6-1-1994

PARALLEL, PROBABILISTIC, SELF-ORGANIZING, HIERARCHICAL NEURAL NETWORKS

Faramarz Valafar
Purdue University School of Electrical Engineering

Okan K. Ersoy
Purdue University School of Electrical Engineering

Parallel, Probabilistic, Self-organizing, Hierarchical Neural Networks

Faramarz Valafar
Okan Ersoy

TR-EE 94-23
June 1994

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907-1285

PARALLEL, PROBABILISTIC, SELF-ORGANIZING, HIERARCHICAL NEURAL NETWORKS*

Faramarz Valafar
Box 482
School of Electrical Engineering
Purdue University
W. Lafayette, IN 47907
faramarz@transform.ecn.purdue.edu

Okan K. Ersoy
School of Electrical Engineering
Purdue University
W. Lafayette, IN 47907
ersoy@amalthea.ecn.purdue.edu

• The Purdue University MASPAR MP-1 research is supported in part by NSF Parallel Infrastructure Grant #CDA-9015696.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Valafar, Faramarz. Ph.D., Purdue University, August 1993. PARALLEL PROBABILISTIC SELF-ORGANIZING HIERARCHICAL NEURAL NETWORKS. Major Professor: Okan K. Ersoy.

A new neural network architecture called the Parallel Probabilistic Self-organizing Hierarchical Neural Network (PPSHNN) is introduced. The PPSHNN is designed to solve complex classification problems, by dividing the input vector space into regions, and by performing classification on those regions. It consists of several modules which operate in a hierarchically during learning and in parallel during testing. Each module has the task of classification for a region of the input information space as well as the task of participating in the formation of these regions through post- and pre-rejection schemes. The decomposition into regions is performed in a manner that makes classification easier on each of the regions. The post-rejector submodule performs a bitwise statistical analysis and detection of hard to classify vectors. The pre-rejector module accepts only those classes for which the module is trained and rejects others.

The PNS module is developed as a variation of the PPSHNN module. If delta rule networks are used to build the submodules of PNS, then it uses piecewise linear boundaries to divide the problem space into regions. The PNS module has a high classification accuracy while it remains relatively inexpensive. The submodules of PNS are fractile in nature, meaning that each such unit may itself consist of a number of PNS

modules. The PNS module is discussed as the building block for the synthesis of PPSHNN.

The SIMD version of PPSHNN is implemented on MASPAR with 16k processors. On all the experiments performed, this network has outperformed the previously used networks in terms of accuracy of classification and speed.

CHAPTER 1

INTRODUCTION

This thesis involves a neural network approach to the problem of classification. Specifically, classification in complex environments. The task of classification is one of the very basic abilities of human beings or all living beings. Every living being at some level has to make the determination of its environment. This determination is made instinctively and subconsciously or intelligently at a conscious level. At any level it goes hand in hand with the classification of the entities of the environment. Despite the long and intensive research in this area, Nature's techniques of classification still elude us.

The most basic and essential determination of the environment for human beings is the sense of locality or the sense of where one is at any given time. This determination is made based on the processing of certain sensory inputs such as images, sounds and odor. These pieces of information are cross- correlated and the higher reasoning region of the brain makes the determination of the where abouts. The processing of information requires classification. For instance, the images that the eyes send to the brain are noisy, distorted, and sometimes not observed previously. Despite such problems, brain usually classifies things correctly, for example, even if the person has never seen the image before.

The ability to classify a certain object correctly, without having to have seen it before, is called generalization. For example, if a person observes a chair which he has not seen before, he still is able to determine that the object in question is a chair.

