3.2 Benchmark datasets

Part of the work realised involved looking into different sites found in the **WWW** that contained collections of datasets and, included the results obtained by applying various learning

Problem	Problem attributes				Inputs **Proben1**				**GROPING**	Classes	
	b	c	n	tot	b	c	m	tot	cont inputs	b	size
cancer	0	9	0	9	0	9	0	9	9	2	699
card	4	6	5	15	40	6	5	51	15	2	690
diabetes	0	8	0	8	0	8	0	8	8	2	768
gene	0	0	60	60	120	0	0	120	120-60	3	3175
glass	0	9	0	9	0	9	0	9	9	6	214
heartc	1	6	6	13	18	6	11	35	13	2	303
mushroom	0	0	22	22	125	0	0	125	22	2	8124
satimage	0	36	0	36	-	-	-	-	36	6	6435
soybean	16	6	13	35	46	9	27	82	35	19	683
thyroid	15	6	0	21	9	6	6	21	21	3	7200
vowel	0	10	0	10	-	-	-	-	10	11	990

Table 6.3: **Attribute structure of datasets in its original form**

algorithms on them. To achieve a rigorous analysis on the scope of **GROPING**'s applicability it was decided that it was necessary to add other sets to this benchmark apart from the ones subtracted from **Proben1**. Each row in table 6.2 corresponds to one of the datasets for which a network was developed using **GROPING** and, each column is associated to one of the sites visited that contained the type of information required. The contents of an entry in the table, which represents a particular (dataset, site) pair, indicates whether the dataset is available (Y) or not (N) in the site.

Given that, in many cases it was much more effective to solve a problem applying the **GROPING** algorithm after the original data had undergone an adequate transformation, a series of steps were integrated into a pre-processing phase, all of which are described in chapter 5. Details of the outcome of this pre-processing phase when applied to each of the datasets included in the benchmark can be found in appendix F, together with other relevant details associated to every classification problem considered. A special effort was made to avoid duplicating information already available at the site from which the set was originally subtracted.

The following subsections contain a summary of the different examples used, including a description of the particular actions that had to be taken to achieve appropriate training processes when operating on any of them. The contents of table 6.3, which were partly subtracted from [Prechelt 94], summarise the structure of the data associated to each of the problems that comprise this benchmark, number and type of inputs and outputs together with the total number of patterns and classes comprising each of them. Given that, the type of pre-processing used in **Proben1** does not necessarily correspond to the one used in **GROPING** and, that it can result in an input structure different from that of the original version of the data, table 6.3 contains three descriptions of the input space of each of the problems: the one associated to the original data, columns 2-5, the number of attributes used in **Proben1**, for the cases in which it

applies, columns 6-9, the number of continuous attributes that constitute the network's input in **GROPING**, column 10. The other columns in the table contain information that applies equally to all three versions of the datasets, *i.e.* the structure of the outputs, the number of classes and, the size of the problem's sample.

Cancer

This sample is originally from the UCI machine learning dataset: Wisconsin breast cancer database. The version used was extracted from the **Proben1** site. There are 9 continuous inputs all scaled to the [0...1] range. In the original UCI set, there are 16 missing values for attribute 6 which were encoded as 0.3, therefore when the data is used in this benchmark, it is registered as not having missing values.

Given that all the inputs are continuous and that the missing values were dealt with in **Proben1**, in this case there is no nominal data to transform. The data is used as is in the following stage of the pre-processing phase which consists in subtracting the required training, validation and test sets and, performing the subsequent data normalisation. Details of the results obtained are contained in appendix F.

Card

This dataset is from the UCI machine learning database: credit-screening. It is interesting because it contains a reasonable amount of missing information which required special treatment. According to the set's description outlined in table 6.3, the original source corresponds to 690 patterns set in a space of 15 dimensions, 5 of which are nominal. To deal with the nominal and missing data, in **Proben1** the structure of the input was transformed to a 51-dimension space. This transformation is not considered to be very effective for the following reasons:

(1) The numerous additional attributes defined in the new space when compared against the size of the original version of the data. Many of these additional dimensions will be zero for most of the patterns because they are interpreted as tags which indicate, if the value of the neighbouring attribute corresponds to missing information or to a proper instantiation of the attribute. This means that, it is possible to produce a training set which has no missing data therefore, the relevant dimension will have zero value throughout and, such uninformative attributes do not ease the process of finding the proper weight values of a network during training.

(2) As the geometric analysis, that is part of **GROPING**'s operations, is performed on a projected version of the original training set, it did not seem very efficient to proceed by

expanding the input to a high-dimension space, that would immediately be reduced by a projection mapping required to generate the appropriate maximal network.

(3) Comparing the results achieved with the network built for the high-dimension space with those obtained with **GROPING** working on a continuous-valued version of the whole set, having replaced the missing information with the mean of the available data, provides the strongest reason in support of the approach underlying the pre-processing phase incorporated in **GROPING**, when compared against the results achieved by applying other strategies.

The data processed with **GROPING** has the same dimension as that of the original data contained in the UCI database. The main difference lies in the fact that all the attributes in the data for this benchmark are continuous-valued. Appendix F contains the details of the transformation and the main results produced during the training-pruning phase.

Diabetes

This dataset is also from the UCI machine learning database, although the version that was used in this benchmark was subtracted from the **Proben1** site. There are 768 patterns in the set.

The problem is set in an 8-dimension operating space and corresponds to a 2-class classification task. All the attributes are continuous-valued and, because they were subtracted from **Proben1** instead of from their original source, the input to the **GROPING** algorithm corresponds to attributes scaled to the [0..1] interval. The data did not require any special preprocessing previous to the normalisation and geometric analysis. Details of the normalisation and, the results obtained by the learning algorithm are contained in appendix F.

Gene

This dataset is from the UCI machine learning database: splice-junction gene sequences. Each vector of the original data represents a given position in the middle of a window of 60 DNA sequence elements (called nucleotides or base-pairs), The aim of the classification task is to predict into which, of the three following categories, does a gene sequence fall:

a.-intron -> exon boundary (ie) [These are sometimes called donors]

b.-exon -> intron boundary (ei) [These are sometimes called acceptors]

c.-neither (n)

To exploit the advantages furnished by the existing statistical and mathematical techniques, in the context of data analysis, it is necessary to define a clear policy for managing nominal data because generally it is not numerical so, it must be transformed through an adequate pre-processing technique before the execution of the relevant data analysis method. This issue was extensively discussed in Chapter 2, which includes a description of the policy pursued in **GROPING**. This dataset was selected as an example that shows the gains obtained in the context of the development of multi-layer perceptron networks, when applying the pre-processing technique devised. To this end, two networks have been defined, one uses as input the data encoded according to the guidelines defined in **Proben1**, the other was tailored to deal with an input that was encoded according to the **GROPING** policy. The results of both training processes are included in appendix F and clearly show that much better results are achieved by transforming the nominal data to a well-specified and small set of real values.

Another reason for having built both networks for this particular dataset is that it provides a good example for analysing the effectiveness of the **GROPING** algorithm on very high dimension spaces and, the encoding defined in **Proben1** to deal with the dataset results in a very high-dimension, *i.e.* 120-dimension space.

The output patterns are 3-dimension binary vectors, which can take one of three values according to the class to which the DNA-sequence belongs. There are 3175 patterns in the complete dataset but, as an inconsistency was found in the data, *i.e.* two input vectors had the same values in all the attributes but were associated to different outputs, the sample used in **GROPING** consists of 3173 vectors each associated with a 3-dimension binary output vector.

Glass

The data corresponds to a classification task involving a high number of classes, given that any input pattern has to be assigned to one of 6 possible classes. All the attributes of the 9-dimensional input space are continuous-valued so there was no need to perform any transformation before partitioning the available data into the corresponding training, validation and test sets.

The data employed in this benchmark was obtained from **Proben1** but, the original source is the UCI database. The most interesting characteristic of this problem concerns the high number of classes that the neural network has to correctly identify given that, for the most part, a diagnosis task only involves two classes.

The results obtained when executing the geometric analysis are based on the number of geometric constructs, *i.e.* edges, connecting points that do not belong to the same class. When

there are only two classes it is straightforward but, a classification task involving more than two categories, incorporates the difficulty of dealing with the additional information. It was necessary to decide on how relevant the issue was to the task of building the initial **maximal** architecture for the problem. It was concluded that, taking account of the information about the precise class to which a point belongs, instead of simply deciding on whether connected points were or were not, part of the same group, would not result in a better network because the geometric complexity of the required solution is independent of the particular class to which the points involved belonged.

Also the algorithm designed to perform the geometric analysis was greatly complicated if, in addition to the information concerning the existence of classes, it had to consider the type of each of the ones involved. This can be explained by viewing this information as the dimensions of the output space and, many examples have proved, [O'Rourke 94], [Barber *et al.* 95], how sensitive this type of algorithm is to the dimensionality of the operating space.

The information on the characteristics of this dataset and the results achieved during the generation of the network required for it are included in appendix F.

Heart

The problem's instances are associated with a heart disease diagnosis task. As the data was collected from four different locations there are different versions of it in the **Proben1** repository. The first difference between them is the original source which in **Proben1** resulted in two examples: one built by uniting the data subtracted from all the locations: Cleveland Clinic Foundation, Hungarian Institute of Cardiology, V.A. Medical Center Long Beach, and University Hospital Zurich. The other contains only the Cleveland data.

The output can be interpreted as a continuous domain or as a binary domain. In the first case the network required has to perform an approximation task. For the binary case, a classifier network solves the problem.

Table 6.3 shows that the original source is a 13-dimension set which is transformed to a 35-dimensional space in **Proben1**. To train a network for the same problem with **GROPING** the nominal attributes of the original source were transformed to continuous-valued ones following the steps of the pre-processing stage described in chapter 2. This means that the data finally used in this benchmark corresponds to a 13-dimensional continuous-valued classification problem of the Cleveland data.

The other versions of the same data were not deemed interesting for the following reasons:

(a) the multi-layer perceptron is not a good model for problems with a high percentage of missing information. As the Cleveland data is the most complete of all the sources it was decided that only this case could be of any value in the present study.

(b) the multi-layer perceptron was analysed in the context of a classification problem. This implies that the interpretation of the problem as an approximation task is of no concern in this research.

The data in **Proben1** was obtained from the UCI repository. The results of the pre-processing phase, the generation of the appropriate intermediate architectures are all included in appendix F. Although the final errors achieved with the **Proben1** network are better than those registered in **GROPING**, the fact that the network architecture is so much more complex gives grounds for considering the results achieved in both benchmarks as analogous, because when describing the best solution a simpler architecture is also an important output.

It could be argued that, given the number of attributes acting as indicators for missing information and, the complexity of the optimal network built in **Proben1**, it is probably the case that for this particular problem an alternative model will provide a better solution for the final application. The results registered when building the architecture required could be taken to be clear signs that the network model is not designed to deal with its particular characteristics and, alternative representation forms will yield better implementations.

Mushroom

This dataset is a 22-dimension sample of completely nominal data. Therefore, all its attributes had to be transformed to numerical-valued ones, before it was possible to start building a multi-layer perceptron network for it. The strategy applied is similar to the one described in the card problem but, adapted to the type of attributes that constitute the elements of this example.

The results of the pre-processing phase and, the generation of the appropriate intermediate architectures are all included in appendix F. Overall, they are particularly good which means this type of data is very well suited to the type of network model that is being analysed. Comparing the results obtained against those published in [Prechelt 94] for the same data, it can be concluded that, for this particular case, both types of manipulation of the nominal data are equally effective even though, the errors achieved with **GROPING** are smaller. Comparing the results in terms of the amount of resources that had to be used in each case, **GROPING**'s approach is much more effective given that, it produces a much smaller input as it is comprised of a lesser number of dimensions.

results published in Statlog		
classifier	error_train	error_test
KNN	0.089	0.094
LVQ	0.048	0.105
Dipol92	0.051	0.111
Radial	0.111	0.121
Alloc80	0.036	0.132
IndCart	0.023	0.138
Cart	0.079	0.138
BackProp	0.112	0.139
BayTree	0.020	0.147
NewId	0.067	0.150
Cn2	0.010	0.150
C4.5	0.040	0.150
Cal5	0.125	0.151
QuaDisc	0.106	0.155
Ac2	*	0.157
Smart	0.123	0.159
LogDisc	0.119	0.163
Cascade	0.112	0.163
Discrim	0.149	0.171
Kohonen	0.101	0.179
Castle	0.186	0.194
Bayes	0.308	0.287
Default	0.758	0.769

Table 6.4: **table of results for satimage**

Satimage

The data associated with this problem was subtracted from the Statlog benchmark. It consists of the multi-spectral values of pixels in 3x3 neighbourhoods in a satellite image, and the classification associated with the central pixel in each neighbourhood. The aim is to predict this classification, given the multi-spectral values. The output is encoded as numeric values in the original source but in **GROPING** the 6 possible classes were labeled with the binary patterns usually associated with a classification task that is to be processed with a multi-layer perceptron.

To compare the results obtained with this problem using **GROPING** which are detailed in appendix F, against those published in [Michie *et al.* 94] it was necessary to assume certain things about the published results because there was not enough detail on the exact form used to calculate them. Table 6.4 contains the error rate on the training and test sets defined in Statlog achieved by the different classifiers that were evaluated in that benchmark. The default rule in the table is performed by allocating each observation to the most common class, the symbol "*" indicates not applicable.

Assuming that the error rate corresponds to the error measure defined in chapter 2 as the most

commonly used in classifier theory, *i.e.* the sum of square errors, it is possible to compare the performance against the **mean error** registered in the table of results shown at the end of the section. Of them all, the best result was achieved with the KNN which is much worse than the error registered with **GROPING** on its best network architecture: $mean\_error = 0.0780$. Taking into account that what is being compared is a squared result against an absolute value the difference gains even more significance.

The result that is particularly important is the one achieved with Cascade correlation given that it is the same type of classifier as **GROPING**.

Soybean

This dataset corresponds to another example of a high-dimension output space because, the network must discriminate between 19 different classes which correspond to diseases of soybeans. The original data as table 6.3 shows, is a good mix of nominal, continuous and binary attributes which in the first source, *i.e.* the UCI database, corresponds to a 35-dimension space.

As there are missing values the data used in **Proben1** is extended to include those dimensions that operate as indicators of missing values and, resulted in an 82-dimension space.

The data used in this benchmark was subtracted from the **Proben1** site. Of the 683 patterns, many turned out to be duplicates of each other given that, when computing the distance that existed between every pair of them, 68 are zero. This is not a problem when building a neural network unless they happen to be associated with different expected outputs. The analysis for determining the existence of duplicate values was realised on the data transformed according to the stipulations of **Proben1** and also according to the encoding devised in this research.

As in previous cases, missing values were replaced by the mean of the available information and, the nominal data was transformed to its corresponding values in the interval [0..1] partitioned according to the size of the set associated to the nominal version of the attribute. The intermediate information generated during the network design process and, the results achieved are all in appendix F.

Thyroid

This dataset is from the UCI machine learning database: thyroid disease. The original data files were used to evaluate the performance of different learning algorithms defined to train networks based on the multi-layer perceptron model. The results of this analysis are described

in [Schiffmann *et al.* 93].

The ones obtained by building a network for this problem in this study were only compared against those registered in the **Proben1** benchmark given that, the training, test sets used and the error measures registered in [Schiffmann *et al.* 93] correspond to another type of approach. The important thing to note about this set is that although some of the attributes are simply indicators confirming the existence or, otherwise of the values of other attributes, in the information available it is not clear which correspond to proper data and which are simply indicators. This meant that, in this particular case it was not possible to treat missing data in the manner specified in **GROPING**'s pre-processing phase and, all the twenty one attributes are processed as data in its own right.

The results contained in appendix F show that this restriction did not affect the overall network building process but, it is worth noting that, if it were possible to deal with the missing results in the manner specified in **GROPING**'s general rules the chances of obtaining even better results are good.

Vowel

The data associated with this problem was subtracted from the CMU repository. The input patterns are 10-dimensional continuous vectors that represent examples of the eleven steady state vowels of spoken English. This is reflected in the sample using a binary vector of the form $(1,0,0,0,0,0,0,0,0,0,0)$ for each of the possible vowel classes. These vectors vary from class to class in the position the output 1 occupies.

Given that the input is continuous it did not require other pre-processing apart from the reduction to zero mean and unitary variance. Details of the elements used for this processing are provided in appendix F.

The final network built with **GROPING** to solve this problem could have been designed using one continuous output and assigning a specific value to each of the classes but, because the algorithm at this time only supports classification tasks in which the outputs must be binary patterns it had to be defined as a high-dimensional output space. Results of the network processing the alternative output representation would probably show a better behaviour as it will be operating on a low-dimension domain.

Pivot architectures for different datasets				
	Proben1		**GROPING**	
dataset	architecture	num_connections	architecture	num_connections
cancer1	9-4-2-2	100	9-15-2	165
card1	51-32-0-2	1832	15-27-2	488
diabetes1	8-32-0-2	370	8-94-2	1578
other_gene1	—	—	60-216-2	7751
gene1	120-4-2-3	1115	120-43-3	11659
glass1	9-16-8-6	572	9-32-6	518
heartc1	35-16-8-2	1112	13-26-2	442
horse1	58-16-8-3	1793	—	—
mushroom1	125-2	252	22-38-2	952
satimage	—	—	36-76-6	3274
soybean1	82-16-8-19	4153	35-28-19	1559
thyroid1	21-16-8-3	794	21-135-3	3278
vowel1	—	—	10-42-11	935

Table 6.5: Best network architectures in **Proben1** and **GROPING**

6.4 Results and discussion

At the moment of comparing the effectiveness of different learning algorithms that have been defined to generate networks based on the same type of model, it is necessary to specify on which parameters the decision criteria is based because, the outcome of the comparison is very dependent on the ones selected; *eg.* comparing algorithms A, B tuning the same network using the same training set, the error registered at the end of the training process might be smaller for algorithm A but, the error on the test set can indicate that the better algorithm is B, the difference lies in the parameter on which the decision is based. In the first case, the selection was made evaluating the mean error on the training set, in the second case the error analysed was the one computed on the test set.

The convenience of designing a multi-layer perceptron network using **GROPING**, is demonstrated through the values calculated for the different parameters listed in table 6.7. Each of the rows in that table, corresponds to the results achieved by the training process operating on the data available for a particular task, which is identified by the name registered in the first column. Those names follow the format $< name_i >$, where $< name >$ is the identifier of each of the classification tasks that constitute the benchmark and <i> indexes the permutation of the data that was used to build the particular set of parameter values included in the same row. Being a pruning-generative method means that different network architectures were trained for a particular task and, the values contained in table 6.7 correspond to the ones achieved with the most appropriate network, *i.e.* the architecture which registered the smallest error at the moment of processing the validation set. Some of the parameters listed in table 6.7 were also calculated in the training processes performed in the **Proben1** benchmark and, it is against

Proben1 averages and standard deviations of square error percentage							
problem	iterations	sep_train	std_dev_train	sep_validation	std_dev_validation	sep_test	std_dev_test
cancer1	152	2.87	0.27	1.96	0.25	1.60	0.41
cancer2	93	2.08	0.35	1.77	0.32	3.40	0.33
cancer3	66	1.73	0.19	2.86	0.11	2.57	0.24
diabetes1	196	14.74	2.03	16.36	2.14	17.30	1.91
diabetes2	119	13.12	1.35	17.10	0.91	18.20	1.08
diabetes3	307	13.34	1.11	17.98	0.62	16.68	0.67
gene1	46	6.45	0.42	10.27	0.31	10.72	0.31
gene2	321	7.56	1.81	11.80	1.19	11.39	1.28
gene3	435	6.88	1.76	11.18	1.06	12.14	0.95
glass1	67	7.68	0.79	9.48	0.24	9.75	0.41
glass2	29	8.43	0.53	10.44	0.48	10.27	0.40
glass3	66	7.56	0.98	9.23	0.25	10.91	0.48
heartc1	45	6.82	1.20	8.75	0.71	17.18	0.79
heartc2	29	10.41	1.76	17.02	1.12	6.47	2.86
heartc3	24	10.30	1.79	15.17	1.83	14.57	2.82
mushroom1	3000	0.014	0.00	0.014	0.00	0.011	0.00
soybean1	665	0.32	0.008	0.85	0.07	1.03	0.05
soybean2	792	0.42	0.06	0.67	0.06	0.90	0.08
soybean3	759	0.40	0.07	0.82	0.06	1.05	0.09
thyroid1	491	0.60	0.53	1.04	0.61	1.31	0.55
thyroid2	660	0.59	0.24	0.88	0.19	1.02	0.18
thyroid3	598	0.69	0.20	0.97	0.13	1.16	0.16

Table 6.6: **Results published in Proben1 for pivot network architecture**

those results that the convenience of applying **GROPING** is compared.

In the **Proben1** benchmark various results were obtained through the execution of a series of training processes operating on the same datasets as the ones used in the **GROPING** benchmark. Of all the results published in [Prechelt 94] the relevant ones for this discussion are listed in table 6.6. For each variant of each problem, 60 runs were 160 performed, each starting with a particular random initialisation of the same architecture, *i.e.* the pivot network. The results published correspond to the averages and standard deviations of the square error percentages calculated in each of those runs. The learning algorithm used for those runs is **rprop** therefore, the training parameters that had to be defined were:

- $\eta^+ = 1.2, \eta^- = 0.5,$
- $\Delta_0 \in 0.005 \ldots 0.02$
- $\Delta_{min} = 0, \Delta_{max} = 50,$
- initial weights, random from $[-0.01 \ldots 0.01]$
- training was terminated using GL_5 criteria [Prechelt 94].

Given that the training component in **GROPING** also uses **rprop**, the parameters associated with the algorithm have been explained in detail in chapter 2. The stopping criterion defined

in [Prechelt 94] is a measure that integrates various occurrences of the square percentage error. These were computed on the validation set at certain stages of the training process. For a particular architecture, Prechelt [1994] refers to it as the generalisation loss of the network. The training is stopped as it exceeds a certain threshold α, whose selection is also discussed in [Prechelt 94].

The best (pivot) network in each classification problem corresponds to the largest topology of all the candidate topologies trained for the task showing the smallest validation square percentage error at the end of the training process. The results obtained with **GROPING** for the same set of tasks are compared to those achieved with this pivot architecture because, in both benchmarks the respective network corresponds to the one showing the best results, *i.e.* the smallest error at the time of processing the same validation set. Table 6.5 shows the size and, structure of the most appropriate networks according to the results produced by each benchmark. Looking at them it is possible to conclude that **GROPING** produced more complex networks relative to the number of weights comprising each of them but, the ones built according to the criteria set in **Proben1** are more complex in terms of the final shape because, many of them use two layers of hidden units and the architectures are not restricted to simple fully-connected feedforward networks, *i.e.* cross-connections between units not in subsequent layers is permissible.

The set of values contained in table 6.7 apart from including the square percentage error calculated on the training, validation and test sets with the optimal architecture, contain the mean error because, it is a more common measure that make these results more accessible to a wider range of similar studies.

Looking at the errors registered on the different types of sets, for each of the classification tasks used for both benchmarks it is possible to note the following interesting differences:

(**1**) the error calculated using the training set is much smaller in any of the networks built with **GROPING** when compared against the same type of error in **Proben1**. It could be argued that this does not make the networks built with **GROPING** a better alternative for the relevant task because, it is very often the case that, a network which is a better recogniser of the elements of the training set, is a deficient performer when classifying data not used in the training process. Contrary to this belief, in this analysis, although there is not so much difference between the errors calculated using the elements of the validation and test sets in both benchmarks, as there is between the errors of the training set, the architectures built with **GROPING** are better generalisers.

(**2**) the number of iterations required to achieve complete recognition of the training set is,

Square error percentage and Mean error using GROPING							
problem	iterations	sep_train	sep_validation	sep_test	merror_train	merror_validation	merror_test
cancer1	101	0.0110	1.9090	1.8561	0.0025	0.0269	0.0284
cancer2	43	0.0038	1.8084	3.0743	0.0014	0.0221	0.0399
cancer3	45	0.0049	3.8262	4.1058	0.0012	0.0437	0.0467
card1	51	3.1121	10.4174	10.8650	0.0893	0.1715	0.1875
card2	101	0.2332	11.2323	18.0754	0.0189	0.1481	0.2199
card3	51	3.5124	9.6063	16.5102	0.1027	0.1722	0.2466
diabetes1	51	9.5291	15.4510	17.6195	0.2294	0.2829	0.3143
diabetes2	51	6.2648	17.5532	20.3791	0.1599	0.2821	0.3111
diabetes3	51	8.5511	17.9033	17.7496	0.1957	0.2975	0.2991
gene1	16	1.3780	6.2421	6.7345	0.0579	0.1191	0.1237
other_gene1	33	0.1832	4.4745	5.4952	0.0177	0.0780	0.0904
gene2	16	1.4602	6.5589	5.7677	0.0527	0.1169	0.1068
gene3	16	1.2182	6.5021	6.2701	0.0551	0.1219	0.1199
glass1	70	0.0157	11.1582	10.8181	0.0037	0.1282	0.1278
glass2	51	0.7889	5.3123	10.9795	0.0316	0.0828	0.1449
glass3	101	0.0126	1.9750	2.1223	0.0029	0.0270	0.0302
heartc1	26	4.1335	9.3562	16.6976	0.1253	0.1681	0.2577
heartc2	26	4.1132	18.5736	5.1456	0.1216	0.2705	0.1055
heartc3	26	2.0861	17.3056	17.1746	0.0785	0.2310	0.2458
mushroom1	25	0.0008	0.0047	0.0029	0.0006	0.0012	0.0010
satimage1	26	2.4156	4.2326	3.9235	0.0598	0.0780	0.0719
satimage2	26	2.4828	3.3480	3.5266	0.0011	0.0520	0.0708
satimage3	51	1.0587	3.4445	3.5186	0.0320	0.0583	0.0587
sonar1	6	12.5943	20.1396	20.6354	0.3418	0.4190	0.4298
sonar2	36	0.3454	20.4049	31.5926	0.0164	0.2445	0.3746
sonar3	6	7.3142	24.0657	22.4297	0.2106	0.3817	0.3723
soybean1	26	0.0836	1.2860	1.2878	0.0029	0.0149	0.0181
soybean2	26	0.0104	0.8952	0.8678	0.0008	0.0110	0.0107
soybean3	26	0.0144	0.9811	0.7225	0.0010	0.0120	0.0096
thyroid1	251	0.0035	0.7225	0.9237	0.0005	0.0092	0.0112
thyroid2	101	0.0901	0.9800	0.8360	0.0039	0.0144	0.0131
thyroid3	51	0.2137	0.8085	0.8527	0.0067	0.0136	0.0145
vowel1	26	3.0363	5.7397	6.4922	0.0791	0.1069	0.1112
vowel2	129	0.0030	1.2668	1.4290	0.0012	0.0194	0.0214
vowel3	146	0.0030	1.7881	1.8352	0.0013	0.0249	0.0249

Table 6.7: **Results for best network architecture using GROPING**

usually smaller if the learning algorithm is working in an architecture defined according to **GROPING** guidelines. Also it varies much less between problems and, between the different permutations of the data in each problem. This is a direct result of the type of weight initialisation, *i.e.* the geometric analysis produces not only the required number of weight vectors, the ones computed are situated in the correct positions of the operating space. The small number of iterations executed by the training component so that the architecture achieves complete recognition of the training set are needed to properly adjust the random weight values of the hidden-output connections.

(3) the example showing the smallest difference between the results achieved with **Proben1** and those obtained with **GROPING** correspond to the cancer data. This can be explained taking into account that all the attributes of the original data are continuous-valued and, there is no missing information. This means, that the original data is structured in the most convenient form that should be input to a multi-layer perceptron network, therefore the pre-processing task in **GROPING** is limited to the normalisation of the data and, almost any learning algorithm should produce a reasonable network even with random initialisation.

(4) the other example for which very similar results were produced with both learning algorithms is the diabetes set. All the attributes in this case are also continuous-valued but, the final results achieved on the validation and test sets for both benchmarks cannot really be considered to be exceptional. This is due to the fact that, although none of the attributes are marked as having missing values, the accompanying documentation does suggest that there are many non-informative zeroes not properly explained. This would suggest that the information subtracted from the training set is not reliable therefore, any learning algorithm for building a multi-layer perceptron network for this problem will probably not be able to build a better generaliser.

(5) the best results in the gene problem were registered on the set whose nominal attributes were totally transformed according to the guidelines specified in **GROPING**. This is not surprising considering that the network built is operating on a continuous-valued space constituted with as many dimensions as those found in the original source which, are half of the number of dimensions generated using **Proben1** techniques. It shows that the pre-processing operations used in this network design method are quite effective and, a very convenient asset to this type of learning method. Further evidence supporting this statement is provided, in the suggestions found in many publications describing research on the multi-layer perceptron model, [Bishop 95], [Michie *et al.* 94], [Prechelt 94], all of which indicate the necessity of properly preparing the data if the aim is to design a

network of this type for the application of interest.

6.5 Conclusions

The contents of this chapter give a clear idea of the enormous amount of material that has been developed in the context of machine learning research, particularly in the area of neural networks.

Executing the different activities for evaluating the performance of the **GROPING** algorithm, it became very clear that the tools available for evaluating and comparing the results achieved through the use of different learning algorithms are still in a very primitive level, although projects such as **DELVE** could change this state of affairs in the near future.

It is difficult to objectively and reliably compare different learning algorithms operating on the same type of model, such as those defined to work on the multi-layer perceptron. The task is even harder when the goal is to compare results across different learning philosophies, *e.g.* the effectiveness of a rule-based system against a multi-layer perceptron network. Nothing much seems to have been done to override this difficulty apart from the study described in [Michie *et al.* 94] which, although it must be admitted that it does try to cover a very wide range of learning methods, publishes results which are difficult to interpret. For example, it was never clearly understood what the measure defined representing the amount of space used when executing any of the learning techniques analysed.

Of the different data and machine learning repositories visited during this research the one found to be most appropriate was **Proben1** because:

(1) It included an extensive and clear documentation of its aims and, the manner in which it was designed so that it was relatively easy to compare results against those contained in it.

(2) The type of problems comprising the benchmark, because all of them correspond to real data.

(3) The organisation of all the data associated with the different problems on which the learning algorithm was evaluated.

(4) The detailed definition of the error measures published.

Of all the results compiled the ones that are most relevant correspond to those included in table 6.7 because, they are a clear example of the convenience of developing a multi-layer perceptron

network using **GROPING**. Its effectiveness is a product of the techniques it applies to subtract the information needed to build a network from the training set; the nature of this information which is, for the most part, geometric is very appropriate considering the manner in which the network model operates as, it can be interpreted in terms of a partitioning process generating regions in the operating space, each corresponding to one of the classes relevant to the problem.

Finally, it is clear that for operating spaces which suffer from incomplete information or cannot be properly transformed to continuous-valued form, the multi-layer perceptron model is not the most convenient solution.

Chapter 7

Conclusions and future developments

This thesis has examined in detail the multi-layer perceptron model so as to be able to specify its limitations and representational capabilities. As a result of this study it was possible to define:

(a) a measure to evaluate the structural complexity of the domain on which the final network of a specific application will operate. This is an estimate that provides an initial insight into the difficulty associated with the task of designing an appropriate architecture allowing the assessment of the convenience of using the network model for the application of interest.

(b) a generative learning algorithm (**GROPING**) which integrates certain geometric techniques and statistical procedures, *i.e.* Delaunay triangulations and principal component analysis, that permit the definition of the network architecture for a particular application following a series of well-defined steps.

Section 7.1 contains the conclusions that have been reached throughout the research process which take into account the convenience of exploiting the potential offered by certain geometric algorithms and data analysis techniques for building multi-layer perceptron networks, in addition to those that refer to the specific learning capabilities of the network model. Section 7.2 includes an outline of possible future research that might be developed to extend the results achieved in this work.

7.1 Conclusions

To perform a reliable study of the multi-layer perceptron, it was necessary to specify a theoretical framework on which to base the analysis of the model's functionality. In this particular case, it was decided that a geometric interpretation provided an effective theoretical tool because previous analysis realised by other researchers, [Hertz *et al.* 91], [Bose & Garga 93], give good geometric interpretations of this type of model and, in this study it was possible to build a learning algorithm based on geometric constructs that provides effective means to generate reasonable architectures for different kinds of applications.

The literature review and the subsequent set of experiments realised throughout this research provided material supporting the following set of conclusions regarding the representational capabilities and ease of use of the multi-layer perceptron model:

(1) Different types of activation functions can be used for specifying the network's units. Considering the complete set of choice available a reasonable one is the ***sigmoid*** function because any of the others can be represented in terms of it, the form of the function imbues the model with a better representational capability and, it complies with the conditions that must be satisfied by the activation function of a network that must be subsequently trained by an optimisation procedure.

(2) Of the existing learning algorithms for this network model, the ones supporting some form of generative supervised learning are the most effective because they include mechanisms for an automatic adaptation of the network's architecture which is a definite advantage, given the arbitrary means by which this model's component had to be defined with the first type of algorithms that were available.

(3) It was possible to distinguish three types of generative algorithms: pruning methods, constructive and mixed . The pruning algorithms are the least effective of the three unless a proper mechanism for defining an architecture that might be considered too large is provided as part of the whole design process. At the same time, the most adequate constructive method will be one that not only expands the architecture, *i.e.* incorporates more hidden units, but also includes ways of setting the weight vectors associated to each new unit in the correct position.

(4) Any network design process must incorporate an adequate data pre-processing phase because the model and associated devices for its manipulation are very dependent on the type of data that is provided as input. It is with continuous data in small intervals with which it shows the best behaviour. Although many mechanisms have been defined to

deal with missing information, the different examples realised showed that in general the model does not deal well with this phenomenon.

(5) Apart from the fact that it is possible to interpret the network model in geometric terms, another reason supporting the convenience of defining a learning algorithm based on this type of device stems in the vast amount of research that is being done at this time aimed at developing geometric algorithms that are easy to use and are comprised of a clear set of steps that incorporate no *ad-hoc* procedure.

(6) The manner in which principal component analysis is used in the network pruning process is particularly appropriate because, it makes the most of the functional capabilities of the data analysis technique by applying it to that part of the network building method in which it will have a better use, *i.e.* the detection of redundant and important dimensions.

(7) Using principal component analysis for initialising the weight vectors of an architecture is not always the best alternative considering that there are many applications requiring more weight vectors than those that could be generated with this type of procedure.

(8) The organisation of an architecture so that there are various layers of units between the inputs and the outputs often results in a better generaliser but, as it is possible to reduce a deeply nested structure to a one-hidden layer network, it was reasonable to base the efforts of improving a network design process for multi-layer perceptron networks with at most one hidden layer. Also, considering that there are other representation tools available it is important to understand that for those applications requiring more complex architectures looking at those alternative representations will generally produce better results.

(9) There is a simple tradeoff between achieving a network able to correctly process all the elements in the training set and, one that will perform as an effective *generaliser* on the same problem. This means that, at the time of building a particular network it is necessary to clearly specify the kind of application requiring the architecture, whether it is more important to produce as simple a network as possible able to correctly process the elements of the training set or, if the aim is towards the best generaliser.

(10) Further work needs to be done towards the definition of **generalisation** because different interpretations can be given to the same property. It is necessary to clearly establish what type of measure should be used for determining the generalisation capacity of a network, the mean square error, the percentage of patterns correctly classified, the square error percentage, etc. Each of them has been used by different researchers to evaluate the same property and, even though for the most part it is possible to arrive at the same conclusion

by using any of them, it is also the case that when comparing two different architectures operating on the same problem, one can appear much better than the other when looking, for example, at the percentage of correctly classified patterns instead of comparing the performance through the mean square error achieved by each. Additionally, this property is very dependent on the quality of the training set used for building the architecture and, although it might seemed evident, the definition of the generalisation term should incorporate an explicit indication of this dependency so that it might be possible to assess the possibilities of achieving a good generaliser in view of the available training data.

(**11**) The benchmarks that exist for evaluating the performance of learning algorithms defined to create multi-layer perceptron networks only include results for algorithms that operate on fixed architectures therefore, it is not possible at this time to realise a proper comparison between generative learning algorithms operating on the same model.

The most useful outcome of this research is the generative learning algorithm defined to tackle the weaknesses that were detected on other existing algorithms found in the literature that operate on the same kind of network model. **GROPING** was designed once the representational capacities of the multi-layer perceptron had been carefully understood and it incorporates a series of mechanisms to deal effectively with the hardest task of any network building process, *i.e.* the definition of the number of hidden units that should be incorporated.

The way in which the geometric information is generated and manipulated is also an interesting achievement of this study considering the difficulties that exist at present for finding a geometry implementation for building the Delaunay triangulation of a set of points that does not grow exponentially with the dimensionality of the input when it is higher than 3.

It could be possible to change the particular algorithm that is used at present to generate the geometric constructs used in **GROPING** but, given that its computational complexity is governed by the pruning/evolving phase of the method rather than by the geometric component that is being used there will be no real gain in replacing the implementation that is used at present by another, faster, algorithm.

The study also allowed to discern the complexity of the task of building the network for a particular application. The same cannot be restricted to the execution of a training process because the chances of achieving a successful implementation are minimal unless the problem is very simple. A complete design method must incorporate:

(**a**) An adequate pre-processing phase so that the data might be tailored to the form on which a learning algorithm is at its best, *i.e.* a continuous standardised form.

(**b**) The training, validation and test set must be specified previous to the process of properly building the network.

(**c**) An initial phase that will produce a maximal architecture so that it might be possible to make the best use of a generative-pruning learning algorithm.

(**d**) The algorithm responsible for properly training the network that will be the final output of the whole process.

(**e**) In a generative learning algorithm various networks are created to solve the same problem, therefore a phase must be included to choose from all of them the most appropriate for the particular application. The most important component of this phase is the selection criteria used.

The results contained in table 6.7 are a practical demonstration of the effectiveness of using **GROPING** to generate the multi-layer perceptron network for a particular application because it is clear that reasonable network generalisers were achieved for a variety of classification tasks. When compared against the results obtained for the same problems by the developer of the **Proben1** benchmark it is easy to see that it was possible to achieve better solutions with **GROPING**. It might be argued that this comparison is not sufficient to claim the effectiveness of the algorithm because it is taking place between a learning algorithm operating on a fixed architecture against a generative learning procedure given that, the second type of learning is always better than the first. It was not possible to realise an evaluation against other generative algorithms because there is no proper set of results against which to compare, at least none were found in the literature reviewed. A few results obtained from generating networks with the cascade correlation algorithm were found in the **STATLOG** benchmark and, when compared against them, **GROPING** fared better. Nevertheless, as there are only a small set of results a useful next step to this research could be the design and execution of a proper benchmark evaluating the performance capacity of different generative algorithms. The list could include the optimal brain damage algorithm [Lecun *et al.* 90], optimal brain surgery [B. Hassibi 93], apart from **GROPING**, the cascade correlation algorithm and, other type of classification techniques such as the ones found in the C4.5 software, [Quinlan 93], an example of a decision-tree classifier.

The other contribution provided by this research concerns the measure that gives an indication of the convenience of using the multi-layer perceptron model to build an implementation. It is not an absolute well-proved parameter providing a certain yes/no answer on the effectiveness of proceeding with the use of the model but, it does try to make the best use of the information that is automatically generated in the first stages of the network design process, organising it in

such a way that a better indication of the adequacy of the task that is being executed is made evident before expending computational resources that could be put to a better use. It might also be the case that a bad result is produced because the training set that is being evaluated is a bad sample of the general structure of the operating domain but, this inconvenience applies equally well to all of the network design process because there is no algorithm that will generate a reasonable network with an inadequate training set. The other objection that could be made against the measure is that it evaluates a projected version of the original set, therefore the conclusions reached might be invalid in the authentic setting. Nevertheless, considering that a projected space is never simpler than its original version because the same information is squeezed in a smaller space, achieving a good result guarantees the adequacy of using the model for the extended space and a negative result could be taken as a warning whose validity will be proved as the network development process progresses.

7.2 Possible continuations

The previous discussion clearly shows that although important steps have been taken in the advancement of the process for building a better learning algorithm for multi-layer perceptrons a lot of work still needs to be realised in the area. The ensuing points describe the tasks that are worth pursuing as their outcome will produce important gains in the development of the kind of application for which a multi-layer perceptron is a good alternative.

Of the various paths that might be followed in the next stage of the research that up to this moment has resulted in the definition of the generative-pruning algorithm, **GROPING**, there are two which are considered to be particularly important. Both can be performed in parallel because, the outcome of each does not depend on the outcome of the other. Their description is organised according to the goal that is to be pursued in each case:

(1) enhancing the **GROPING** algorithm by improving the complexity measure. Such a measure needs to be better specified and it is necessary to prove that it does in effect correspond to a reliable parameter for deciding on the issue of the convenience of developing the application of interest with the multi-layer perceptron model. The tasks that are necessary to successfully complete this goal are:

   (a) study other existing geometric mechanisms available for analysing the structure of an input space. Alternatively it might be the case that if the present measure is integrated with some form of statistic it may become more reliable than it is at the present time.

(b) it will be necessary to show the effectiveness of this enhanced version of the complexity measure. To fulfill this goal it is necessary to setup the experiments that will allow a proper evaluation of its effectiveness. As it can only be of a strict statistical nature, given the kind of problem that is being attacked, it is necessary to define as many experiments as possible each comprised of reasonable sized sets of data. This data could correspond to real world problems or, it could be artificially generated to analyse how the measure behaves with data complying with different properties.

(c) implementation of the exact form of the complexity measure finally used integrating it to the already existing implementation of **GROPING**. This task can involve the development of a user-friendly platform for evaluating the structural details of the input space of an application that need not be restricted to the multi-layer perceptron model because, the measure concept is applicable to the general context of knowledge representation.