This ability to classify and generalize when necessary is one of the brain's most basic functions. Trying to simulate or emulate this ability is a grand challenge. There has been many designs of classifiers which can generalize. However, none of these designs have yet come close to the perfection and accuracy with which the brain operates. The accuracy of most of man-made systems is usually problem-dependent and varies greatly from one case to another. Also in some cases the classifier can only operate in a very limited and highly controlled environment, which usually is not the case in nature. For example, some of the existing speech recognition systems are speaker-dependent, meaning that they can only recognize one person's speech. While there exists some technology to develop a recognizer which is speaker-independent and even recognizes continuous speech (with no pause between the words, or even partially overlapped words), the recognition of such a system with a large vocabulary is slow and not sufficiently accurate.

Despite all this, the improvement in classification technology has been remarkable in the last decade. Alternative ideas have shed new light at the problem and offered alternative solution strategies. Perhaps the best example of such alternative ideas comes from the area of neural networks. These networks contain very simple processing units called neurons and connections which connect these units. Though the operation of the individual neurons are simple, their collective capabilities are remarkable.

The idea of neural networks was inspired by the study of the brain, especially in the early 60's. Since then, these networks have been used to perform a variety of tasks, many of which have been classification. While we are still not certain of the physical organization of the neurons in the brain or their learning strategy, scientists have developed many types of architectures and learning algorithms for these networks.

Some of the difficulties in classification problems facing neural networks today are under- or unproportional-representation of classes in the training set, highly complex boundaries between classes in a high-dimensional problem space, and training time required to learn such boundaries in such spaces.

In this thesis, a new neural network system, called the Parallel Probabilistic Self-organizing Hierarchical Neural Network (PPSHNN), is introduced to address these problems. The PPSHNN is designed especially for unusually difficult and complex classification problems, such as the ten-class remote sensing Colorado problem.

The concept of the PPSHNN module has evolved as a result of analyzing the major causes of error in classification problems. These causes can be categorized into the following:

1. Patterns of different classes which are very close to the same class boundary are usually difficult to distinguish.
2. The class boundaries may be extremely nonlinear.
3. A particular class may be undersampled such that the number of training samples

from that class are too few, as compared to other classes. Figure 1.1 a) visualizes such a scenario with Class 1 being the undersampled class as compared to Class 2.

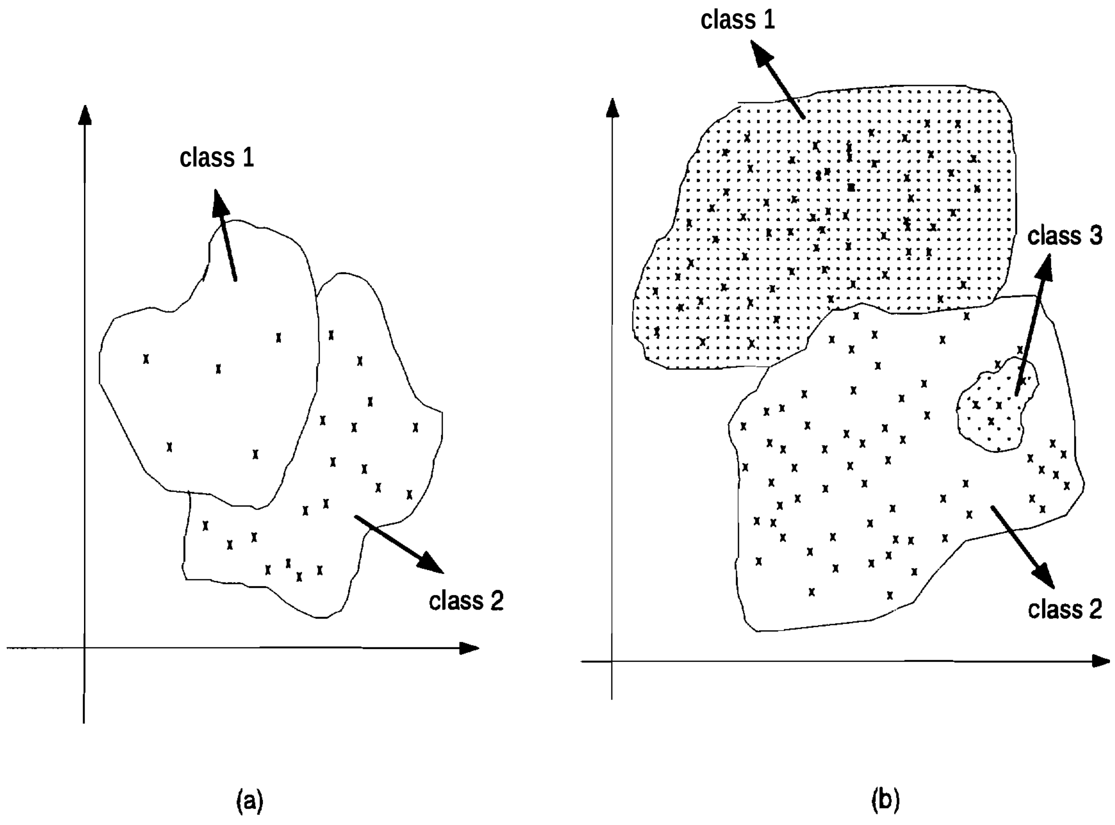


Figure 1.1. (a) An Example of an Undersampled Class (Class 1).
(b) An Example of a Geometrically Small Class (Class 3).

4. A particular class may be geometrically small compared to other classes in the sample space such that the number of training samples gathered from the region of that class is too few. This is visualized in Figure 1.1 b) where class 3 is geometrically smaller than classes 1 and 2.

The PPSHNN addresses the above problems directly. It is designed, and synthesized by a number of self-organizing modules to minimize classification error due to the mentioned difficulties.

The PPSHNN belongs to the class of Parallel Self-organizing Hierarchical Neural Networks (PSHNN) [5-8]. PPSHNN, similar to the PSHNN, is a modular neural network system whose modules run in a hierarchical fashion during training and in parallel during testing (recall). Each module of PPSHNN is quite different from the previous modules. Perhaps the three most original contributions of PPSHNN are: (1) the P-unit submodule, (2) the bitwise postrejector, (3) The SIMD implementation of PPSHNN algorithm.

The P-unit (pre-rejector) submodule is a two-class classifier and is trained to reject all the data belonging to difficult-to-classify classes such as the under- and/or unproportionally-represented classes. The P-unit is an optional unit and might not exist in some modules. Secondly, there is a statistical/adaptive postrejection unit, which consists of a statistical unit called the ***Bit-Rejector*** (BR) and an adaptive unit called the ***Vector-Rejector*** (VR). The bit rejector performs bitwise statistical analysis on every output bit of the network. The vector rejector is trained to decide whether or not to reject the classification of the input pattern based on the output of the neural network classifier and the results of the bitwise statistical analysis.

To address the problem of long training time, PPSHNN is designed such that it can easily be implemented in a Single Instruction Multiple Data (SIMD) environment. This version of PPSHNN is called the SIMD-PPSHNN and is implemented on Purdue University's Electrical Engineering Parallel Processing Laboratory's MasPar MP-1 with 16K Processing Elements (PEs).

As mentioned before, the main motivation for the design of PPSHNN came from the analysis of various causes of classification error in neural network systems. There are two major types of classification error that even the more sophisticated neural network models cannot escape. The first type occurs when data of two or more classes lie too close to a complex class boundary. The second type of error is due to the misclassification of data which belongs to a class which has significantly less number of patterns in the training set (the under- and unproportionally-represented classes) compared to other classes. There are various designs which address the first error type, including some probabilistic approaches [9, 10, 22, 25] and even some statistical-neural network approaches [16]. Unfortunately, all the probabilistic approaches used with neural networks have been statistical analysis in high dimensional spaces (vector statistics). This approach has been limiting and often inaccurate, simply due to the fact that there are not enough sample points to estimate the n-dimensional density functions accurately. Instead of statistical analysis of the input vectors, we have designed a bitwise analysis scheme at the output, called the bit-rejector. A neural network unit called the vector-rejector is trained to reject or accept a pattern based on the bitwise analysis. We also call the combination of the bit-rejectors and the vector-rejector, the ***postrejector.***

To reduce the second type of error mentioned above, a pre-rejector unit (P-unit) was designed. An additional function of the postrejector is to detect the under- and unproportionally-represented classes and. Once such a class(es) is detected, the training of a P-unit to reject the class(es) and to send it to the next module is initiated. By doing so, the classification complexity of each module is significantly reduced and, thereby, its classification accuracy increased.