(**2**) developing a benchmark that will allow the comparison, not only of different learning algorithms for the same network model but also the possibility of measuring the effectiveness of different knowledge representation forms for the same kind of application. The tasks that must be completed to achieve this goal are much more complex than the ones required for the previous one because, it is a much more ambitious aim but, it can be partitioned in stages each of which, when completed, will facilitate the realisation of the ensuing stage. Limiting the definition of a benchmark at this point to generative learning algorithms for the multi-layer perceptron and maybe, learning algorithms for the radial basis function networks, the tasks that should be executed are the following:

(a) familiarisation with the radial basis function model to clearly understand the differences with the multi-layer perceptron and the kind of requirements that a benchmark for such a model should fulfill.

(b) design of the benchmark platform, this step involves comparing the requirements that must be fulfilled to properly evaluate the multi-layer perceptron and the radial basis functions, clearly specifying the common ones and the particular ones. A study of platforms that are already under development to reduce redundant effort is an additional important activity for achieving this goal.

(c) building the data sets for the platform. This is similar to the second task of the previous goal, therefore its outcome could be used in both cases. As the goal pursued at this point is much more ambitious its realisation is more complicated.

(**d**) a proper implementation of the whole platform. This operation requires a very careful design of the user interface so that the benchmark might fulfill the requirements of the research community in the long term.

Other interesting extensions regarding the improvement of the pruning phase of **GROPING**, could correspond to any of the following alternatives:

(**1**) the application of other types of principal component analysis. Using a non-linear version of this technique could result in a faster implementation and maybe a better final network architecture.

(**2**) introducing a simple similarity-based filter to decide on the number of redundant dimensions in the hidden units space, whilst still operating on its principal components.

Finally, the algorithm has only been applied to the design of networks for classification tasks and it would be interesting to evaluate and extend its applicability to the design of networks for function approximation.

Bibliography

[Alpaydin 91] E. Alpaydin. GAL: Networks that Grow when they Learn and Shrink when they Forget. *International Computer Science Institute Technical Reports*, TR-91-032, 1991.

[Amari 92] S. Amari. Information geometry of boltzmann machines. *IEEE Transactions on Neural Networks*, pages 260–271, 1992.

[Aurenhammer 91] F. Aurenhammer. Voronoi diagrams.- a survey of a fundamental geometric data structure. *ACM Computing Surveys*, pages 345–405, 1991.

[Aviles-Cruz *et al.* 95] C. Aviles-Cruz, A. Guérin-Dugué, J.L. Voz, and D. Van Cappel. Deliverable r3-b1-p, task b3: Databases, elena. Technical report, ESPRIT, 1995.

[B. Hassibi 93] D.G. Stork B. Hassibi. Second order derivatives for network pruning: optimal brain surgeon. *Advances in Neural Information Processing Systems*, 5:164–171, 1993.

[Baldi & Hornik 94] P. Baldi and K. Hornik. Learning in linear neural networks: a survey. *Neuroprose ftp-site*, 1, 1994.

[Barber *et al.* 95] C.B. Barber, D.P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hull. Technical report, The Geometry Center, 1995.

[Baum & Haussler 89] E.B. Baum and D. Haussler. What size net gives valid generalization? *Neural Computation*, 1(1):151–160, 1989.

[Bello 92] M.G. Bello. Enhanced training algorithms, and integrated training/architecture selection for multi-layer perceptrons. *IEEE Transactions on Neural Networks*, 3(6):864–875, 1992.

[Bishop 95] C.N. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[Blum & Li 91] E.K. Blum and L.K. Li. Approximation theory and feedforward networks. *Neural Networks*, 4(4):511–515, 1991.

[Bose & Garga 93] N.K. Bose and A.K. Garga. Neural network design using voronoi diagrams. *IEEE Transactions on Neural Networks*, 4(5):778–787, 1993.

[Bourlard & Kamp 88] H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59:291–294, 1988.

[Cater 87] J.P. Cater. Succesfully using peak learning rates of 10 (and greater) in back-propagation networks with the heuristic learning algorithm. In M. Caudill C. Butler, editor, *IEEE First International Conference in Neural Networks*, pages 645–651. IEEE, 1987.

[Clarkson *et al.* 92] K. L. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. In *Proceedings, Symposium Theoretical Aspects of Computer Science*, 1992.

[Conway & Sloane 82] J.M. Conway and N.J. Sloane. Voronoi regions of lattices, second moments of polytopes, and quantization. *IEEE Transactions Information Theory*, 28:211–226, 1982.

[Cybenko 88] G. Cybenko. Continuous valued neural networks with two hidden layers are sufficient, technical report. Technical report, Department of Computer Science, Tufts University, USA, 1988.

[Cybenko 89] G. Cybenko. Approximation by supperpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

[Denker *et al.* 87] J. Denker, D. Schwartz, B. Wittner, S. Solla, R. Howard, L. Jackel, and J. Hopfield. Large automatic learning, rule extraction and generalization. *Complex Systems*, 1:877–922, 1987.

[Dietterich 94] T. Dietterich. Experimental methodology for benchmarking. Slides of talk concerning network benchmarking methodologies, 1994.

[Fahlman & Baluja 94] Scott Fahlman and Shumeet Baluja. Reducing network depth in the cascade-correlation architecture. Technical Report CMU-CS-94-209, Carnegie Mellon University, 1994.

[Fahlman & Lebiere 90] S. Fahlman and C. Lebiere. The cascade correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*. 1990.

[Fahlman 88] S. Fahlman. An empirical study of learning speed in backpropagation networks. Technical Report CMU-CS-88-162, Carnegie Mellon University, 1988.

[Franzini 87] M.A. Franzini. Speech recognition with backpropagation. In *Proceedings of the ninth annual conference of the IEEE engineering in medicine and biology society*, pages 1702–1703. IEEE, 1987.

[Frean 89] M. Frean. The upstart algorithm: a method for constructing and training feed-forward neural networks. Technical Report 89/469, Dept. of Physics, University of Edinburgh, 1989.

[Freund 92] John E. Freund. *Mathematical Statistics*. Prentice Hall, 5th edition, 1992.

[Fritzke 93] B. Fritzke. Growing cell structures - a self-organising network for unsupervised and supervised learning. Technical Report TR-93-026, International Computer Science Institute, USA, 1993.

[Funahashi 89] K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:349–369, 1989.

[Gallinari *et al.* 91] P. Gallinari, S. Thiria, F. Badran, and F. Fogelman-Soulie. On the relations between discriminant analysis and multilayer perceptrons. *Neural Networks*, 4(3):349–360, 1991.

[Gardner 87] E. Gardner. Maximum storage capacity in neural networks. *Europhysics Letters*, 4:481–485, 1987.

[Gardner 88] E. Gardner. The space of interactions in neural network models. *Journal of Physics*, A-21:257–271, 1988.

[Geman *et al.* 92a] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, 1992.

[Geman *et al.* 92b] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–55, 1992.

[Georgiou & C 92] G. M. Georgiou and Koutsougeras C. Complex domain backpropagation. *IEEE Transactions on Circuits and Systems. II: Analog and Digital Signal Processing*, 39(5):330–334, 1992. Talks about sigmoid $\frac{u}{1+|u|}$, two other references within on the same subject.

[Georgiou 93] G.M. Georgiou. The multivalued and continuous perceptrons. 1993.

[Gilbert 61] E.N. Gilbert. Random subdivisions of space into crystalls. *Ann. Mathematical Statistics*, 33:958–972, 1961.

[Hallam & Ross 93] J. Hallam and P. Ross. *Course in NeuroComputing, MSc in Information Technology.* Dept of AI, University of Edinburgh, 1st edition, 1993.

[Harel 92] D. Harel. *ALGORITHMICS: the spirit of computing.* Addison-Wesley, 2nd edition, 1992.

[Hecht-Nielsen 89] R. Hecht-Nielsen. Theory of the back-propagation neural network. In *Proceedings of the International Joint Conference on Neural Networks,*, volume 1, 1989.

[Hertz *et al.* 91] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the theory of neural computation.* Addison Wesley, 2nd edition, 1991.

[Hinton 87] G.E. Hinton. Learning translation invariant recognition in massively parallel networks. In *Proceedings PARLE Conference on Parallel Architectures and Languages Europe*, pages 1–13, 1987.

[Honavar & Uhr 88] V. Honavar and L. Uhr. A network of neuron-like units that learns to perceive by generation as well as reweighting of its links. In D. Touretzky, editor, *Proceedings of the 1988 Connectionist Summer School*, pages 472–484, 1988.

[Hornik 91] K. Hornik. Approximation capabilities of multi-layer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[Hu 90a] C.J. Hu. A geometrical approach to the design of an efficient neural-network supervised learning scheme. In *IEEE International Conference on Systems Engineering*, pages 617–620, 1990.

[Hu 90b] C.J. Hu. A novel, one-step geometrical supervised learning scheme. In *IJCNN - International Joint Conference on Neural Networks*, pages 635–638, 1990.

[Hwang *et al.* 94] J. Hwang, I. Jou, S. Lay, and S. You. What's Wrong with A Cascaded Correlation Learning Network: A Projection Pursuit Learning Perspective. Neuroprose ftp-site, 1994.

[Jacobs 88] R. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307, 1988.

[Ji *et al.* 90] C. Ji, R.R. Snapp, and D. Psaltis. Generalising smoothness constraints from discrete samples. *Neural Computation*, 2(2):188–197, 1990.

[Jolliffe 86] J. Jolliffe. *Principal Component Analysis*. Springer-Verlag, 2nd edition, 1986.

[Judd 87] J. S. Judd. Learning in networks is hard. In Maureen Caudill and Charles Butler, editors, *IEEE First International Conference on Neural Networks*, pages 685–692, San Diego, CA, June 21-24 1987.

[Karnin 90] E.D. Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE Transactions on Neural Networks*, 1(2):239–242, 1990.

[Kleinbaum *et al.* 88] D.G. Kleinbaum, L.L. Kupper, and K.E. Muller. *Applied Regression Analysis and other Multivariate Methods*. Addison Wesley, 2nd edition, 1988.

[Kohonen 82] T. Kohonen. Self-organised formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[Kolmogorov 57] A.N. Kolmogorov. On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition. *Oklady Akademiia Nauk SSSR*, 114(5):953–956, 1957.

[Kramer *et al.* 89] A. Kramer, Vincentelli, and A. Sangiovanni. Efficient parallel learning algorithms for neural networks. *Advances in Neural Information Processing Systems*, 1, 1989.

[Kruschke 91] J.K. Kruschke. Improving generalisation in backpropagation networks with distributed bottlenecks. In *International Joint Conference Neural Networks (Baltimore)*, pages 443–447, 1991.

[Lapedes & Farber 88] A. Lapedes and R. Farber. *How neural nets work*, volume 2, pages 442–456. Anderson, 1988.

[Lecun *et al.* 90] Y. Lecun, J. Denker, and S. Solla. Optimal brain damage. *Advances in Neural Information Processing Systems*, 2:598–605, 1990.

[Lippmann 87] R. Lippmann. An introduction with computing with neural nets. *IEEE ASSP Magazine*, pages 4–22, 1987.

[Martinetz & Schulten 94] T. Martinetz and K. Schulten. Topology representing networks. *Neural Networks*, 7(3):507–522, 1994.

[M.C. Mackey 77] L. Glass M.C. Mackey. Oscillation and chaos in physiological control systems. *Science*, 197(287):287–289, 1977.

[McCulloch & Pitts 43] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[Mezard & Nadal 89] M. Mezard and J.P. Nadal. Learning in feedforward layered networks. *Journal of Physics*, 22:2191–2203, 1989.

[Michie *et al.* 94] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine learning, neural and statistical classification.* Ellis Horwood, 1st edition, 1994.

[Minsky & Papert 69] M.L. Minsky and S.A. Papert. *Perceptrons.* Cambridge, MIT Press, 2nd edition, 1969.

[Motwani & Raghavan 95] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1st edition, 1995.

[Mozer & Smolensky 89] M.C. Mozer and P. Smolensky. Skeletonization: a technique for trimming the fat from a network via relevance assessment. *Connection Science*, pages 3–26, 1989.

[Muller & Reinhardt 91] B. Muller and J. Reinhardt. *Physics of Neural Networks: Neural Networks.* Springer-Verlag, 2nd edition, 1991.

[nana & Ross 96] E. Pèrez-Mi nana and P. Ross. Strategies defined for the development of applications using multilayer perceptrons: A survey. *Expert Systems*, 13(2):133–142, 1996.

[Oja 92] E. Oja. Principal components, minor components, and linear neural networks. *Neural Networks*, 5:927–935, 1992.

[Oja *et al.* 92a] E. Oja, H. Ogawa, and J. Wangviwattana. Principal component analysis by homogeneous neural networks, Part I: The weighted subspace criterion. *IEICE Transactions Information and Systems*, E75-D(3):366–375, 1992.

[Oja *et al.* 92b] E. Oja, H. Ogawa, and J. Wangviwattana. Principal component analysis by homogeneous neural networks, Part II: Analysis and extensions of the learning algorithms. *IEICE Transactions Information and Systems*, E75-D(3):376–382, 1992.

[Okabe *et al.* 92] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams, series in Probability and Statistics.* John Wiley & Sons, 2nd edition, 1992.

[Omohundro 89a] S.M. Omohundro. The delaunay triangulation and function learning, December 1989. International Computer Science Institute.

[Omohundro 89b] S.M. Omohundro. Geometric learning algorithms, October 1989. International Computer Science Institute.

[O'Rourke 94] J. O'Rourke. *Computational Geometry in C.* Cambridge University Press, 2nd edition, 1994.

[Prechelt 94] L. Prechelt. **PROBEN1** — a set of benchmarks and benchmarking rules for neural network training algorithms. Technical report, *Fakultät für Informatik, Universität Karlsruhe*, 1994.

[Prechelt 96] Lutz Prechelt. A quantitative study of current benchmarking practices. *Neural Networks*, 9:457–462, 1996.

[Press *et al.* 86] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling. *Numerical Recipes.* Cambridge University Press, 2nd edition, 1986.

[Quinlan 93] J.R. Quinlan. *C4.5 Programs for Machine Learning.* Morgan Kaufmann, 1993.

[Rasmussen *et al.* 96] C.E. Rasmussen, R.M. Neal, G.E. Hinton, D. van Camp, M. Revow, Z. Ghahramani, R. Kustra, and R. Tibshirani. The delve manual. Technical report, university of Toronto, 1996.

[Reed 93] R. Reed. Pruning algorithms: a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993.

[Riedmiller & Braun 93] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: the rprop algorithm. In *Proceedings of the ICNN*, pages 586–591, apr 1993.

[Riedmiller 94] M. Riedmiller. Advanced supervised learning in multi-layer perceptrons - from backpropagation to adaptive learning algorithms. *International Journal of Computer Standards and Interfaces: special issue on Neural Networks*, 5(1), 1994.

[Robinson 89] A.J. Robinson. *Dynamic Error Propagation Networks.* Unpublished PhD thesis, University of Cambridge, 1989.

[Romaniuk & Hall 93] S.G. Romaniuk and L.O. Hall. Divide and Conquer Neural Networks. *Neural Networks*, 6:1105–1116, 1993.

[Rumelhart *et al.* 86a] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal represantations by error propagation. *Parallel distributed processing*, 1:318–362, 1986.

[Rumelhart *et al.* 86b] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning representations by backpropagating errors. *Nature*, 323, 1986.

[Sarle 94] Warren Sarle. Neural networks and statisitical models, April 1994. Proceedings of the 19th annual SAS Users Group International Conference.

[Schiffmann *et al.* 93] W. Schiffmann, M. Joost, and R. Werner. Optimization of the backpropagation algorithm for training multilayer perceptrons. Technical report, Institute of Physics, University of Koblenz, 1993. networks, hidden units, backpropagation.

[Sejnowski & Qian 88] T.J. Sejnowski and Ning Qian. Predicting the secondary structure of globular proteins using neural network models. *Journal of Molecular Biology*, 202:865–884, 1988.

[Sejnowski & Rosenberg 87] T.J. Sejnowski and C.R. Rosenberg. Parallel networks that learn to pronounce english text. *Complex Systems*, 1:145–168, 1987.

[Smyth 92] S.G. Smyth. Designing multi-layer perceptrons from nearest-neighbour systems. *IEEE Transactions on Neural Networks*, 3(2):329–333, 1992.

[Tesauro & Janssens 88] G. Tesauro and B. Janssens. Scaling relationships in backpropagation learning. *Complex Systems*, 2:39–44, 1988.

[Tveter 96] D. Tveter. Backpropagator's review tveter's home page, 1996. summary of his studies.

[Vogl *et al.* 88] T. Vogl, J. Mangis, A. Rigler, W. Zink, and D. Alkon. Accelerating the convergence of the back-propagation method. *Biological Cybernetics*, 59:257–263, 1988.

[Watrous 87] R. Watrous. Learning algorithms for connectionist networks: applied gradient methods of non-linear optimization. In *Proceedings IEEE First International Conference on Neural Networks*, pages 619–627. IEEE, 1987.

[Webb & Lowe 88] A. Webb and D. Lowe. A hybrid optimization strategy for adaptive feed-forward layered networks. Technical Report 4193, Royal Signals and Radar Establishment, 1988.

[Werbos 74] P. Werbos. *Beyond regression: new tools for prediction and analysis in the behavioral sciences, PhD thesis.* Harvard University, 1st edition, 1974.

[Wilkinson & Reinsch 71] H. Wilkinson and C. Reinsch. *Handbook for automatic computation, Linear Algebra.* Springer-Verlag, 1971.

Appendix A

Generating sets of the Continuous parity problem

The C source code included in this appendix corresponds to the implementation developed in the research to generate patterns for the parity problem with a continuous input domain.

It computes a set of n-dimensional continuous patterns in the interval $[0, 1]$ using the pseudo-random generator functions of the C library, drand48() and srand48(). These functions produce pseudo-random numbers using the linear congruential algorithm and 48-bit integer arithmetic.

Given the type of problem that is being simulated, the output of the program will be a set of P input-output patterns complying with the structure required by a multi-layer perceptron network, as described in chapter 2. The output is one-dimensional and binary. An N-dimensional input pattern $x^p = (x_1^p, x_2^p, ..., x_i^p, ..., x_n^p)$, will have associated a unitary output if the number of elements x_i^p of x^p such that, $x_i^p > 0.5$ is odd, otherwise the output associated to x^p will be zero.

Call with arguments: parity_gen npoints dim > nom_file

```
#include <stdio.h>  #include <math.h>
#include <string.h> #include <malloc.h>

extern double drand48();
extern void srand48();
extern long time();
extern double atof();
extern int atoi();

main(argc,argv)
char **argv;
{
 int def_tr, i, j, k, npoints, dim, tmp_num, def_ts;
 int tam_tr=0, tam_ts=0, point_tr=0, num_ones;
 double tmp_rand, abs_rand;
 float print_rand, width_min, width_max, dif;
 FILE *net_tr;

  dim = 4; width_min=0.0; width_max=0.4; dif=0.0;
  num_ones=0;

  if(argc > 1)
     npoints = atoi(argv[1]);
  if (argc > 2)
     dim = atoi(argv[2]);
  if (argc == 1 || argc > 3) {
     fprintf(stderr, "usage: %s [npoints [dim [min_w [max_w ]]]]]\n",argv[0]);
     exit(1);
  }

  srand48(time(0));
  tam_tr = abs(irint(drand48()*npoints)); tam_ts = npoints - tam_tr;

  if (tam_tr < tam_ts) {
    tmp_num = tam_tr; tam_ts = tam_tr; tam_tr = tmp_num;
  };
  setbuf(stdout, NULL);
  def_tr=0; point_tr=0;

  do {
    num_ones=0;
    for (j=0; j<dim; j++) {
     tmp_rand = drand48();
     if (tmp_rand > 0.5) num_ones++;
     print_rand = (float) tmp_rand; fprintf(stdout,"%8.4f ",print_rand);
    };

    def_tr++;
    if (num_ones%2 == 0) fprintf(stdout," 0\n");
    else fprintf(stdout," 1\n");

   } while (def_tr < tam_tr);

  printf("TRAIN SET SIZE = %4d\n",tam_tr); for (i=0;i<3;i++) printf("\n");
  def_ts=0;
  do {
    num_ones=0;
    for (j=0; j<dim; j++) {
       tmp_rand = drand48();
       if (tmp_rand > 0.5) num_ones++;
       print_rand=(float)tmp_rand; fprintf(stdout,"%8.4f ",print_rand);
    };
    def_ts++;
    if (num_ones%2 == 0) fprintf(stdout," 0\n");
    else  fprintf(stdout," 1\n");
   } while (def_ts < tam_ts);
  printf("TEST SET SIZE = %4d\n",tam_ts);
}
```

Appendix B

Methods for building Multi-layer Perceptrons

The majority of the pruning/constructive methods mentioned throughout this work use information generated by the learning algorithm itself during training. This is produced by modifying the associated cost function or through the definition of additional measuring elements, in both cases it is the result of the integration of statistical and optimisation methods complying with the theoretical elements on which the initial multi-layer perceptron model and related learning algorithm are based. The constructive methods use special control mechanisms to ensure that the evolving architecture does so in an organised way because it is very easy to miss the best architecture and produce one which is not such a good generaliser as the training set becomes too subsumed in it. To this end proper new techniques have been defined many of them based on novel geometrical concepts which have been successfully applied in areas such as robotics and image processing. This type of approach has not been used for improving the results achieved when applying the pruning strategy although it might be worth considering, at least for the definition of the initial **MAXIMAL** architecture required. Section B.1 contains a survey of the pruning/constructive methods based on the same theoretical foundations as multi-layer perceptron and backpropagation, whilst section B.2 comprises a description of the ones developed using new information sources about the structure of the operating space, section B.3 discusses the existing limitations and advantages in the area stressing the importance in the underlying existing relation between Computational Geometry and the multi-layer perceptron model and, section B.4 is composed of the conclusions reached with this study.

B.1 Conventional pruning/constructive methods

Dynamic processes for the generation of an appropriate multi-layer perceptron network for a particular problem have only existed in the literature during the last decade when it became clear that restricting the improvement of the learning process of a network to the manipulation of the learning algorithm's parameters only provided an effective solution to a reduced number of alternatives.

Given that some of these processes have been defined by extending the existing theoretical basis of the multi-layer perceptron model and backpropagation with elements of the same type whilst others incorporate thoroughly novel techniques it is possible to speak of conventional or novel configuration processes and this has been the basis for structuring this publication. This section includes a survey of the conventional pruning/constructive algorithms that have been

defined, the novel ones are described in the succeeding section.

B.1.1 Pruning algorithms

The main drawback this type of approach has is the achievement of an initial **SUFFICIENTLY BIG NETWORK**. In a particular problem it is only possible to consider a network as being too large once the learning algorithm operating on it has already converged as this is the only situation that allows for the generation of information conducing to the fact that the architecture comprises redundant elements. This limitation might be overcome if it is possible to apply some technique to the known elements of the problem of interest so that the definition of a maximal network, at least in terms of this particular set, can be ensured. It must be pointed out that even though this step in the network development through a pruning process is crucial it is very little mentioned in the literature reviewed.

Generally for this type of algorithm the main idea is that the weights of all unnecessary connections and the output of all unnecessary units will approach zero. It is possible to detect this in two ways [Alpaydin 91]:

- Explicitly computing how important the existence of a connection/unit is in keeping the error low after the network has been trained so that a number of the least important may be deleted. The information generated for deciding on the importance of the units/connections is generally of a heuristic nature.

    1. The skeletonisation procedure [Mozer & Smolensky 89] trains a network till a certain performance criterion is met. The relevance of each connection is then computed in terms of the partial derivative of the absolute error with respect to the connection, a more reliable calculation than the conventional squared error given the manner in which its derivative decreases when the error is low.

    2. [Karnin 90] computes the sensitivity in the same way but sums the values computed throughout learning instead of computing once at the end. The advantage it has over the skeletonisation procedure is that it requires less additional computations as the same type of error measure as backpropagation is employed when generating the sensitivity factors.

    3. Optimal brain damage [Lecun *et al.* 90] uses an information theoretic measure to compute the saliency of a connection using the second derivative of the error function. Training proceeds until the error comes down to a certain value at which point saliencies are computed, a number of the least salient are deleted and the remaining network is retrained.

- Instead of approximating how much the error will change if the unit/connection is eliminated, it is possible to modify the related learning algorithm so that after training, the unnecessary connections/units will have zero weight/output.

    1. It is possible to build a tendency in the learning algorithm to have those weights that are not relevant decay to zero by decrementing them by a certain factor at each weight update, an approach described by [Hertz *et al.* 91] . The important weights are set away from zero whilst the useless ones simply decay to zero value. Other variants of the weight decay rule can be found in (Chauvin 1989; Hanson 1989; Ishikawa 1990; Plaut 1986). The main methodological difference between them, apart from the fact that each defines a unique decay rule, is that some favour decaying weights whilst others explicitly involve decaying hidden units.

2. Another alternative is to use the information theoretic idea of minimum description length and add a term to the cost function that penalises network complexity (Weigend, Rumelhart, Huberman 1991). During gradient descent, the algorithm will settle to the network that has the best trade-off between error and complexity.

3. [Ji *et al.* 90] modify the error function to minimise the number of hidden nodes and the magnitude of the weights. They consider a single-hidden-layer network with one input and one linear output. The error is defined as the sum of two terms, ϵ_0, the normal sum of squared errors, and ϵ_1, a term measuring node significance. The second term makes the algorithm favour solutions with fewer significant hidden units. Once an acceptable level of performance is achieved, small magnitude weights can be removed and training resumed. They note that the modified error functions increase the training time.

4. [Kruschke 91] describes the local bottlenecks method in which the hidden units compete to survive. The degree to which a unit participates in the cost function computed by the network is measured by the magnitude of its weight vector. Units are redundant when their weight vectors are nearly parallel /antiparallel and they compete with others having similar directions. To detect this the weight vector is treated as a separate parameter designated the gain, a positive term that approaches zero as the associated unit's usefulness decreases.

As has already being mentioned the main problem with all these algorithms is the need to operate on an architecture that must be initially too large otherwise it is not possible to detect which of the units and/or connections are in effect redundant and a network can only be said to be too large if the learning algorithm processing it manages to converge, ie. if it is at least able to recognise all the elements in the training set. Another drawback is in relation to the nature of the heuristic information any of them uses as it makes the decision process extremely subjective and not very reliable. The concepts of saliency, acceptable level of performance, sensitivity are all extremely volatile therefore easily tailored to anybody's preference.

B.1.2 Constructive algorithms

All these processes start from a simple network and add units and/or connections to decrease the error. The building process cannot be done in a straightforward manner (Alpaydin 1991) especially in networks where associations are distributed over a number of shared connections. It is necessary to define mechanisms whereby addition of a new unit improves success without corrupting the harmony normally expected in a distributed environment, a difficult task that might be considered the main component in any of these methods and should therefore receive the utmost attention by any potential user.

The advantage they have over the pruning methods is that no important a-priori decision with respect to the network architecture needs to be taken. Also many of these constructive alternatives include a pruning phase to clean the architecture of possibly useless elements which means there are particular cases which take into account the advantages of both approaches.

1. The first incremental learning algorithm is the Restricted Coulomb Energy (*RCE*) model (Reilly, Cooper, Elbaum 1982). Associated with each unit is a number of prototypes where a prototype gets activated only if the input falls into its domination region, determined by a distance computation followed by a thresholding. If an input does not activate any prototype, a new prototype is created at that position with an initially large domination region. Prototypes activated for inputs that belong to different classes are penalised by having their regions decreased which is done by modifying the threshold. It also includes a tuning process for adjusting units closer to class boundaries.

2. Recruitment learning (Diederich 1989), is used in the case of structured connectionist networks where a previously free unit is committed to represent a new concept and required connections are built up dynamically. This is a one-shot learning algorithm, i.e. one iteration is sufficient to learn a new concept.

3. The upstart algorithm (Frean 1989), was developed to build reasonable multi-layer perceptron to solve classification tasks in the binary domain; the networks it develops have a threshold activation function in all its layers. The basic idea is that a unit builds other units to correct its mistakes, these new units are termed daughter units of the one that induced their creation and because the whole operation is taking place in the binary domain and due to the type of activation function employed it is possible to estimate the value the new connections should have so that the network classifies correctly a greater number of patterns. The final architecture has a hierarchical tree structure which was a point against the strategy, whereupon the authors in [Frean 89] extended it so that it will generate a one-hidden layer network and be able to manage multiple outputs. It is a convenient algorithm but only for building networks that express boolean mappings.

4. The Cascade correlation algorithm developed by [Fahlman & Lebiere 90] evolves its architecture so that the hidden units establish high-order non-linearity in approximating the residual error. The connection weights associated to each new candidate unit are trained while freezing those of existing units. In fact, in cascade correlation only a single layer of weights (either hidden/output) associated with each candidate hidden unit is being trained at any given time based on a maximum correlation criterion for hidden weights and a minimum mean square error criterion for output weights, considerably speeding the training process. Its main disadvantage lies in the complex architecture finally developed as there is just one hidden unit in each layer. This shortcoming has been recently solved [Fahlman & Baluja 94] with a variant of cascade correlation which allows the creation of more than one unit in a hidden layer.

5. The divide and conquer neural network algorithm (*DCN*) described in (Romaniuk, Hall 1993), creates a network architecture combining a divide phase and a conquer phase into the learning process which begins operating on a minimal architecture comprised only of the input nodes because the hidden and the outputs required are created during training. The learning algorithm is independent of the (learning rule, activation function) pair chosen. In the conquer phase a new cell is added, and it serves as the first cell on a new layer of the network, if its output correctly classifies all the examples for the output currently being processed it serves as an output cell. The divide phase adds a cell in the same hidden layer as the last conquer cell, it concentrates on learning only the examples that were not already learned by cells on the same hidden layer. The final network architecture does not exactly conform to the multi-layer perceptron given it permits connections between nodes not complying with the feedforward rule in a similar way as the topologies produced by the *CCN*. Its general structure guarantees maximum pattern recognition rate, at least for the training set and the experiments performed by the authors show a good degree of generalisation.

Even though in the first instance this approach seems to be more appropriate than the pruning strategy because it does not require the initial definition of a complete and correct network architecture most of the alternatives effectively developed tend to generate more complex architectures than the one needed for the problem under consideration. This is due to the kind of information used in the decision process concerned with the generation of additional connections and/or units. As the state of the network at that moment does not allow one to decide whether it is enough to add a unit and/or connection to solve the problem or if a more drastic change, ie the introduction of an additional layer is a better action, the applications defined usually perform the second one given that it is always the case that the introduction of an additional layer greatly improves the learning capacity of a network based on the multi-layer

perceptron model. The inefficiency of this action stems from the little concern it takes of Cybenko's results in relation to the capacity of a one hidden layer multi-layer perceptron and is what makes the pruning algorithms seem more effective if it is possible to define an initial network showing a high pattern recognition rate.

B.2 Novel Generative algorithms

The methods described in this section differ from those previously mentioned in the type of information and approach employed when creating/deleting an element of a network. The most arresting characteristics many of them show are: a) the use of more than one type of network model resulting in a hybrid architecture; b) the integration of constructive and pruning elements within one approach, trying to profit from the advantages of both. The ones reviewed are:

1. [Alpaydin 91] describes the grow and learn algorithm, a combination of constructive and pruning operations. It generates a hybrid architecture, the input and output layer conforming to the multi-layer perceptron whilst the hidden layers comprise a winner take all network which is empty at the start of the training process. When presented with a new pattern if the architecture in its present form is unable to correctly classify it the generation of a new element within the hidden layers takes place and the weights of the different connections are calculated in such a way as to produce a one-shot LA. Its pruning phase is performed during a sleep mode in which the network is closed to the environment, the inputs are generated by the system itself, and units that are no longer necessary due to recent additions are removed.

2. [Honavar & Uhr 88] provide a theoretical framework for generative learning processes which are able to produce parallel architectures of simple computing elements whose particular form and size is a function of experience. It includes the different mechanisms that any generative algorithm might include emphasising those they believe are the most appropriate for a coherent, sound and effective algorithm. A control mechanism based on an error-driven minimal generation strategy is considered the most adequate way for striking a suitable balance between the existing nodes and the nodes that will be subsequently incorporated and which are not restricted to a particular layer because the algorithm includes conditions that must be satisfied for the creation of units and/or layers. Each new unit is connected to all the existing units in the previous layers. It also includes a pruning phase within the global generative process to eliminate possible ineffectual network components. The method they implemented is based on the generation of a recognition cone able to modify its own topology by growing links and recruiting units whenever the related learning algorithm ceases to improve its performance during the traversal of the training set and subsequent update of the existing connection's weights.

3. The algorithm developed by [Smyth 92] is the result of the integration of multi-layer perceptron techniques and nearest-neighbour systems (*NNS*) theory, this last for determining the number of hidden units that a one hidden layer network might require to solve the problem of interest and, the values of the weight vectors associated to each of these units. The estimation process involves generating the cluster centers from the training set, calculating the boundaries between the regions these produce and which were defined as the perpendicular bisectors of the inter-center segments. Special considerations are used to initialise the connections associated with the network's bias term. All this information is used to build a sorted boundary list which will be the input to an iterative process that runs through it generating a hidden unit to deal with each border so that a greater number of the elements in the training set might be correctly classified

ensuring no redundant hyperplanes are created. The weights of the hidden-output connections are obtained by inverting the sigmoid activation function and considering the desired targets of the training set. A problem with this strategy is that the matrix that must be calculated to correctly determine the values of the hidden-output connections is not invertible therefore an additional step comprising a singular value decomposition process is required. Another problem is that there is no guarantee that the weight vectors calculated with the *NNS* will always result in the space separators required but it permits the generation of an initial architecture whose weight vectors will generally be much closer to the desired global minimum than a random initialisation, therefore the learning algorithm selected to train and fine-tune the network has a good chance of reaching a good solution in a reasonable amount of time. This implementation can provide a good alternative for the network design if the number of regions (clusters) does not have to be specified as part of the input but is a product of the *NNS*, if this is not the case there is an important element of information associated to the operating space which should be part of the architecture estimation process and is not.

4. [Bello 92] provides an example of what a dynamic generation process of a multi-layer perceptron should not be. It employs non-linear least squares and quasi-Newton optimisation techniques in its learning algorithm and integrates it with an estimation process that selects the best network architecture for the problem of interest from an initial pool of possible solutions. The first thing that is not adequate is the selected learning algorithm because various analysis have shown that second-order techniques, eg. the quasi-Newton process, are not the most efficient of alternatives for training multi-layer perceptron. Moreover the architecture generation process requires an initial **SET** of possible solutions, a very difficult condition to satisfy given the amount of work involved in coming up with just one feasible alternative. The fact that it does not provide an option for dealing with the situation of choosing a completely wrong initial set can also be taken as a point against this particular implementation.

5. The growing cell structures algorithm (*GCS*) defined by [Fritzke 93] integrates not only the pruning and constructive strategies but also the supervised and unsupervised philosophies of the learning algorithm for training a network model. The unsupervised learning is used to train the network's hidden layer which is comprised of a set of radial basis function units whose synaptic vectors are in principle adapted according to the strategy proposed by [Kohonen 82]. The variations introduced by the algorithm were designed to ensure that throughout the training process the existing units and connections in the hidden layer satisfy the specifications associated with a Voronoi tessellation, a geometrical arrangement representing minimum distances between sets of points which has great applicability in a network design process. The fact that at any moment the hidden units must comply with the requirements of a voronoi tessellation means that the generation/deletion of a unit must be carefully controlled, therefore this particular set of update operations is a fundamental constituent of the *GCS*. The supervised algorithm trains the hidden-output connections ensuring that a balance is maintained with the structure generated in the hidden layer. The error produced during the supervised phase is used to determine whether more units are required in the hidden layer and because different and somewhat independent network components are updated in each phase it is possible to perform both tasks in parallel. There are no formal proofs showing that the *GCS* strategy always produces a voronoi diagram at the hidden layer but the 2-dimensional problems described show that such is the case and the authors mention that it might be possible to extend the results to higher dimensions. *GCS* is an adequate contribution to generative algorithms not only because it tries to exploit many of the strategies and possible alternatives that have been defined in this area but also because it presents a possible application of certain geometrical mechanisms to solve the problem of generating an appropriate network architecture which seems to be a very reasonable option.

6. [Bose & Garga 93] describe an approach which draws upon some results in computa-

tional geometry and graph theory to provide a solution to the multi-layer perceptron design problem. The learning algorithm initially calculates from the points of every class comprising the problem of interest the associated convex hull. From each of them it is possible to generate a voronoi diagram occupying a particular region in feature space that might have a certain degree of intersection with the regions occupied by the voronoi diagram of the other classes. The authors differentiate two types of intersection each requiring a particular solution. The first one occurs between a pair of classes; the other type involves more than two classes and corresponds to the case where the intersection of every pair taken separately is empty but not that of the set of joint classes, the pairwise linearly separable problem, which is solved in [Bose & Garga 93] with two layers in the network, one comprised of the units producing the hyperplanes required by each pair of classes and the second containing as many units as there are classes. The connections between the hidden layers guarantees the units will be correctly activated, ie. each of the hyperplanes in the first layer will activate the unit in the second layer corresponding to the class of interest. To solve the intersection problem between a pair of classes requires the construction of a cluster of clusters through the generation of an intersection of hyperplanes if the resulting cluster is convex or the union of a set of convex regions in the case of a non-convex cluster. Each of the last two alternatives requires a particular architecture, for the convex cluster a two hidden layer architecture solves the problem whilst a non-convex one is tackled with a three-hidden layer network. Although the learning algorithm is not exactly one-shot it does require a much shorter number of iterations than the standard backpropagation because the precise value of each of the connections can be explicitly calculated using information about the position of each of the convex hulls generated. The inconvenience with this approach is the great number of layers it is generating probably due to the fact that its units use simple threshold functions, the simplest and most reliable way to generate hyperplanes. It should be possible to improve on this approach introducing an element of non-linearity through the use of sigmoid functions without forgetting to exploit on the new sources of information suggested, ie. the computational geometry concepts.

B.3 Discussion of the alternatives considered

To ensure the successful completion of any algorithm it is necessary to clearly specify its initial operating conditions. In the particular case of the pruning approach it is compulsory they begin to work on a network architecture that provides a solution to the problem because only then is it possible to conclude anything in relation to the appropriateness of its size. This situation favours the constructive algorithms but for the majority of tasks these methods usually generate network architectures that are more complex than the one required.

This state of affairs has been tackled in (Alpaydin 1991; Honavar 1988; Fritzke 1993), with the development of integrated learning algorithms, trying to profit from the advantages each approach offers. To this end it is necessary to produce a complete and reliable block of knowledge from which it might be possible to conclude whether the inherent degree of complexity of a particular architecture is the one really required to effectively solve the problem of interest. Many different mechanisms are employed in the generative algorithms described for producing this kind of knowledge, each has its advantages and seems specially appropriate mainly because of the way its developers apply it in the network context.

Of all the set of possibilities the most useful are the ones subtracted from computational geometry given the close link that exists between its underlying philosophy and that of the multi-layer perceptron, a matter that has been shown in various studies advanced in the last couple of years not only within the realm of generative algorithms and not only restricted to computational geometry as the potential of the different existing geometrical theories has been

exploited in various studies . [Hu 90a], [Hu 90b] describe works that use convex-cone geometry to study the conditions the one-hidden-layer, threshold function network model should satisfy so that it might capture the form of binary mappings in n-dimensional input-output domains. In this case it is possible to write the form of the hyperplanes required in terms of the inner-product of all the input-output pairs comprising the training set and a group of vectors generated through other associations between the elements of the same set. If this new group of vectors forms a convex-cone then it is possible to define a network to represent the mapping expressed by the training set under scrutiny.

[Amari 92] describes the equivalence between a set of Boltzmann machines with a fixed topology and a geometrical manifold of high dimension establishing the convenience of studying the geometry of a neural manifold over the behaviour of a single network for achieving a better understanding of the capabilities and limitations of networks of a fixed topology .

[Martinetz & Schulten 94] describe the Topology Representing Network model which is based on the fact that for a number of information processing tasks the problem that has to be solved efficiently can be reduced to a geometric one dealing with the proximity of points in a metric space, for this reason they designed a learning algorithm comprised of a hebbian adaptation rule with winner-take-all like competition which when it is executed develops connections between a given set of neural units with neighbouring receptive fields whilst eliminating those existing between non-neighbouring units, the resulting network architecture coincides with the Delaunay Triangulation of the points in the training set. The delaunay triangulation is a concept in computational geometry very closely related to the voronoi diagram, one is the dual of the other, both provide very good information on the existing proximity between the points from which they are generated and this is what makes them particularly attractive sources of information for the design process of a network.

The work developed by [Omohundro 89b], (1989b) is not based on a particular network model or generative algorithm, nevertheless it is an important contribution to this research because it emphasises the existing potential in using Geometrical concepts in the specification of a learning process. It describes a learning algorithm in [Omohundro 89b] which combines k-d trees and delaunay triangulation arguing that their usefulness stems from the point proximity information embedded in them given that this is the main device responsible for the generalisation property found in any kind of biological learning.

B.4 Conclusions

A general framework providing a tool for the categorisation of the different existing methods for the development of a multi-layer perceptron model was presented. Within it many different alternatives were outlined, describing the reasons which made a particular type of method more convenient with respect to the others by pointing out its main advantages and/or weaknesses from a theoretical perspective given the properties of the concepts upon which it is based.

Overall the conclusions reached can be summarised in the following way:

1. The generative algorithms offer more appropriate methods for developing a network application given that they require much less data pre-processing and network/learning algorithm parameter specification before they are applied to the task of interest.