The motivation for a SIMD algorithm for PPSHNN was the slow training procedure, which plagues most neural network algorithms in applications such as the 10-class Colorado problem discussed in the subsequent chapters. Considering that a simple backpropagation network, run on a Sun 3/60 station, requires over **24** hours for the training of the 10-class problem, it was essential to devise an algorithm which takes advantage of the SIMD nature of PPSHNN.

This thesis is organized in six chapters. Chapter 1 is the introduction. Chapter **2** is the background research, describing two complex classification problems and some neural network architectures which have attempted to solve these problems. In Chapter 3, the architecture and the operation of PPSHNN is discussed in detail. Chapter **4** discusses special topics in variations of the PPSHNN module such as the PNS module. Chapter 5 discusses the parallel version of PPSHNN, the SIMD-PPSHNN, and some speed-up issues. A comparison of time complexities is also provided between backpropagation, the PPSHNN, and the SIMD-PPSHNN. Chapter 6 discusses the results achieved with PPSHNN and two other previous networks. Chapter 7 covers conclusions and a discussion of future research issues.

CHAPTER 2

BACKGROUND RESEARCH

The main goal of designing the neural network system described in this thesis was to design a systems which performs better than the existing neural network architectures, specifically in dealing with complex classification problems. As a background to this issue, two known and very complex classification problems are discussed in this chapter. In addition, some details of several neural network systems which have dealt with these problems are described.

2.1 Complex System Classification

In this section, two classification problems are described which are highly complex with multi-dimensional and highly nonlinear problem space.

The first problem is that of text-to-speech conversion (speech synthesis) in the English language. The second problem is a ten-class remote sensing problem.

2.1.1 Text-to-Speech Conversion: Problem Description, Complexity Analysis

Problem description: Each sound (phoneme) in any part of a pronounced word carries features by which it is distinguished from other sounds. These features are called the articulatory features. They describe the way human vocal system produces the sounds. For example the articulatory features of the phoneme pronouncing "p" in the word "post" are: unvoiced, labial, and stop. Unvoiced means that the vocal cords are not actually moving while pronouncing the "p". Some sounds are voiced and some are unvoiced, meaning that the vocal cords do not actually move for all sounds in the English language. "p" is also labial, because in order to produce this sound, the use of the lips are essential. Some phonemes are labial and some are not. "p" is also stop, because in order to produce the sound, one must stop the flow of air out of the mouth for a short period of time and then let it out in a bursting fashion.

In order to produce the sound of a given character in the text (i.e. to pronounce the correct phoneme), one must know what the contextually appropriate articulatory features are. Thus, considering each of articulatory feature as a class, the problem becomes a classification problem. The task of the classification system is to classify each character in the text into the correct classes (features). Because each phoneme is characterized by a set of articulatory features, each input pattern belongs to all its corresponding classes (features) and should be classified as such.

Complexity description: In this thesis, by complexity we mean the difficulty of the classification task in a given problem. The first complexity factor in the speech synthesis problem is that, since each character in the text maps into several features and other characters might share one or more features, the classes are overlapped in some regions of the problem space. The second difficulty is due to the fact that some characters sound differently depending on the characters around them. In other words, such characters map into different sets of features (classes) depending on the characters in the surrounding text. This requires the classification system to be able to classify time-series as well. The best example is the case of FLAP sound. This is the case when "t"* or "d" is placed between two vowels. In this case they sound as what is called a FLAP sound. For example "catering". This effect is not word limited either. FLAP replaces /t/ or /d/ even if the above is the case over two neighboring words in the text. For example "eat it". The same is true even if there are two "t"'s or "d"'s one after the other. For example "cutting". This phenomenon of a single character mapping onto different phonemes in different context occurs with a number of letters in the English alphabet, such as "c, g, h, s" of consonants and almost all the vowels. For example, "c" maps onto the sound /k/ in the word "case", but it maps to /s/ in the word "peace".