2. There are three main types of generative algorithms: pruning methods, constructive and mixed. The pruning alternative is the least effective given that an operating network must be generated before proceeding to execute the algorithm. On the other hand the constructive methods usually produce architectures that are more complex than the one required. This situation makes an integrated approach the most adequate given it tries

to benefit from the qualities provided by both the building/pruning alternatives.

3. Any of the methods described involves an important information processing task. The best results were obtained with those producing the correct amount and type of information. For a classification task good information is the one providing the algorithm with a clear description of the structure of the network's operating space and the best devices that have been provided to produce it belong to Geometry. Some use sources defined in Classifiers theory given that this area is also concerned with the structure of its operating space. Even though these concepts have other names and seem to require special processes for their calculation the underlying philosophy makes use of a few basic geometric concepts, eg. euclidean distance, therefore it is a much cleaner and direct approach to apply geometry theory in the first place.

4. The algorithms that make use of geometry concepts either operate different network models or are applied to the simplest of multi-layer perceptron models, ie. the threshold activation function model. Given the potential such application should be extended to the more powerful non-linear multi-layer perceptron.

Appendix C

Geometry terminology

As a major part of this research is based in geometry, the present appendix includes the basic terminology on which most of the ideas applied is based. The concepts were for the msot part subtracted from [O'Rourke 94] and [Okabe *et al.* 92].

1. **Non-collinearity assumption:** for a given set of points $\mathbf{X} = \{x^1, x^2, \cdots, x^n\} \in R^N$, $(n \geq 2)$, $x^1, x^2, \cdots, x^n$ are not on the same line.

2. **Non-circularity assumption:** given a set of points $\mathbf{X} = \{x^1, x^2, \cdots, x^n\} \in R^2$, $(4 \leq n < \infty)$, there does not exist a circle, C, such that $x^{i1}, \cdots, x^{ik} \in \mathbf{X}$, $k \geq 4$, are on C, and all points $\mathbf{X} \setminus \{x^{i1}, \cdots, x^{ik}\}$ are outside C.

3. **N-dimensional cartesian product:** if $X_1, X_2, \cdots, X_N$ are non-empty sets, then the N-dimensional cartesian product $X_1 \times X_2, \cdots, X_{N-1} \times X_N$ is the set of ordered N-tuples $(x_1, x_2, ..., x_N)$ with $x_i \in X_i, \forall i \in [1, N]$. When $X_i = R, \forall i \in [1, N]$, the cartesian product is called the N-dimensional real Cartesian space.

4. **point x^p:** in the N-dimensional real cartesian space R^N, an N-tuple or the cartesian coordinates $(x_1^p, x_2^p, ..., x_N^p)$ indicate a point denoted by x^p.

5. **norm(x^p):** the norm of vector $x^p \in R^N$, $\|x^p\|$, is given by
$$\|x^p\| = \sqrt{((x^p)^T * x^p)} = \sqrt{(\sum_{i=1}^{N} (x_i^p)^2)}$$

6. **euclidean-distance(x^p, x^m):** the euclidean distance from point $x^p = (x_1^p, x_2^p, ..., x_N^p)$ to point $x^m = (x_1^m, x_2^m, ..., x_N^m)$ is given by:
$$\begin{aligned}\|x^p - x^m\| &= \sqrt{(x^p - x^m)^T * (x^p - x^m)} \\ &= \sqrt{(\sum_{i=1}^{N} (x_i^p - x_i^m)^2)}\end{aligned}$$
The N-dimensional real cartesian space with the euclidean distance is denoted the N-dimensional euclidean space.

7. **open ball:** for a point $c \in R^N$ and $\epsilon > 0$, the set: $N_\epsilon = \{x \mid \|x - c\| < \epsilon\}$ is called an **open ball** with radius ϵ centred at c. Using the definition , the set $X \subset R^N$ is **open** if every point in X is the centre of some open ball entirely contained in X; the set $X \in R^N$ is **closed** if $R^N \setminus X$ is open in R^N.

8. **boundary point x^p:** a point $x^p \in R^N$ is a boundary point of X, if every open ball centred at x^p contains a point of X and a point of $R^N \setminus X$. The set of all boundary points of X is the **boundary** of X and it is denoted by ∂X.

9. **hyperplane H:** the set
$$H = \{x \mid x = \sum_{i=1}^{N-1} \lambda_i x^i + (1 - \sum_{i=1}^{N-1} \lambda_i) x^N, \lambda_i \in R, i \in [1, N-1]\},$$

where $x^2 - x^1, \cdots, x^N - x^1$ are linearly independent, corresponds to a **hyperplane** in R^N. Alternatively, the hyperplane H can be written as:

$$H = \{x \mid \Lambda^T \mathbf{x} = b, \Lambda, \mathbf{x} \in R^N, \|\Lambda\| \neq 0\}$$

10. **half-space** R_h: a hyperplane H splits a space into two disjoint regions. The region defined by:

$$R_h = \{x \mid \Lambda^T \mathbf{x} \leq b, \Lambda, \mathbf{x} \in R^N, \|\Lambda\| \neq 0\},$$

is referred to as a **closed half space**.

11. **bisector**(x^k, x^m): given two points $x^m, x^k \in \mathbf{X}$, the line $b(x^m, x^k)$ is referred to as the **bisector** of the segment $\overline{x^m x^k}$, if every point in the line $b(x^m, x^k)$ is equidistant from x^m and x^k. Formally $b(x^m, x^k)$ can be written as:

$$b(x^m, x^k) = \{x \mid \|x - x^k\| = \|x - x^m\|\}, k \neq m$$

12. **convex set X:** a set $\mathbf{X} \in R^N$ is convex, if for any two given points, x^k, x^m, such that, $x^k \in \mathbf{X}, x^m \in \mathbf{X}$, the line segment $\overline{x^k x^m}$ joining the points is contained in $\mathbf{X}$, *ie.*

$$\lambda x^k + (1 - \lambda) x^m \in \mathbf{X}, \forall 0 \leq \lambda \leq 1,$$

otherwise $\mathbf{X}$ is a **non-convex** set. In particular if for any two points $x^k, x^m \in \mathbf{X}$, the relation

$$\lambda x^k + (1 - \lambda) x^m \in \mathbf{X} \setminus \partial\mathbf{X}, \forall 0 \leq \lambda \leq 1,$$

holds, the set $\mathbf{X}$ is said to be **strictly convex**.

13. **convex hull:** of a set of points $\mathbf{X} \in R^N$, $\mathbf{CH(X)}$, is the set of all convex combinations of $N + 1$ (or fewer) points of $\mathbf{X}$. Alternatively it is the intersection of all convex sets that contain $\mathbf{X}$. An important characteristic of this geometric construct is that it corresponds to the smallest convex set that contains the set $\mathbf{X}$, therefore if $\mathbf{X}$ is a convex set, the convex hull of $\mathbf{X}$ is $\mathbf{X}$ itself.

14. **simplex:** in R^N, a **simplex** is the convex hull of any set of $N + 1$ points which do not lie on one hyperplane in R^N.

15. **tessellation:** let $\mathbf{S}$ be a closed subset of R^N and $\mathcal{S} = \{S_1, S_2, \cdots, S_t\}$ where S_i is a closed subset of $\mathbf{S}$. If the elements in $\mathcal{S}$ satisfy

$$[S_i \setminus \partial S_i] \cap [S_j \setminus \partial S_j] = \emptyset, i \neq j, i, j \in (1, t), \text{ and}$$
$$\bigcup_{i=1}^{t} S_i = \mathbf{S}$$

then the set $\mathcal{S}$ is a **tessellation** of $\mathbf{S}$.

Appendix D

Hypothesis testing on the error measure

Once the idea of initialising the network architecture with the information subtracted from the geometric constructs, generated from the training set associated to the problem of interest, had been clearly specified, it was necessary to devise an adequate mechanism to prove its usefulness. In this context, a statistical inference procedure that will allow the comparison of, the error achieved during training operating on the network instantiated with the geometric initialisation against, the average of the error achieved by the same training process working on the same network architecture but, instantiated with various random initialisations provided an appropriate tool.

As the statistical inference used is the hypothesis testing method, it involves performing a statistical test of a null hypothesis concerning a parameter that complies with certain properties. In this context, the sample data is used to test whether the estimated value for the parameter is different enough from a hypothesised value to conclude that the null hypothesis is unlikely to be true. The general procedure followed when executing this test corresponds to the one described in [Kleinbaum *et al.* 88] and, is comprised of the following steps:

1. Check the assumptions concerning the properties of the underlying variable being measured that are needed to justify the use of the testing procedure being considered.
2. State the null hypothesis H_0 and the alternative hypothesis H_A.
3. Specify the significance level α.
4. Specify the test statistic to be used and its distribution under H_0.
5. Form the decision rule for rejecting and not rejecting H_0, *i.e.* specify the rejection and acceptance regions for the test.
6. Compute the value of the test statistic from the observed data.
7. Generate the conclusions concerning the rejection or non-rejection of H_0.

For the case of interest in this research, the previous steps were realised in the following way:

1. **Assumptions:** the error measure corresponds to the **minimum square error** registered at different stages of the training process; the value taken into consideration for this analysis is the error registered at the end of the training procedure. There is one such value

set_id	$\overline{mse\_tr}$	std_dev(mse_tr)	mse_gm_tr	num_iter
1	0.0104	0.0003	0.0101	15000
2	0.0037	0.0001	0.0039	205
3	0.0481	0.0082	0.0120	15000
4	0.0141	0.0002	0.0140	15000
5	0.0076	0.0014	0.0057	23
6	0.0039	0	0.0039	15000
7	0.0205	0.0088	0.0111	15000
8	0.0179	0.0004	0.0167	15000
9	0.0083	0.0003	0.0070	437
10	0.0094	0.0024	0.0031	38
11	0.0085	0.0002	0.0082	15000
12	0.0657	0.0078	0.0149	15000
13	0.0050	0.0002	0.0049	15000
14	0.0082	0.0031	0.0084	9
15	0.0068	0.0014	0.0055	15000
16	0.0040	0.0005	0.0036	15000
17	0.0520	0.0118	0.0066	16
18	0.0772	0.0531	0.0045	26
19	0.0082	0.0037	0.0087	38
20	0.0058	0.0001	0.0056	15000
21	0.0058	0.0002	0.0052	15000
22	0.0040	0	0.0040	15000
23	0.0168	0.0074	0.0043	14
24	0.0180	0.0135	0.0176	18
25	0.0125	0.0004	0.0121	15000
26	0.0286	0.0099	0.0348	8
27	0.0113	0.0002	0.0048	15000
28	0.0138	0.0004	0.0135	15000

Table D.1: **data collected on training set for hypothesis testing**

for each of the random initialisations and, one **minimum square error** value associated with the training process when operating on the architecture initialised with the weight vectors that resulted from the geometric analysis. As there are 28 problems, 28 possible conclusions regarding the null hypothesis and the alternative hypothesis can be built with the statistical procedure. The network architecture in each case was initialised with 30 different sets of randomly generated weight vectors, using values in the $[-1.00, 1.00]$ interval. It is feasible to assume that the distribution underlying the error measure that is of interest, is the normal distribution applying the results of the central limit theorem, which concern the distribution governing the mean of a sufficiently large sample. Therefore a hypothesis testing procedure is applicable.

2. **Hypotheses** H_0: there is no significant difference between, the error achieved at the end of training the network architecture initialised with the geometric information and, the error achieved training the same network for the same number of iterations but, randomly initialised at the beginning of the training process; H_A: there is a significant difference between the error measures of interest, *i.e.* the error achieved at the end of training the architecture initialised with the geometric information is smaller, than the error achieved training the same network but using weights randomly initialised at the beginning of the training process. Considering:

   $\overline{mse}$: average of the **minimum square error** obtained with random initialisations,

mse_{gm}: **minimum square error** computed in the training process working on the network

initialised with the geometric analysis, the hypotheses can be mathematically expressed as:

$$H_0 : mse_{gm} = \overline{mse} \quad H_A : mse_{gm} < \overline{mse} \tag{D.1}$$

3. **Significance** α: The significance level $\alpha = 0.025$. It represents the probability of making a type I error during the inference procedure. Type I error refers to, incorrectly rejecting the null hypothesis H_0.

4. **Test statistic:** considering that the parameter being analysed is normally distributed and, that there is information of the mean, standard deviation, of a randomly selected sample of that parameter, it is possible to define the random variable T using the same variables as in steps 2 and 3:

$$T = \frac{(\overline{mse} - mse\_gm)}{\frac{S}{\sqrt{29}}} \tag{D.2}$$

T has the **t distribution** with 29 degrees of freedom. For this reason, the subsequent analysis of the error consists in looking up tabulated percentiles of this distribution for each of the 28 artificial datasets built with DGP-2.

5. **Decision rule:** it is defined dividing the possible values of T into two sets: one of them is the *rejection region*, and consists of those values of T for which H_0 is rejected. The other set, the *acceptance region* consists of the T-values for which H_0 is accepted as true. The critical region is determined by choosing a point from the t tables, called the **critical point** which defines the boundary of the acceptance and rejection regions. The value chosen for this study was $t_{29,0.975} = 2.045$. Thus reject H_0 if the following condition is satisfied:

$$\overline{mse} > (mse\_gm + (\frac{S}{\sqrt{29}}) * 2.045) \tag{D.3}$$

A conclusion was reached regarding the previously defined hypothesis test for each of the 28 problems built with DGP-2. Furthermore, given that in supervised learning the same error is computed for two different types of data, *i.e.* the training and, test data, it was possible to obtain 2 types of results with the same inference procedure:

1. using the **minimum square error** achieved with the training set at the end of the training operation. Table D.1 contains the errors computed on the training set of each of the problems considered. The last column in the table contains the iteration at which the error was registered.

2. using the error measured on the test set. The values obtained in each problem for this particular measure are in table D.2.

Table D.3 contains the outcome of the statistical analysis. The fourth and fifth columns of the table indicate whether the null hypothesis H_0 was or not rejected according to the evidence available. The first three columns correspond to a confidence interval test that provides additional information on how similar or otherwise the errors turned out to be. The one used in this evaluation was defined by the formula:

$$\overline{mse} \mp t_{29,0.975} \tag{D.4}$$

which gives the $100(1-\alpha)\%$ confidence interval for $\overline{mse}$ when the standard deviation is obtained from a sample of the data. The lower and upper limits of this interval for each case correspond to the contents of columns 1 and 2 in table D.3 and, the third column indicates whether the

set_id	$\overline{mse\_ts}$	std_dev(mse_ts)	mse_gm_ts
1	0.1079	0.0102	0.0896
2	0.0138	0.0004	0.0131
3	0.1504	0.0081	0.1496
4	0.0900	0.0060	0.0857
5	0.0019	0.0007	0.0073
6	0.0029	0	0.0029
7	0.1254	0.0100	0.1159
8	0.0871	0.0044	0.0958
9	0.1226	0.0045	0.1279
10	0.0273	0.0030	0.0175
11	0.0165	0.0004	0.0176
12	0.0617	0.0075	0.0562
13	0.0238	0.0005	0.0240
14	0.0116	0.0044	0.0079
15	0.0221	0.0020	0.0238
16	0.0302	0.0075	0.0369
17	0.0716	0.0097	0.0353
18	0.0940	0.0563	0.0106
19	0.0042	0.0020	0.0061
20	0.0050	0.0003	0.0056
21	0.0012	0.0001	0.0012
22	0.0133	0	0.0133
23	0.0259	0.0067	0.0146
24	0.0222	0.0140	0.0263
25	0.0254	0.0007	0.0256
26	0.0207	0.0084	0.0361
27	0.0295	0.0027	0.0292
28	0.0955	0.0108	0.1025

Table D.2: **data collected on test set for hypothesis testing**

error obtained with the networks initialised with the geometric information, *i.e.* mse_{gm}, fell within its limits. A positive answer provides evidence that the error of the random initialisation and the geometric error have approximately the same kind of value.

Looking at the values in table D.3 it is possible to conclude that the geometric initialisation does in effect produce a better training procedure because the error reached at the end is significantly lower than the error obtained with networks randomly initialised. This important result set the base for developing *GROPING* in the following stages of the research.

set_id	lower_limit	upper_limit	confidence_test	t_test_train	t_test_test
1	0.0103	0.0105	NO	YES	YES
2	0.0035	0.0039	NO	NO	YES
3	0.0445	0.0517	NO	YES	NO
4	0.0140	0.0142	NO	YES	YES
5	0.0070	0.0082	NO	YES	NO
6	0.0039	0.0039	NO	NO	NO
7	0.0166	0.0244	NO	YES	YES
8	0.0177	0.0181	NO	YES	NO
9	0.0082	0.0084	NO	YES	NO
10	0.0083	0.0105	NO	YES	YES
11	0.0084	0.0086	NO	YES	NO
12	0.0623	0.0691	NO	YES	YES
13	0.0049	0.0051	NO	YES	NO
14	0.0068	0.0096	YES	NO	YES
15	0.0062	0.0074	NO	YES	NO
16	0.0038	0.0042	NO	YES	NO
17	0.0468	0.0572	NO	YES	YES
18	0.0539	0.1005	NO	YES	YES
19	0.0066	0.0098	YES	NO	NO
20	0.0058	0.0058	NO	YES	NO
21	0.0057	0.0059	NO	YES	NO
22	0.0040	0.0040	NO	YES	YES
23	0.0136	0.0200	NO	YES	YES
24	0.0121	0.0239	YES	NO	NO
25	0.0123	0.0127	NO	YES	NO
26	0.0243	0.0329	NO	NO	NO
27	0.0112	0.0114	NO	YES	NO
28	0.0136	0.0140	NO	YES	NO

Table D.3: **results of various statistical inference procedures**

Appendix E

Assessing parallelism varying θ

As the initial stage of the network design method defined in this research produces a **maximal** architecture, it seemed reasonable to provide a first reduction step that would involve the elimination of those weight vectors considered to be sufficiently parallel given that it implies redundancy. This evaluation facilitates the subsequent pruning stage of the network development.

As there are no well defined rules indicating what corresponds to sufficiently parallel weight vectors, to provide an appropriate upper limit, it was necessary to generate different network architectures for various of the data sets included in the benchmark and, measure the performance of the learning process in each case. A convenient upper limit on the angle measure would be the one producing the better overall network generaliser in the first instance. Nevertheless, it was also necessary to take into account the number of intermediate architectures that would be subsequently generated because, this influences the possibility of producing a better network given that, a larger sample always gives more guarantee of a better result.

As there is a lot of symmetry in the range of possible angles that can be measured between any two vectors, in this evaluation it was termed sufficient to take into account the angles in the interval $[0, \frac{\pi}{2}]$, *i.e.* all the possible values of the first quadrant in the 2-dimensional circle domain. As the angle value that divides this interval in two equal parts corresponds to $\theta = \frac{\pi}{4}$, the final set of possible angle values used to perform the set of learning processes this specific experiment required was:

$$\{\frac{\pi}{3}, \frac{\pi}{4}, \frac{\pi}{6}\} \tag{E.1}$$

In all the experiments, once the geometric analysis of the training set had been performed and, the initial set of weight vectors had been defined, it was possible to measure the angle that existed between all the possible pairs, grouping them according to their degree of parallelism. The general form of the rule used corresponds to:

$$if\ (angle(w_i, w_j) > \theta)\ thentable\_parallel(w_i, w_j)\ =\ \mathbf{TRUE} \tag{E.2}$$

Finally, one weight vector was selected from each of these groups and, taken to be the **representative** vector of all the others contained in the same group.

The specific evaluation consisted in producing initial maximal architectures changing the value of θ in E to each of the possible instances described in E.1. Subsequently, each of these initial architectures was trained using **rprop**. The performance of the final network produced by the training process was measured through the computation of the error achieved on the training

Square error percentage and Mean error						
angle	num_units	iterations	sep_train	sep_validation	merror_train	merror_validation
CANCER DATA SET						
$\frac{\pi}{6}$	173	52	0.01153	3.10520	0.00285	0.03965
$\frac{\pi}{4}$	76	62	0.00876	2.75144	0.00207	0.02497
$\frac{\pi}{3}$	26	85	0.00930	3.56227	0.00192	0.04075
CARD DATA SET						
$\frac{\pi}{6}$	90	463	0.00462	15.94941	0.00072	0.17061
$\frac{\pi}{4}$	55	150	0.00505	12.78892	0.00147	0.15097
$\frac{\pi}{3}$	20	481	0.00493	18.44226	0.00110	0.19721
DIABETES DATA SET						
$\frac{\pi}{6}$	265	500	3.76000	23.67182	0.13144	0.33961
$\frac{\pi}{4}$	94	338	0.0241	23.0699	0.0075	0.2773
$\frac{\pi}{3}$	29	405	0.02555	27.73760	0.00808	0.31291
THYROID DATA SET						
$\frac{\pi}{6}$	194	149	0.14199	1.02759	0.00682	0.01907
$\frac{\pi}{4}$	79	233	0.00186	0.9985	0.00049	0.01203
$\frac{\pi}{3}$	48	273	0.00200	1.26620	0.00052	0.01524

Table E.1: **Results for initial architectures obtained with different θ**

and validation sets. The results presented in table E.1 were obtained on the classification problems: {CANCER, CARD, DIABETES, THYROID}. As they all reflect similar characteristics, they can be summarised in the following way:

- In all cases the best maximal architecture, at least in terms of the error measure computed with the validation set, was produced with an an angle $\theta = \frac{\pi}{4}$.
- The largest architectures were computed with $\theta = \frac{\pi}{6}$ and the smallest with $\theta = \frac{\pi}{3}$ which, considering the form of rule is as should be because, a lower limit on the value of θ will always result on a larger number of vector pairs being labeled as parallel.
- Selecting the value of the most appropriate θ in terms of the error measure of the validation set, is reasonable considering that: (a) the design method is aimed at building better network generalisers and, (b) there is not much variation between the values of other possible parameters that could have been selected as decision components such as, the total number of iterations needed for training the network.

To provide further proof on the convenience of having selected $\theta = \frac{\pi}{4}$ as relevant limit for the parallelism criteria, table E.2 shows the error measures registered with the best networks and, table ?? contains the size of those same networks, they correspond there is one set of values for each of the classification problems used. In all cases, the results tabulated were obtained with $\theta = \frac{\pi}{4}$. Comparing the network sizes with those included in table E.1 it is clear that if the angle value selected for the rule finally used in **GROPING** had been $\frac{\pi}{3}$ instead of $\frac{\pi}{4}$, the chances of achieving an appropriate generaliser would have been greatly hampered because, the size of the maximal architectures for $\theta = \frac{\pi}{3}$ is, for the majority of the problems, smaller.

Finally, looking at the error measures achieved on the validation sets with the best networks produced with $\theta = \frac{\pi}{4}$ it is evident that these correspond to better generalisers and, although for the other angle values analised only the maximal architectures were built there is no possibility of loosing on a better network generalisers because, with either the larger or smaller angle, the only possible architectures that could have been missed belong to the group of less effective networks.

Square error percentage and Mean error					
problem	iterations	sep_train	sep_validation	merror_train	merror_validation
CANCER	43	0.0038	1.8084	0.0014	0.0221
CARD	101	0.2332	11.2323	0.0189	0.1481
DIABETES	51	6.2648	17.5532	0.1599	0.2821
THYROID	51	0.2137	0.8085	0.0067	0.0136

Table E.2: **Results for best network architecture using GROPING**

Pivot architectures for different data sets		
data set	architecture	num_connections
CANCER	9-15-2	165
CARD	15-27-2	488
DIABETES	8-94-2	1578
THYROID	21-135-3	3278

Table E.3: Best network architectures produced with $\theta = \frac{\pi}{4}$

Appendix F

Benchmark details

The creation of an appropriate multi-layer perceptron network for a particular classification task using **GROPING**, involves an extensive manipulation of the training set resulting, in the generation of various intermediate composite variables. These provide knowledge to the algorithm, on the structure of the space on which the network will be operating. Following the guidelines described in [Prechelt 94], to achieve a proper evaluation of the performance of a learning algorithm working on the data sample of a particular problem, it is necessary to repeat the learning process on different permutations of the same data so as to measure how sensitive the algorithm is to the specific sample it has to work on. To this end, in **Proben1**'s website there are three permutations of the data associated to a particular classification task and, the results published include the performance achieved by their learning algorithm in each of these sets.

The same approach has been pursued in this research but, because so much intermediate information is produced in one execution, in the ensuing description of how it evolves in each of the classification tasks comprising this benchmark, only the most important data subtracted from the first permutation of the dataset is presented. The table that summarises the best results obtained in all cases does include the networks generated in each of the permutations. The intermediate structures that were built when designing a network for the same problem but using the other permutations of the data are all available at **GROPING**'s ftp address:

`ftp://ftp.dai.ed.ac.uk/pub/user/elenapm/.`

Although the construction of a network for a classification task, results in the generation of numerous structures at different stages of the design process, it is worthwhile noting that almost none are required throughout, therefore it is possible to build one, use it and, immediately discard it, this way there is no need to utilise enormous amounts of memory in any one instant of the execution. The intermediate structures generated and their contents are summarised in the ensuing list:

1. a matrix of the transformed data which is comprised only of real-valued attributes.
2. number of inputs, number of outputs of transformed version.
3. the covariance matrix and mean vector of the training set.
4. matrices with normalised versions of the training, validation and test sets calculated using the mean vector and, covariance matrix of the training set.
5. eigenvectors, eigenvalues of the training set's covariance matrix. Projected elements of the training set to the 3-dimension space spanned by the first three eigenvectors of this matrix.

6. delaunay triangulation of the projected training set.
7. initial **maximal** architecture.
8. intermediate networks produced during the pruning phase.

Given that listing all the intermediate elements built for each of the problems is not essential for understanding the algorithm's functioning, only the ones included in the following list will be mentioned here, the complete set related to each problem are available by ftp at **site**:

1. dimensions of the continuous version of the data.
2. covariance matrix and mean vector of the training set.
3. eigenvectors used in projection.
4. initial maximal architecture.
5. values of the performance measures registered at the end of the training process of the intermediate network architectures.

F.1 Cancer

There are 699 patterns in the original dataset. 350 of them were assigned to the training set, 175 were used to build the validation set and, 174 were included in the test set. The resulting sizes correspond to 50%, 25%, 25% of the complete available dataset, as required in the guidelines contained in **Proben1**. All the attributes of the input are continuous-valued. The covariance matrix that was computed from the training set has the following values:

$$Cov(cancer) = \begin{pmatrix} 0.0793 & 0.0554 & 0.0548 & 0.0391 & 0.0325 & 0.0596 & 0.0383 & 0.0461 & 0.0169 \\ 0.0554 & 0.0931 & 0.0822 & 0.0615 & 0.0508 & 0.0755 & 0.0562 & 0.0674 & 0.0240 \\ 0.0548 & 0.0822 & 0.0883 & 0.0580 & 0.0474 & 0.0760 & 0.0533 & 0.0653 & 0.0224 \\ 0.0391 & 0.0615 & 0.0580 & 0.0815 & 0.0379 & 0.0684 & 0.0464 & 0.0526 & 0.0205 \\ 0.0325 & 0.0508 & 0.0474 & 0.0379 & 0.0490 & 0.0464 & 0.0333 & 0.0425 & 0.0182 \\ 0.0596 & 0.0755 & 0.0760 & 0.0684 & 0.0464 & 0.1297 & 0.0594 & 0.0635 & 0.0209 \\ 0.0383 & 0.0562 & 0.0533 & 0.0464 & 0.0333 & 0.0594 & 0.0595 & 0.0496 & 0.0144 \\ 0.0461 & 0.0674 & 0.0653 & 0.0526 & 0.0425 & 0.0635 & 0.0496 & 0.0932 & 0.0224 \\ 0.0169 & 0.0240 & 0.0224 & 0.0205 & 0.0182 & 0.0209 & 0.0144 & 0.0224 & 0.0294 \end{pmatrix}$$

The input to the training process has the same number of dimensions as the original version of the data, therefore for this particular problem the network will be processing a 9-dimension input space. The mean vector of the original data and, the three eigenvectors that were used to project the normalised training set to a 3-dimension space are:

$$mean(cancer) = \begin{pmatrix} 0.4418 \\ 0.3134 \\ 0.3207 \\ 0.2807 \\ 0.3216 \\ 0.3544 \\ 0.3438 \\ 0.2867 \\ 0.1589 \end{pmatrix} \qquad eigen\_vectors(C) = \begin{pmatrix} -0.2119 & 0.2880 & 0.0730 \\ -0.1213 & 0.1670 & -0.0472 \\ 0.1581 & 0.5131 & 0.5147 \\ 0.3131 & -0.1289 & -0.2696 \\ -0.1548 & 0.1768 & 0.3351 \\ -0.0626 & -0.0126 & 0.0402 \\ -0.8845 & -0.1231 & -0.0055 \\ 0.0290 & 0.3623 & 0.2371 \\ 0.0797 & -0.6568 & 0.6962 \end{pmatrix}$$

The geometric analysis produced the weight vectors for a perceptron with one hidden layer of 76 units:

maximal network: 9-76-2

Table F.1 contains the results achieved by the training processes performed on the different intermediate architectures generated by **GROPING** during the pruning phase. The first column contains the network architecture, the second column is the number of iterations required by **rprop** to tune the weights and achieve complete recognition of the training set, columns 3-4-5 contain the value of **SEP** calculated using the weights registered at the end of the training process for each of the sets involved in a training phase, *i.e.* the training, validation and test set; columns 6-7-8 also correspond to information obtained from these sets but, this time the measure calculated is the **ME**.

The results obtained at the time of processing the other datasets comprising this benchmark, will be discussed following the same format as the one used for presenting the results achieved on the cancer problem.

Results on cancer							
architecture	iterations	sep_tr	sep_val	sep_test	me_train	me_val	me_test
9-76-2	62	0.00876	2.75144	2.07059	0.00257	0.03871	0.03198
9-64-2	73	0.00667	2.01956	1.65336	0.00176	0.02746	0.02398
9-54-2	73	0.00483	2.8188	2.90104	0.00155	0.03529	0.03827
9-48-2	70	0.00517	3.27405	2.82909	0.00188	0.04018	0.03823
9-33-2	69	0.01197	3.13170	3.12122	0.00309	0.04147	0.04142
9-23-2	76	0.00371	4.78909	5.21594	0.00134	0.05697	0.06265
9-19-2	77	0.01414	2.96891	2.92377	0.00323	0.03759	0.03592
9-16-2	121	0.00467	3.49898	2.33060	0.00130	0.04531	0.03099
9-15-2	102	0.00994	1.93021	1.93679	0.00241	0.02702	0.02926
9-9-2	296	0.00512	4.79185	4.19679	0.00169	0.05603	0.05035
9-6-2	365	0.00570	4.32876	4.36088	0.00175	0.04803	0.04862

Table F.1: **intermediate results of pruning phase in cancer dataset**

F.2 Card

The original dataset is comprised of 690 patterns and, corresponds to instances of a 2-class classification task. The most interesting characteristic of this problem, has to do with the

results that were achieved with the network design process compared, to the ones obtained in **Proben1**. The improvement is due to:

a.- the strategy pursued in the network development task,

b.- the policies designed to deal with the missing data,

c.- the mechanism defined to manipulate the nominal attributes.

Results on card							
architecture	iterations	sep_tr	sep_val	sep_test	me_train	me_val	me_test
15-47-2	217	0.00536	18.97300	14.77909	0.00113	0.21110	0.16433
15-45-2	230	0.00480	15.60022	16.68119	0.00094	0.18077	0.19301
15-44-2	224	0.00633	15.45970	15.94236	0.00219	0.17595	0.18924
15-37-2	229	0.00660	15.05630	14.06518	0.00212	0.16483	0.16864
15-35-2	149	0.00648	18.76504	19.94952	0.00177	0.21432	0.22946
15-31-2	184	0.00499	16.02935	16.27374	0.00119	0.18015	0.18813
15-27-2	412	0.00579	16.17959	18.52422	0.00115	0.17754	0.19573
15-26-2	319	0.00548	15.39461	16.81764	0.00152	0.16831	0.19695
15-25-2	246	0.00510	15.58955	15.26747	0.00101	0.17429	0.17545

Table F.2: **intermediate results of pruning phase in card dataset**

The transformation of the nominal data to a continuous-valued form was realised with the perl program listed in the following page.

```
encoding:
   0. binary, values: b, a, ?.   (12 missing) positions (0,1,2)
      b=1 0 0=0, a=0 1 0=1, ?=0 0 1=-1
   1. continuous, values 13.75...80.25     (12 missing) positions (3,4)
      treated the missing as having initial value -1 which will be
      replaced at the moment of transforming the data to zero mean and
      unit variance.
   2. continuous, values 0...28 position (5)
      obtained by replacing the data with the value read v scaled to v*28
   3. nominal, values: u, y, l, t, ?.   (6 missing) positions (6,7,8,9,10)
                  1/4 2/4 3/4 1; missing: -1
   4. nominal, values g, p, gg, ?.   (6 missing) positions (11,12,13,14)
                  1/3 2/3 1 -1
   5. values c,    d,   cc,   i,    j,    k,    m,    r,    q,
        1/14, 2/14, 3/14, 4/14, 5/14, 6/14, 7/14, 8/14, 9/14
         w,      x,      e,    aa, ff, ?. (9 missing)
        10/14, 11/14, 12/14, 13/14, 1, -1
   6. nominal, values v, h, bb, j, n, z, dd, ff, o, ?.   (9 missing)
      positions(30,31,32,33,34,35,36,37,38,39)
   7. continuous, values 0...28.5 positions (40)
   8. binary, values t-->1, f-->0. positions (41)
   9. binary, values t-->1, f-->0. positions o(42)
   10. continuous, values 0...67 positions (43)
   11. binary, values t-->1, f-->0. positions (44)
   12. nominal, values g,   p,  s positions (45,46,47)
                  1/3, 2/3, 1
   13. continuous, values 0...2000    (13 missing) positions (48,49)
    transformed the missing digit to -1
   14. continuous, values 0...100000 (must be log-transformed to be useful) positions (50)
-------
  15 inputs
   15. binary, values: +, -
     2:  +=1 0, -=0 1
--------
  2 outputs
```

Encoding for transforming card sample to continuous-valued form

For this problem 345 patterns were assigned to the training set, 173 to the validation set and, 172 comprise the test set. The original space is 15-dimension with 5 nominal attributes, the pre-processing phase in **GROPING** produces a 15-dimension sample of only continuous-valued attributes. There is no missing data at the moment of designing the network because, each of the attributes whose value is not known in some of the patterns available, was subtracted from the original set, subsequently the mean was computed from the available information and, finally all the missing slots in the data took on the relevant mean value. The covariance matrix is a 15-dimension square matrix whose values can be accessed at **GROPING**'s ftpsite. The values of the mean vector used in the normalisation process and, the eigenvectors of the covariance matrix of the normalised data used to generate the 3-dimension projection required for the geometric analysis are:

$$mean(card) = \begin{pmatrix} 0.3176 \\ 31.7658 \\ 4.3589 \\ 0.3112 \\ 0.4149 \\ 0.5017 \\ 0.2463 \\ 2.1888 \\ 0.5304 \\ 0.4551 \\ 2.6464 \\ 0.4667 \\ 0.3903 \\ 228.6308 \\ 0.2559 \end{pmatrix} \quad eigen\_vectors(C) = \begin{pmatrix} -0.1563 & -0.0054 & 0.3921 \\ 0.3969 & 0.4347 & -0.0235 \\ -0.3019 & 0.2863 & -0.0057 \\ -0.2019 & 0.0805 & -0.1402 \\ 0.0215 & -0.1209 & 0.3143 \\ -0.2148 & 0.4811 & 0.2179 \\ -0.0073 & 0.1389 & -0.3375 \\ -0.0169 & -0.0063 & 0.3954 \\ 0.2241 & -0.0605 & 0.3923 \\ 0.4299 & 0.4726 & -0.1242 \\ 0.2798 & 0.1122 & 0.1963 \\ -0.2036 & 0.2738 & 0.2797 \\ -0.5064 & 0.2035 & -0.1980 \\ -0.1661 & 0.1866 & 0.2794 \\ -0.0191 & -0.2575 & 0.0786 \end{pmatrix}$$

The geometric analysis produced the weight vectors for a perceptron with one hidden layer of 27 units:

maximal network: 15-27-2

Table F.2 contains the results achieved by the training processes performed on the different intermediate architectures generated by **GROPING** during the pruning phase, its contents follow the same format as that of the table of results associated to the cancer dataset.

F.3 Diabetes

The 768 patterns of the sample are partitioned in the following way: a subset of 384 patterns for the training set, 192 are assigned to the validation set and, the final 192 comprise the test set.

The covariance matrix computed from the training set that was used for the normalisation phase has the following values:

$$Cov(diabetes) = \begin{pmatrix} 0.0419 & 0.0032 & 0.0038 & -0.0043 & -0.0041 & -0.0008 & -0.0005 & 0.0222 \\ 0.0032 & 0.0248 & 0.0041 & 0.0024 & 0.0071 & 0.0051 & 0.0038 & 0.0058 \\ 0.0038 & 0.0041 & 0.0258 & 0.0064 & 0.0020 & 0.0050 & 0.0018 & 0.0069 \\ -0.0043 & 0.0024 & 0.0064 & 0.0240 & 0.0096 & 0.0082 & 0.0052 & -0.0033 \\ -0.0041 & 0.0071 & 0.0020 & 0.0096 & 0.0189 & 0.0040 & 0.0047 & -0.0038 \\ -0.0008 & 0.0051 & 0.0050 & 0.0082 & 0.0040 & 0.0144 & 0.0029 & 0.0014 \\ -0.0005 & 0.0038 & 0.0018 & 0.0052 & 0.0047 & 0.0029 & 0.0217 & 0.0005 \\ 0.0222 & 0.0058 & 0.0069 & -0.0033 & -0.0038 & 0.0014 & 0.0005 & 0.0326 \end{pmatrix}$$

The values of the mean vector used in the normalisation process and, the eigenvectors of the covariance matrix of the normalised data used to generate the 3-dimension projection required for the geometric analysis are:

$$mean(diabetes) = \begin{pmatrix} 0.2301 \\ 0.6129 \\ 0.5696 \\ 0.2073 \\ 0.0987 \\ 0.4846 \\ 0.1706 \\ 0.1903 \end{pmatrix} \quad eigen\_vectors(C) = \begin{pmatrix} 0.1697 & -0.2633 & 0.1496 \\ 0.0415 & -0.3000 & -0.0202 \\ -0.1445 & 0.0529 & 0.5026 \\ -0.4374 & 0.3034 & 0.6584 \\ 0.0869 & -0.3982 & 0.2990 \\ -0.7690 & -0.3082 & -0.3544 \\ 0.3725 & 0.2348 & -0.0028 \\ -0.1398 & 0.6612 & -0.2761 \end{pmatrix}$$

Results on diabetes							
architecture	iterations	sep_tr	sep_val	sep_test	me_train	me_val	me_test
8-94-2	338	0.0241	23.1456	23.0699	0.0075	0.2716	0.2773
8-79-2	283	0.0238	22.3380	26.2648	0.0082	0.2707	0.3118
8-65-2	304	0.0269	22.1743	26.1677	0.0085	0.2651	0.3032
8-59-2	235	0.0374	21.1668	26.6364	0.0104	0.2546	0.3154
8-53-2	310	0.0317	19.9044	25.3269	0.0093	0.2458	0.3070
8-52-2	294	0.0285	21.0189	23.3261	0.0093	0.2566	0.2782
8-48-2	284	0.0316	21.6262	24.2384	0.0095	0.2604	0.2830
8-47-2	267	0.0478	22.3727	23.2685	0.0122	0.2718	0.2851
8-45-2	276	0.0320	21.5280	23.9538	0.0094	0.2538	0.2833
8-42-2	354	0.0182	22.2592	26.6948	0.0068	0.2578	0.3172
8-39-2	314	0.0322	21.9558	25.9429	0.0095	0.2640	0.3007
8-34-2	362	0.0396	25.8881	26.4978	0.0100	0.2935	0.3029
8-33-2	320	0.0273	20.8643	29.7587	0.0087	0.2472	0.3396
8-29-2	351	0.0384	26.3311	31.2916	0.0101	0.3033	0.3521
8-26-2	338	0.0207	21.9152	26.3073	0.0069	0.2533	0.2967

Table F.3: **intermediate results of pruning phase in diabetes dataset**

The geometric analysis produced the weight vectors for a perceptron with one hidden layer of 94 units:

maximal network: 8-94-2

Table F.3 contains the results achieved by the training processes performed on the different intermediate architectures generated by **GROPING** during the pruning phase.

F.4 Gene

The information about the mean and covariance matrices together with that of the eigenvectors used in the projection of the space for the geometric analysis for this example, can be accessed at the **GROPING** site. They were not included in this work due to the high dimensionality of the data which renders their publication particularly useless also, the additional evaluations realised with this set provided enough material for the discussion at this point.

It was selected to compare the effectiveness of treating the nominal data of a classification problem, according to the specifications outlined in **proben1** against the steps defined in **GROPING**.