To simplify the complexity of this problem, we reduced the alphabet set and created an English-like language in our experiment. Instead of 26 characters, we only included 8 consonants and 5 vowels. We were particularly interested in the performance of the system in the FLAP cases. Another point of interest was the fact that characters "z", "p", "o", and "u" were severly under-represented. It was interesting to see how the network

* In this thesis, when we put a character in quotes such as "t", we mean the character letter "t" of the alphabet in the written text. However, by /t/, we mean the sound (phoneme) of that written character (i.e. pronounced tee)

was going to pronounce these characters in testing. In the case of children being faced with a similar situation, first they do not pronounce the sound at all. After a few repetion (sweeps of training), they start pronouncing the new sounds, however the produced sounds are not exactly the desired sounds. They are rather sound which are already in their vocabulary of sounds and have comon features with the new sound. For example a child who knows tha sound /b/ but not /p/, would pronounce "p" with the sound /b/ at the begining. Both /p/ and /b/ are labial sounds, meaning that in order to produce them one has to use his lips. In Chapter 6, the results of these experiments using a backpropagation [1] network, a PSCNN [2], and a PPSHNN are discussed. We will see that for example the backpropagation network, easily produced the skip phenomena. Where it just did not produce the new sound. However we had a hard time finding a point in the training after that at which, it would pronounce "p" with /b/. The PPSHNN and the PSCNN exhibited this feature more easily.

2.1.2 Remote Sensing: Problem Description, Complexity Analysis

Problem description: The Colorado data set [3] consists of 7 data channels obtained from the following 4 data sources:

1. Landsat MSS data (4 data channels)
2. Elevation data (in 10m contour intervals, ldata channel)
3. Slope data (0-90 degrees in degree increments, 1 data channel)

4. Aspect data (1-180 degrees in 1 degree increments, 1 data channel)

The area used for classification is a mountainous area in Colorado. It has 10 ground cover classes which are listed in Table 2.1. Each channel

Table 2.1. The Listing of the Ten Classes of the Colorado Problem.

Class	Field
1	Water
2	Colorado blue spruce
3	Mountain/Subalpine meadow
4	Aspen
5	Ponderosa pine
6	Ponderosa pine/Douglas fir
7	Engelman spruce
8	Douglas fir/White fir
9	Douglas fir/Ponderosa pine/Aspen
10	Douglas fir/White fir/Aspen

comprises an image of 135 rows and 131 columns, all of which are co-registered.

Ground reference data were compiled for the area by comparing a cartographic map to a color composite of the Landsat data and also to a line printer output of each Landsat channel [3]. By this method, 2019 ground reference points (11.4% of the area) were selected. Ground reference consists of two or more homogeneous fields in the imagery for each class. For each class, the largest field was selected as a training field. The other fields were used for testing. Overall, 1188 pixels were used for training and 831 pixels for testing the classifiers. The number of the samples from each class are shown in Table 2.2.

Based on the information received, we want to decide which class the received data vector belongs to.

Complexity description: One problem with the data set discussed above is that some of the classes are extremely under-represented. For example, class 9 has only 25 samples in the training set. This is 2.1% of the training set. In a training sweep, the number of samples in classes 1, 5, 6, and 7 constitute more than 72% of the set. This uneven representation of classes in training causes the network to ignore the under-represented classes and only learn the well-represented ones. An additional problem is the highly nonlinear separation of the classes. The mentioned problems and other discovered and undiscovered difficulties combine to manufacture an extremely difficult classification problem. The 10 class Colorado classification problem is by far more difficult than the speech synthesis problem. The best previous results offered by neural networks for this problem was around 53%. See chapter 6 for PPSHNN results.