Results on gene 120-dimension							
architecture	iterations	sep_tr	sep_val	sep_test	me_train	me_val	me_test
120-96-3	29	0.00614	7.15348	7.31555	0.00284	0.09452	0.09671
120-43-3	31	0.03250	7.80001	8.59946	0.00541	0.10171	0.10950
120-25-3	56	0.00172	9.15788	9.71865	0.00096	0.10553	0.10986
120-15-3	64	0.00525	7.87928	9.11470	0.00212	0.09672	0.10548
120-9-3	137	0.00456	10.73599	9.99343	0.00180	0.11906	0.11266
120-6-3	265	0.00138	12.11847	11.01021	0.00090	0.12797	0.11655

Table F.4: **intermediate results of pruning phase for 120-dimension gene dataset**

Results on gene 60-dimension							
architecture	iterations	sep_tr	sep_val	sep_test	me_train	me_val	me_test
60-216-3	153	0.02356	5.07204	6.22152	0.00085	0.06486	0.07738
60-173-3	47	0.01174	4.82230	5.76796	0.00485	0.07441	0.08345
60-157-3	48	0.01090	5.15989	5.90132	0.00428	0.07686	0.08409
60-52-3	65	0.00234	5.84905	7.08987	0.00165	0.07760	0.09037
60-27-3	86	0.00130	5.98098	8.14680	0.00076	0.07182	0.09441
60-15-3	176	0.00155	7.95636	8.36351	0.00090	0.09050	0.09559

Table F.5: **intermediate results of pruning phase for 60-dimension gene dataset**

The comparison concerns the results that can be achieved with the same learning algorithm used for building a multi-layer perceptron network for both versions of the data. Tables F.4 and F.5, contain the different networks that were obtained with **GROPING**, those included in F.4 were the product of searching for an appropriate network for the gene data formatted according to the rules described in **proben1** which resulted in a 120-dimension sample. The results in F.5, were obtained with the gene sample built according to the rules specified in **GROPING**, which resulted in a 60-dimension sample. The differences in the results, even though the algorithm is working on the same problem, are explained in the following points:

a.- **number of intermediate architectures:** the higher-dimension space required a larger number of intermediate architectures because, it is possible to generate an initial maximal architecture more in accordance to the particularities of the training set given that there are a larger number of dimensions describing it. This also, encourages the generation of training processes achieving a complete recognition of the patterns included in the set used for that task, when operating on the different intermediate network architectures.

b.- **errors on training set:** the errors achieved at the end of the training processes are much smaller in the 120-dimension sample than in those obtained when training the data in the 60-dimension space.

c.- **errors on validation, test sets:** these errors are much smaller in the 60-dimension sample, which is not surprising given that, when working in this space there is less redundancy. This means that the method in charge of building the network works on much more effective information, i.e. one providing a better description of the structure of all the input space, not just of the region from which the training sample has been subtracted.

d.- **number of hidden units:** due to the amount of description incorporated in the higher dimension space it is also possible to reduce the number of conflicting regions that require a higher number of weight vectors for an appropriate partitioning of the space. This results in less units in the hidden laver and, is the reason for achieving complete recognition of the elements of the training set with less hidden units in the 120-dimension sample.

In each case the initial maximal architectures required according to the results of the geometric analysis are:

60-dim maximal net: 60-216-3
120-dim maximal net: 120-96-3

Of the set of differences described above, the most important one concerns the results achieved with the validation and test sets. Processing the 60-dimension sample, it was possible to generate networks which are better at processing data not available during training, therefore they are better **generalisers**. This means that, the steps defined in **GROPING** to transform nominal data, so that it might be used by a multi-layer perceptron network, are more effective than those included in other approaches.

F.5 Glass

This is a small dataset of 214 patterns, 107 of which constitute the elements of the training set, 54 were assigned to the validation set and the last 53 were put into the test set. The input space is a subset of R^9 and, as there is no missing data it was possible to realise the normalisation process without any pre-processing. The covariance matrix of the original training set that was used in the normalisation has the following form:

$$\begin{pmatrix}
0.0000 & -0.0006 & -0.0005 & -0.0005 & -0.0015 & -0.0003 & 0.0036 & -0.0002 & 0.0001 \\
-0.0006 & 0.7015 & -0.4305 & 0.0501 & 0.0307 & -0.1615 & -0.3341 & 0.1746 & -0.0185 \\
-0.0005 & -0.4305 & 2.1997 & -0.3074 & -0.3342 & 0.1269 & -0.8927 & -0.3895 & 0.0031 \\
-0.0005 & 0.0501 & -0.3074 & 0.1907 & 0.0877 & -0.0015 & -0.1123 & 0.0937 & 0.0001 \\
-0.0015 & 0.0307 & -0.3342 & 0.0877 & 0.5234 & 0.0724 & -0.4366 & 0.0655 & -0.0159 \\
-0.0003 & -0.1615 & 0.1269 & -0.0015 & 0.0724 & 0.1268 & -0.1493 & -0.0237 & 0.0035 \\
0.0036 & -0.3341 & -0.8927 & -0.1123 & -0.4366 & -0.1493 & 2.0547 & -0.1273 & 0.0272 \\
-0.0002 & 0.1746 & -0.3895 & 0.0937 & 0.0655 & -0.0237 & -0.1273 & 0.2177 & -0.0050 \\
0.0001 & -0.0185 & 0.0031 & 0.0001 & -0.0159 & 0.0035 & 0.0272 & -0.0050 & 0.0102
\end{pmatrix}$$

The mean vector of the original data and, the three eigenvectors that were used to project the normalised training set to a 3-dimension space are:

$$mean(glass) = \begin{pmatrix} 1.5186 \\ 13.4579 \\ 2.6447 \\ 1.4123 \\ 72.6892 \\ 0.4223 \\ 9.0450 \\ 0.1595 \\ 0.0567 \end{pmatrix} \qquad eigen\_vectors(C) = \begin{pmatrix} 0.2300 & 0.3787 & 0.3147 \\ 0.0169 & 0.0439 & 0.0853 \\ -0.0804 & 0.0119 & 0.0603 \\ -0.0090 & 0.0334 & 0.0468 \\ 0.4777 & 0.4791 & 0.1004 \\ -0.5741 & -0.1364 & -0.1059 \\ -0.6029 & 0.7181 & 0.1136 \\ -0.0537 & -0.1962 & 0.7487 \\ 0.1267 & 0.2261 & -0.5413 \end{pmatrix}$$

The geometric analysis produced the weight vectors for a perceptron with one hidden layer of 39 units:

maximal network: 9-39-6

Table F.6 contains the errors registered on completion of the training process for each of the intermediate architectures. There is a strong difference between the values registered for the error on the training set and those registered for the error of the other two sets, *i.e.* the validation and test set. Nevertheless, as they are very similar to those published in [Prechelt 94] it is possible to conclude that the final network obtained with **GROPING** is better. The reason for the low level of generalisation can be explained by the small size of the training set and the high number of classes, there was not enough information for the algorithm to grasp the structure of the operating space.

Results on glass							
architecture	iterations	sep_tr	sep_val	sep_test	me_train	me_val	me_test
9-32-6	109	0.00789	2.03624	2.22773	0.00789	0.00221	0.02632
9-28-6	334	0.01067	2.51391	2.89014	0.00262	0.02822	0.03341
9-27-6	155	0.01575	3.00347	5.37734	0.00369	0.03671	0.06175
9-25-6	92	0.01746	2.12573	3.33627	0.00399	0.03007	0.04371
9-22-6	163	0.01551	2.70598	3.93015	0.00349	0.03409	0.04934
9-21-6	291	0.01781	2.60600	3.71805	0.00375	0.03298	0.04435

Table F.6: **intermediate results of pruning phase for 9-dimension glass dataset**

F.6 Heart

The two versions of the problem available in the **Proben1** site are stored in the directories labeled **heart** and **heartc**. The one that was brought to the **GROPING** benchmark is the one contained in **heartc**. It corresponds to the 303 examples of the Cleveland dataset. As in the previous cases, the data was partitioned so that 152 patterns were assigned to the training set, 76 to the validation set and, the 75 remaining patterns are included in the test set.

The interesting feature of this example is the number of additional attributes used in **Proben1** to indicate missing information. As can be seen by the mean vector presented in the following page, the particular data that was processed in that benchmark there are many null attributes because most of the data is known. Nevertheless, these additional features greatly affect the training process, given the complex architecture required for achieving the set of results presented in the general tables 6.5, 6.6 in chapter 6. For this reason, the pre-processing phase in **GROPING** approaches the manipulation of missing information in a different manner and, even though the results are not as good, they certainly compare well considering the difference in the network architectures.

The 13 attributes of the UCI data format, one binary, six continuous and six nominal were pre-processed in **GROPING** through the following encoding procedure:

```
encoding:
.  (continuous)  28...77 --> 0...1
    in proben1 attribute 2 indicates missing
.  (sex) 1 (binary) no missing
.  (cp) chest pain type
       1: typical angina,   2: atypical angina
              1/4                  1/2
       3: non-anginal pain  4: asymptomatic
              3/4                  1
  4 continuous; attrmissing -1,
  in proben1 attrib 8 indicates this data is missing
. (trestbps) trestbps: resting blood pressure
  1 (continuous)  80...200;  attrmissing = -1; in proben1 missing attrib 10
. (chol) serum cholestoral in mg/dl
  1 (continuous)  85...603; attrmissing -1; in proben1 missing attrib 12
. (fbs)  (fasting blood sugar > 120 mg/dl)  (1 = true; 0 = false)
  1 (continuous);  -1 attrmissing; in proben1 missing attrib 15
. (restecg)  resting electrocardiographic results
     1/3: normal   2/3: ST-T wave abnormality  1: left ventricular hypertrophy
    attrmissing -1; missing attrib in proben1 19
. (thalach)   maximum heart rate achieved
 1 (continuous)  60...202 --> 0...1
  -1 attrmissing; missing attrib in proben1 21
. (exang)  exercise induced angina (1 = yes; 0 = no)
   prechelt(21-23) code2(15-16)
 -1 attrmissing; proben1 attrib 23
. (oldpeak) ST depression induced by exercise relative to rest
   2 (continuous)  -2.6...6.2-->0...1
  (0 -1) attrmissing  (%g 0) good value; proben1 attrib 25 missing
. (slope)   the slope of the peak exercise ST segment
       1: upsloping   2: flat  3: downsloping
            1/3           2/3          1
  1 (continuous); -1 attrmissing ; 29 is proben1 missing
. (ca)  number of major vessels (0-3) colored by flourosopy
  1 (continuous)  0...3-->0...1; -1 attrmissing; attrib 31
. (thal)  3: normal;  6: fixed defect;  7: reversable defect
                    1/3          2/3                1
  1  continuous; -1 attrmissing; proben1 attrib 35
-------------
proben1 35 inputs
groping 1   inputs
. (output)  (the predicted attribute: 0=healthy, 1-4: increasingly sick)
  two versions:  a) as function approximator 0,1,2,3,4 --> 0.1, 0.6, 0.7, 0.8, 0.9  OR
                 b) as classifier 0 -> 1 0    other -> 0 1
-------------
  1 or 2 ooutputs
```

The mean vector of the 35 dimension data used to train the network in the **Proben1** benchmark has the following values:

$$mean(heartc) = \begin{pmatrix} 0.5334 \\ 0.6645 \\ 0.0987 \\ 0.1776 \\ 0.2961 \\ 0.4276 \\ 0 \\ 0.4407 \\ 0 \\ 0.2985 \\ 0 \\ 0 \\ 0.1579 \\ 0 \\ 0.5658 \\ 0.0132 \\ 0.4211 \\ 0 \end{pmatrix} \qquad cont\_mean(heartc) = \begin{pmatrix} 0.6037 \\ 0 \\ 0 \\ 0.2895 \\ 0 \\ 0.4037 \\ 0 \\ 0.4737 \\ 0.4605 \\ 0.0658 \\ 0 \\ 0.2083 \\ 0.0132 \\ 0.5461 \\ 0.0724 \\ 0.3684 \\ 0.0132 \end{pmatrix}$$

The mean vector of the training set obtained after applying the transformation procedure and, the three eigenvectors that were used to project the normalised data to a 3-dimension space are:

$$mean(heartc) = \begin{pmatrix} 0.5334 \\ 0.6645 \\ 0.7632 \\ 0.4407 \\ 0.2985 \\ 0.1579 \\ 0.6184 \\ 0.6037 \\ 0.2895 \\ 0.4037 \\ 0.5307 \\ 0.2111 \\ 0.6067 \end{pmatrix} \qquad eigen\_vectors(C) = \begin{pmatrix} 0.5149 & 0.1030 & -0.1921 \\ -0.0982 & 0.3231 & -0.0239 \\ 0.0996 & -0.0478 & 0.0450 \\ -0.3620 & -0.3738 & -0.4717 \\ 0.1572 & 0.3929 & 0.3866 \\ 0.0421 & 0.3575 & -0.1576 \\ 0.1683 & 0.1942 & -0.4538 \\ -0.4083 & 0.3466 & -0.3283 \\ 0.0226 & 0.3585 & -0.2153 \\ -0.3077 & 0.3558 & 0.0511 \\ 0.5134 & -0.0513 & -0.3574 \\ -0.0280 & -0.2097 & 0.0434 \\ 0.0557 & 0.0353 & 0.2672 \end{pmatrix}$$

The geometric analysis of the continuous-valued attributes produced the weight vectors for a perceptron with one hidden layer of 76 units:

maximal network: 13-76-2

Table F.7 contains the results achieved by the training processes performed on the different intermediate architectures generated by **GROPING** during the pruning phase. It is clear from those results that achieving a good generalisation level with this data is a extremely difficult task.

F.7 Mushroom

The original source for this dataset is the UCI machine learning database: mushroom but, the set used in this benchmark was subtracted from the **Proben1** site. The resulting network

Results on heart							
architecture	iterations	sep_tr	sep_val	sep_test	me_train	me_val	me_test
13-76-2	45	0.02940	12.05531	20.92150	0.00953	0.16653	0.25049
13-47-2	46	0.03438	14.38860	19.58661	0.00962	0.18851	0.24779
13-36-2	65	0.01782	16.92880	22.20152	0.00651	0.21665	0.25813
13-29-2	53	0.01741	18.53175	20.43714	0.00614	0.21873	0.23744
13-26-2	56	0.01822	14.50076	22.62753	0.00664	0.17858	0.26367
13-21-2	58	0.01896	13.61210	21.06830	0.00634	0.18141	0.25514
13-16-2	84	0.02756	16.66950	21.51554	0.00781	0.19902	0.26758
13-9-2	169	0.01513	16.61367	21.83818	0.00471	0.19240	0.25131

Table F.7: **intermediate results of pruning phase in the heart dataset**

architecture was very simple and, as the developers of the **Proben1** benchmark only processed one permutation of the data available on the problem, the same approach was followed in this study.

The original dataset is comprised of 22 nominal attributes which, following the rules defined in **Proben1** were encoded to a 123-dimension binary space and, following the guidelines specified in **GROPING** resulted in a subspace of R^{22}, *i.e.* it is conformed of 22 continuous-valued attributes. The dataset is comprised of 8124 patterns which means that, it was possible to build the training set with 4062 elements, the validation set using 2031 and, the test set using the remaining 2031 patterns.

One third of the data has unknown value for the 11th attribute, a factb that was indicated with the '?' symbol. It was replaced by zero in **GROPING**'s encoding mechanism. These null values during the normalisation of the data were replaced by the mean of the available information. One thing that was noted is that there is no variance on the 16th attribute which meant an almost null dimension for the normalised data but as it did not create major difficulties during the geometric analysis it was not processed in any special way. The encoding used for transforming the data from its original form, of nominal attributes to real-valued ones, is depicted in the following page.

```
 1. cap-shape:                bell=b,conical=c,convex=x,flat=f,
                                knobbed=k,sunken=s
   associated set: {b=0.1667,c=0.333,x=0.5,f=0.667,k=0.833,s=1.00}
 2. cap-surface:              fibrous=f,grooves=g,scaly=y,smooth=s
   associated set: {f=0.25,g=0.5,y=0.75,s=1.00}
 3. cap-color:                brown=n,buff=b,cinnamon=c,gray=g,green=r,
                                pink=p,purple=u,red=e,white=w,yellow=y
   associated set: {n=0.1, b=0.2, c=0.3, g=0.4, r=0.5, p=0.6, u=0.7, e=0.8, w=0.9, y=1.0}
 4. bruises?:                 bruises=t,no=f
   associated set: {t=0,f=1}
 5. odor:                     almond=a,anise=l,creosote=c,fishy=y,foul=f,
                                musty=m,none=n,pungent=p,spicy=s
   associated set: {a=0.111, l=0.222, c=0.333, y=0.444,
                   {f=0.556, m=0.667, n=0.778, p=0.889, s=1.00}
 6. gill-attachment:          attached=a,descending=d,free=f,notched=n
   associated set: {a=0.25, d=0.5, f=0.75, n=1.00}
 7. gill-spacing:             close=c,crowded=w,distant=d
   associated set: {c=0.333, w=0.667, d=1.00}
 8. gill-size:                broad=b,narrow=n
   associated set: {b=0, n=1}
 9. gill-color:               black=k,brown=n,buff=b,chocolate=h,gray=g,
                                green=r,orange=o,pink=p,purple=u,red=e,
                                white=w,yellow=y
   associated set: {k=0.083, n=0.1667, b=0.25, h=0.333, g=0.4167, r=0.5,
                    o=0.5833, p=0.667, u=0.75, e=0.833, w=0.916, y=1.00}
10. stalk-shape:              enlarging=e,tapering=t
  associated set:  {e=0, t=1}
11. stalk-root:               bulbous=b,club=c,cup=u,equal=e,
                                rhizomorphs=z,rooted=r,missing=?
  missing=replace missing value with the mean
          over the existing values; at first instant when creating
          the encoding simply replace the missing character with the
          value zero
12. stalk-surface-above-ring: ibrous=f,scaly=y,silky=k,smooth=s
  associated set: {f=0.25, y=0.5, k=0.75, s=1.00}
13. stalk-surface-below-ring: ibrous=f,scaly=y,silky=k,smooth=s
  associated set: {f=0.25, y=0.5, k=0.75, s=1.00}
14. stalk-color-above-ring:   brown=n,buff=b,cinnamon=c,gray=g,orange=o,
                                pink=p,red=e,white=w,yellow=y
  associated set: {n=0.111, b=0.222, c=0.333, g=0.444, o=0.556, p=0.667,
                   e=0.779, w=0.889, y=1.00}
15. stalk-color-below-ring:   brown=n,buff=b,cinnamon=c,gray=g,orange=o,
                                pink=p,red=e,white=w,yellow=y
  associated set: {n=0.111, b=0.222, c=0.333, g=0.444, o=0.556, p=0.667,
                   e=0.779, w=0.889, y=1.00}
16. veil-type:                partial=p,universal=u
  associated set: {p=0, u=1}
17. veil-color:               brown=n,orange=o,white=w,yellow=y
  associated set: {b=0.25, o=0.5, w=0.75, y=1.00}
18. ring-number:              none=n,one=o,two=t
  associated set: {n=0.333, o=0.667, t=1.00}
19. ring-type:                cobwebby=c,evanescent=e,flaring=f,large=l,
                                none=n,pendant=p,sheathing=s,zone=z
  associated set: {c=0.125, e=0.25, f=0.375, l=0.5, n=0.625, p=0.7500, s=0.875, z=1.00}
20. spore-print-color:        black=k,brown=n,buff=b,chocolate=h,green=r,
                                orange=o,purple=u,white=w,yellow=y
  associated set: {k=0.111, n=0.222, b=0.333, h=0.444, r=0.556, o=0.667,
                   u=0.779, w=0.889, y=1.00}
21. population:               abundant=a,clustered=c,numerous=n,
                                scattered=s,several=v,solitary=y
  associated set: {a=0.167, c=0.333, n=0.5, s=0.667, v=0.833, y=1.00}
22. habitat:                  grasses=g,leaves=l,meadows=m,paths=p,
                                urban=u,waste=w,woods=d
  associated set: {g=0.143, l=0.286, m=0.429, p=0.571, u=0.714, w=0.857, d=1.00}
```

Given the dimensionality of the real-valued input it did not seem useful to publish the covari-

ance matrix used in the normalisation process but, to give an idea of the overall structure of the transformed data the mean of the original real-valued version of the training set and, the 3 eigenvectors that span the space unto which the data was projected to perform the geometric analysis are shown in the following vectors:

$$mean(mushroom) = \begin{pmatrix} 0.5826 \\ 0.6864 \\ 0.5348 \\ 0.5812 \\ 0.6412 \\ 0.7362 \\ 0.3859 \\ 0.3038 \\ 0.4801 \\ 0.5702 \\ 0.2157 \\ 0.8756 \\ 0.8572 \\ 0.7177 \\ 0.7126 \\ 0 \\ 0.7405 \\ 0.6902 \\ 0.5374 \\ 0.4515 \\ 0.7739 \\ 0.5991 \end{pmatrix} \quad eigen\_vectors(C) = \begin{pmatrix} 0 & 0 & 0 \\ -0.2917 & 0.1339 & -0.1767 \\ 0.3901 & -0.1446 & 0.4227 \\ -0.1827 & -0.1545 & 0.2919 \\ -0.2381 & -0.2450 & -0.3240 \\ -0.1373 & -0.1997 & -0.0370 \\ -0.0683 & 0.1863 & -0.2263 \\ 0.4473 & -0.0401 & -0.1780 \\ -0.0249 & -0.0223 & 0.1556 \\ -0.2114 & -0.2144 & -0.0030 \\ 0.3302 & 0.0002 & -0.2246 \\ -0.0324 & 0.1532 & 0.1241 \\ 0.1087 & -0.2084 & 0.1643 \\ -0.0529 & -0.0628 & 0.1060 \\ -0.1906 & 0.0440 & -0.0895 \\ 0.1365 & 0.0839 & 0.1864 \\ -0.4000 & 0.0287 & 0.2689 \\ -0.0829 & -0.5050 & -0.1477 \\ -0.0638 & 0.0534 & 0.0570 \\ -0.1303 & -0.2211 & -0.1136 \\ 0.1579 & -0.5702 & -0.1337 \\ -0.1059 & -0.2051 & 0.4654 \end{pmatrix}$$

The geometric analysis of the projected training points produced the following maximal architecture for this case:

maximal network: 22-165-2

Table F.8 contains the errors obtained at the end of the different training processes realised on the various intermediate network architectures that were produced by **GROPING** to reflect the mapping underlying this dataset. The results clearly indicate that it is a very easy problem, *i.e.* particularly appropriate to the underlying philosophy of the multi-layer perceptron model.

F.8 Satimage

The mean vector of the continuous-valued training set and, the eigenvectors spanning the space on which the data is projected into 3-dimension have the following values:

Results on mushroom							
architecture	iterations	sep_tr	sep_val	sep_test	me_train	me_val	me_test
22-165-2	31	0.00173	0.14255	0.08974	0.00112	0.00550	0.00480
22-84-2	28	0.00200	1.72083	1.51221	0.00140	0.02689	0.02364
22-59-2	31	0.00193	0.02264	0.02104	0.00108	0.00351	0.00335
22-48-2	25	0.00206	0.32319	0.25427	0.00106	0.00865	0.00779
22-43-2	29	0.00064	0.25080	0.20434	0.00064	0.25080	0.20434
22-38-2	25	0.00083	0.00471	0.00294	0.00056	0.00122	0.00101
22-36-2	27	0.00109	1.12088	1.17224	0.00079	0.01702	0.01731
22-35-2	26	0.00230	0.05346	0.05285	0.00127	0.00517	0.00502
22-20-2	27	0.00055	1.56228	1.49141	0.00036	0.02033	0.01914
22-15-2	31	0.00123	0.39053	0.28111	0.00075	0.00886	0.00729
22-11-2	60	0.00102	0.18376	0.07724	0.00056	0.00593	0.00371
22-6-2	74	0.00233	1.54594	1.36306	0.00113	0.02013	0.01817

Table F.8: **intermediate results of pruning phase in mushroom dataset**

$$mean(satimage) = \begin{pmatrix} 70.8648 \\ 82.4969 \\ 98.3984 \\ 81.8216 \\ 70.5183 \\ 82.0087 \\ 98.2085 \\ 81.6976 \\ 70.1852 \\ 81.5553 \\ 97.9490 \\ 81.5966 \\ 70.6992 \\ 82.3863 \\ 98.4553 \\ 81.8266 \\ 70.3707 \\ 81.9310 \\ 98.2088 \\ 81.7371 \\ 70.0963 \\ 81.4913 \\ 97.8605 \\ 81.5715 \\ 70.4686 \\ 82.2582 \\ 98.2483 \\ 81.7769 \\ 70.1796 \\ 81.9105 \\ 98.0569 \\ 81.6790 \\ 69.8757 \\ 81.4667 \\ 97.8369 \\ 81.5466 \end{pmatrix} \quad eigen\_vectors(C) = \begin{pmatrix} 0.0047 & 0.0115 & 0.0194 \\ -0.0066 & -0.0334 & 0.0237 \\ -0.0462 & -0.0076 & -0.0181 \\ -0.0098 & -0.1199 & -0.0621 \\ -0.0036 & -0.0777 & -0.0004 \\ 0.0064 & 0.0624 & 0.0634 \\ -0.0235 & -0.0141 & -0.0393 \\ 0.0027 & -0.0680 & 0.1440 \\ -0.0380 & -0.0130 & -0.1158 \\ 0.0092 & 0.0438 & 0.0275 \\ -0.0148 & -0.0377 & -0.0548 \\ 0.0163 & -0.1121 & -0.0025 \\ -0.0121 & -0.0214 & -0.0717 \\ -0.0231 & 0.1321 & 0.0185 \\ 0.0060 & 0.0589 & 0.0723 \\ -0.0037 & 0.0261 & -0.0070 \\ 0.0035 & 0.0672 & 0.0935 \\ 0.0085 & -0.0173 & 0.0842 \\ -0.0066 & -0.1492 & 0.0767 \\ -0.0790 & 0.0642 & 0.0497 \\ -0.0135 & 0.1638 & 0.0384 \\ 0.0175 & -0.2040 & -0.0165 \\ 0.0110 & -0.1292 & -0.0642 \\ 0.0334 & -0.1975 & -0.1149 \\ -0.0265 & -0.0481 & -0.0492 \\ 0.0404 & 0.0515 & 0.0967 \\ 0.0240 & 0.0404 & 0.2074 \\ -0.0190 & -0.4796 & 0.2415 \\ 0.0246 & 0.3354 & -0.0255 \\ 0.2244 & 0.1378 & 0.8028 \\ -0.1543 & 0.0130 & -0.0727 \\ -0.2409 & -0.4969 & -0.0212 \\ 0.0529 & 0.3257 & -0.2782 \\ -0.0178 & -0.1408 & 0.0801 \\ -0.9201 & 0.1869 & 0.2043 \\ -0.0223 & -0.0007 & -0.0890 \end{pmatrix}$$

Results on satimage							
architecture	iterations	sep_tr	sep_val	sep_test	me_train	me_val	me_test
36-543-6	451	0.14049	5.25130	5.33440	0.01238	0.07138	0.07125
36-257-6	451	0.33209	5.19432	5.24138	0.01877	0.07347	0.07346
36-180-6	451	0.01638	4.87056	4.98537	0.00358	0.06184	0.06164
36-115-6	179	0.00664	5.77055	6.00753	0.00239	0.07134	0.07309
36-90-6	158	0.00519	5.82593	6.42386	0.00194	0.06901	0.07442
36-76-6	169	0.00687	5.99362	6.85241	0.00225	0.07099	0.07926
36-40-6	322	0.00265	6.45666	7.26779	0.00109	0.07197	0.08035

Table F.9: **intermediate results of pruning phase for 36-dimension satimage dataset**

The output of the geometric analysis realised on the projected satimage data corresponds to the following multi-layer perceptron:

maximal network: 36-543-6

Table F.9 contains the different error measures computed in this benchmark. These were registered at the end of the training processes of the intermediate network architectures generated during the pruning phase.

F.9 Soybean

In this dataset there are 683 examples Of these, 342 were assigned to the training set, 171 to the validation set and, the remaining 170 constitute the test set.

There is a high number of possible outputs which could easily complicate the generation of an appropriate network architecture but, because there is a lot of redundancy in the data available, the task was not as hard as was originally expected. There is a fair mix of nominal, binary and continuous data also, a significant degree of missing values in certain attributes. The transformation process to generate an input comprised only of continuous-valued attributes was realised according to the rules described in the following encoding procedure. It is necessary to mention that in this encoding missing information is denoted with the value -1 which will be subsequently replaced by the attribute's mean computed from the available data at the moment of the normalisation procedure. The transformation resulted in an input space which is a subset of R^{35}.

```
Attribute encoding:
. date:       april,may,june,july,august,september,october,?. [miss:1]
               0...6 -> 0...1, ? -> -1
. plant-stand: normal,lt-normal,?.  [miss: 36] -->      (0      1       -1)
. precip:      lt-norm,norm,gt-norm,?. [miss: 38]
                  1/3  2/3   3/3   -1
. temp:        lt-norm,norm,gt-norm,?. [miss: 30]
                 1/3   2/3     1   -1
. hail:        yes,no,?.  [miss: 121] -->  (1   0 -1)
. crop-hist:   diff-lst-year,same-lst-yr,same-lst-two-yrs,
                 1/4           1/2             3/4
                same-lst-sev-yrs, ?. [miss: 16] -->   (1       -1)
. area-damaged: scattered,low-areas,upper-areas,whole-field,?.  [miss: 1]
                   1/4       1/2       3/4          1      -1
. severity:        minor,pot-severe,severe,?.  [miss: 1]
                   1/3     2/3         1     -1
. seed-tmt:        none,fungicide,other,?.  [miss: 121]
                   1/3     2/3     3/3  -1
  10. germination:     90-100%,80-89%,lt-80%,?.  [miss: 112]
                   1/3     2/3     1     -1
  11. plant-growth:    norm,abnorm,?.  [miss: 16] -->  (0      1  -1)
  12. leaves:          norm,abnorm.  [miss: 0] -->      (0       1)
  13. leafspots-halo:  absent,yellow-halos,no-yellow-halos,?.  [miss: 84]
                   1/3     2/3           1          -1
  14. leafspots-marg:  w-s-marg,no-w-s-marg,dna,?.  [miss: 84]
                   1/3      2/3          1   -1
  15. leafspot-size:   lt-1/8,gt-1/8,dna,?.  [miss: 84]
                   1/3     2/3     1   -1
  16. leaf-shread:     absent,present,?.  [miss: 100]
                   0       1     -1
  17. leaf-malf:       absent,present,?.  [miss: 84]
                    0       1    -1
  18. leaf-mild:       absent,upper-surf,lower-surf,?.  [miss: 108]
                  1/3     2/3         1        -1
  19. stem:            norm,abnorm,?.  [miss: 16]
                   0   1    -1
  20. lodging:         yes,no,?.  [miss: 121]
                  1   0   -1
  21. stem-cankers:    absent,below-soil,above-soil,above-sec-nde,?.  [miss: 38]
                  1/4    1/2        3/4         1            -1
  22. canker-lesion:   dna,brown,dk-brown-blk,tan,?.  [miss: 38]
                  1/4  1/2  3/4            1   -1
  23. fruiting-bodies: absent,present,?.  [miss: 106]
                  0      1       -1
  24. external decay:  absent,firm-and-dry,watery,?.  [miss: 38]
                   1/3    2/3           1      -1
  25. mycelium:        absent,present,?.  [miss: 38]
                  0      1       -1
  26. int-discolor:    none,brown,black,?.  [miss: 38]
                  1/3  2/3   1      -1
  27. sclerotia:       absent,present,?.  [miss: 38]
                   0      1       -1
  28. fruit-pods:      norm,diseased,few-present,dna,?.  [miss: 84]
                  1/4  1/2      3/4          1   -1
  29. fruit spots:     absent,colored,brown-w/blk-specks,distort,dna,?. [miss: 106]
                   1/5    2/5     3/5                4/5    1     -1
  30. seed             norm,abnorm,?.  [miss: 92] ->  (0      1       -1)
  31. mold-growth:     absent,present,?.  [miss: 92] ->  (0      1       -1)
  32. seed-discolor:   absent,present,?.  [miss: 106] ->  (0      1       -1)
  33. seed-size:       norm,lt-norm,?.  [miss: 92] ->  (0      1       -1)
  34. shriveling:      absent,present,?.  [miss: 106] ->  (0      1       -1)
  35. roots:           norm,rotted,galls-cysts,?.  [miss: 31] -> (1/3  2/3    1
-1)
```

The mean vector of the continuous-valued training set and, the eigenvectors spanning the space

on which the data is projected into 3-dimension have the following values:

$$mean(soybean) = \begin{pmatrix} 0.5814 \\ 0.4581 \\ 0.8520 \\ 0.7025 \\ 0.2133 \\ 0.7097 \\ 0.6488 \\ 0.6100 \\ 0.5122 \\ 0.6842 \\ 0.4064 \\ 0.8772 \\ 0.6730 \\ 0.6519 \\ 0.7928 \\ 0.1519 \\ 0.0442 \\ 0.3654 \\ 0.5894 \\ 0.0800 \\ 0.5023 \\ 0.5045 \\ 0.1830 \\ 0.4199 \\ 0.0181 \\ 0.3938 \\ 0.0604 \\ 0.4059 \\ 0.4817 \\ 0.1635 \\ 0.0994 \\ 0.0915 \\ 0.0865 \\ 0.0523 \\ 0.3811 \end{pmatrix} \quad eigen\_vectors(C) = \begin{pmatrix} 0.2853 & -0.1649 & -0.0446 \\ 0.0164 & 0.0724 & -0.2659 \\ 0.2907 & -0.1742 & -0.0776 \\ 0.2959 & -0.2112 & 0.2473 \\ 0.2199 & 0.2004 & 0.0679 \\ -0.1114 & -0.0522 & -0.0592 \\ -0.2630 & 0.1422 & 0.3052 \\ -0.1051 & -0.0719 & 0.0677 \\ 0.0228 & -0.1176 & -0.0085 \\ 0.1682 & -0.2933 & 0.1932 \\ -0.1070 & -0.1047 & 0.1131 \\ 0.0981 & 0.0741 & 0.0559 \\ -0.0246 & 0.1059 & 0.2460 \\ -0.0127 & -0.2221 & 0.0066 \\ 0.0228 & 0.1204 & -0.0181 \\ 0.0082 & -0.1728 & 0.0539 \\ 0.2048 & 0.3047 & 0.0026 \\ 0.1658 & -0.0689 & -0.2474 \\ 0.0969 & 0.3409 & 0.0639 \\ -0.1155 & -0.2798 & 0.1620 \\ -0.2667 & 0.0708 & 0.3236 \\ 0.0466 & 0.0681 & -0.0543 \\ -0.1854 & 0.0424 & -0.1122 \\ 0.0885 & -0.0105 & 0.2248 \\ 0.2719 & -0.0049 & -0.0916 \\ 0.2270 & 0.0363 & 0.2260 \\ 0.1570 & -0.3170 & 0.2719 \\ 0.1878 & 0.0790 & 0.0325 \\ 0.2760 & 0.0277 & -0.1978 \\ 0.0469 & 0.0471 & -0.1348 \\ 0.0491 & 0.2300 & 0.0660 \\ 0.1059 & 0.1007 & 0.3170 \\ 0.0624 & 0.0513 & -0.0791 \\ 0.2409 & 0.3350 & 0.2635 \\ -0.0841 & 0.1022 & -0.0507 \end{pmatrix}$$

The output of the geometric analysis realised on the projected soybean data corresponds to the following multi-layer perceptron:

maximal network: 35-28-19

Table F.10 contains the different error measures computed in this benchmark. These were registered at the end of the training processes of the intermediate network architectures generated during the pruning phase.

F.10 Thyroid

This example consists of a set of vectors comprised of 21 real-valued attributes, each representing a point that is classified into one of 3 possible classes. The complete sample has 7200

Results on soybean							
architecture	iterations	sep_tr	sep_val	sep_test	me_train	me_val	me_test
35-51-19	33	0.00055	0.54222	1.12183	0.00014	0.00678	0.01244
35-44-19	28	0.00051	0.62995	1.05059	0.00015	0.00763	0.01267
35-35-19	26	0.00101	0.64416	0.90787	0.00029	0.00770	0.01089
35-32-19	33	0.00055	0.54222	1.12183	0.00014	0.00678	0.01244
35-27-19	32	0.00100	0.80849	0.92507	0.00897	0.01097	0.00997
35-22-19	50	0.00029	0.95937	1.41129	0.00007	0.01012	0.01483
35-28-19	70	0.00076	1.76229	1.36577	0.00020	0.01855	0.01567
35-17-19	60	0.00062	1.73967	1.86405	0.00017	0.01857	0.02002

Table F.10: **intermediate results of pruning phase for 35-dimension soybean dataset**

pairs of inputs and expected outputs thus, the training set was built using 3600 of them, the validation set has 1800 and, the remaining 1800 were assigned to the test set.

There was no need to define an Encoding procedure as there are no nominal attributes and, there is no information concerning which of them represent simple indicators of missing information, therefore the complete dataset was processed as one in which all the information is available. The pre-processing was limited to generating the normalised version of the sample using the mean and covariance of the training set. The mean vector used and, the three eigenvectors spanning the space on which the data was projected to 3-dimensions are:

$$mean(thyroid) = \begin{pmatrix} 0.5211 \\ 0.3057 \\ 0.1355 \\ 0.0167 \\ 0.0133 \\ 0.0392 \\ 0.0119 \\ 0.0170 \\ 0.0186 \\ 0.0636 \\ 0.0665 \\ 0.0117 \\ 0.0095 \\ 0.0231 \\ 0.0003 \\ 0.0517 \\ 0.0048 \\ 0.0200 \\ 0.1089 \\ 0.0976 \\ 0.1128 \end{pmatrix} \quad eigen\_vectors(C) = \begin{pmatrix} 0.0217 & 0.2363 & 0.2234 \\ -0.0450 & -0.0008 & 0.0629 \\ 0.1859 & -0.4690 & 0.1366 \\ 0.0190 & 0.2827 & 0.0428 \\ 0.2474 & -0.0872 & 0.1296 \\ 0.1630 & 0.1997 & 0.4292 \\ 0.0488 & 0.1521 & -0.1634 \\ 0.1285 & 0.4828 & 0.2058 \\ -0.0901 & -0.1894 & 0.5320 \\ -0.0214 & 0.0992 & -0.4051 \\ -0.5017 & 0.0415 & 0.1144 \\ 0.6184 & -0.0420 & -0.0777 \\ 0.2598 & -0.1639 & 0.0975 \\ 0.1396 & 0.2062 & -0.2329 \\ 0.2805 & -0.0323 & 0.0490 \\ 0.0164 & -0.0592 & -0.1323 \\ -0.0678 & -0.0137 & 0.1454 \\ -0.1396 & 0.0247 & -0.0380 \\ -0.0943 & -0.4264 & -0.0208 \\ -0.1196 & -0.1135 & 0.0587 \\ -0.0042 & -0.1481 & -0.2815 \end{pmatrix}$$

The geometric analysis produced the following initial maximal architecture:

maximal network: 21-441-3

Table F.11 contains the errors computed with the different intermediate architectures built

Results on thyroid							
architecture	iterations	sep_tr	sep_val	sep_test	me_train	me_val	me_test
21-79-3	233	0.00186	1.03465	0.99850	0.00049	0.01265	0.01203
21-60-3	282	0.00168	1.00974	0.97583	0.00047	0.01267	0.01227
21-37-3	253	0.00168	0.97152	1.00025	0.00045	0.01142	0.01158
21-35-3	380	0.00117	1.06187	1.09569	0.00032	0.01226	0.01192

Table F.11: **intermediate results of pruning phase for 21-dimension thyroid dataset**

during the pruning phase.

F.11 Vowel

This example consists of a set of vectors comprised of 10 real-valued attributes, each representing a point that is classified into one of 11 possible classes. The complete sample has 990 pairs of inputs and expected outputs. The training set was built using 528 of them, the validation set has 231 and, the remaining 231 were assigned to the test set.

There was no need to define an Encoding procedure as there are no nominal attributes and, there is no information concerning which of them represent simple indicators of missing information, therefore the complete dataset was processed as one in which all the information is available. The pre-processing was limited to generating the normalised version of the sample using the mean and covariance of the training set and to the transformation of the phonemes associated to each of the possible output classes to the binary encoding described in section 6.3.2. The mean vector used and, the three eigenvectors spanning the space on which the data was projected to 3-dimensions are:

$$mean(vowel) = \begin{pmatrix} -3.1667 \\ 1.7353 \\ -0.4480 \\ 0.5250 \\ -0.3893 \\ 0.5850 \\ 0.0175 \\ 0.4174 \\ -0.2681 \\ -0.0846 \end{pmatrix} \quad eigen\_vectors(C) = \begin{pmatrix} -0.1529 & 0.2996 & -0.3400 \\ -0.5313 & -0.0823 & -0.3794 \\ 0.1327 & 0.1312 & 0.1735 \\ -0.3601 & 0.2548 & 0.4656 \\ 0.0417 & 0.3489 & -0.0713 \\ 0.1730 & -0.1361 & -0.0105 \\ -0.3132 & 0.1049 & -0.4854 \\ 0.5306 & -0.2942 & -0.4148 \\ -0.3686 & -0.7444 & 0.0978 \\ 0.0000 & -0.1712 & 0.2669 \end{pmatrix}$$

Results on vowel							
architecture	iterations	sep_tr	sep_val	sep_test	me_train	me_val	me_test
10-62-11	117	0.00125	7.34236	8.39973	0.00054	0.08270	0.09444
10-49-11	138	0.00243	7.18118	7.47880	0.00091	0.07961	0.08234
10-42-11	147	0.00175	7.11968	7.88264	0.00074	0.07760	0.08704
10-38-11	161	0.00149	7.78821	8.39018	0.00058	0.08701	0.09249
10-24-11	415	0.00356	8.98707	9.60646	0.00112	0.09824	0.10544

Table F.12: **intermediate results of pruning phase for 10-dimension vowel dataset**

The geometric analysis produced the following initial maximal architecture:

maximal network: 10-62-11

Table F.11 contains the errors computed with the different intermediate architectures built during the pruning phase.