Table **2.2** Number of Samples of each Class for the Colorado Data Set.

Class	Training(1188)	Testing(831)
1	408	195
2	88	24
3	45	42
4	75	65
5	105	139
6	126	188
7	224	70
8	32	44
9	25	25
10	60	39

2.2 Backpropagation

The most often used neural networks for classification are backpropagation networks **[2]**. There are many different variations of the backpropagation ***(generalized delta rule)*** algorithm depending on the type of neurons and the descent algorithm used. Here we will describe the most commonly used version which uses the ***gradient descent*** algorithm ***[4]***and is what we used in our experiments.

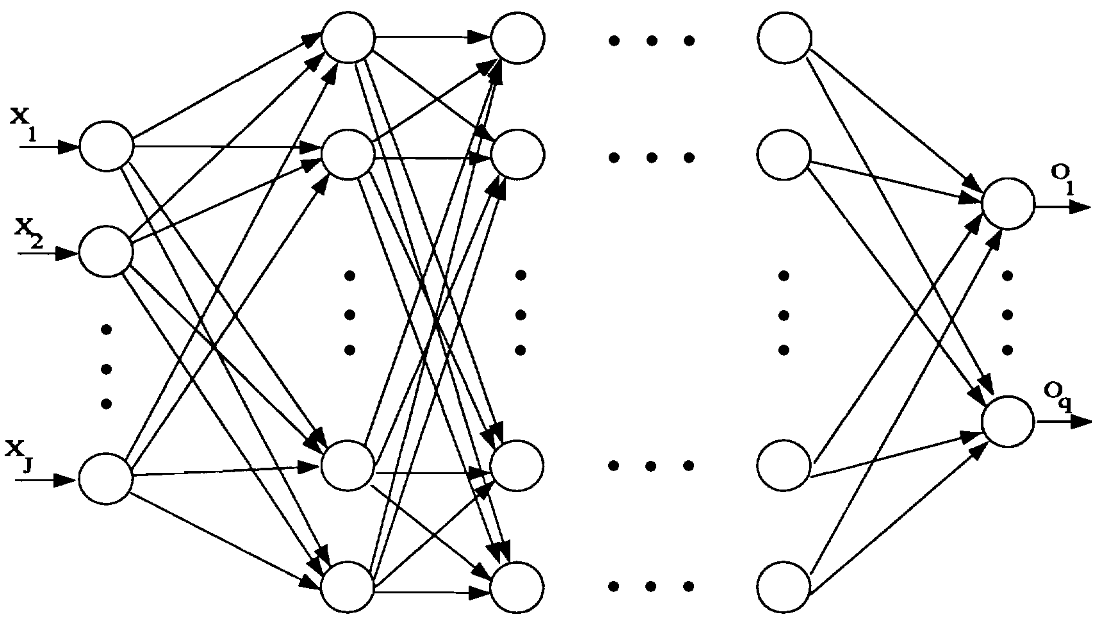


Figure 2.1 Multilayered, Feed Forward Network.

The network is multi-layered [4] and feed-forward [4] (Figure 2.1). Its neurons are standard neurons with a sigmoid function as their activation function [1]. The activation function for the jth neuron is

$$f(s_j) = \frac{1}{1 + e^{-(s_j - \theta_j)}}. \tag{1}$$

Where θ_j is the threshold for jth unit and

$$s_j = \sum_i x_i \, \omega_{ji} . \tag{2}$$

x_i is the ith input to the neuron and ω_{ji} is the weight of the connection between the ith input and the jth neuron.

During training, an input vector is presented to the network and an output vector is computed and compared to the desired output vector we would like to see at the output. Once this is done, an error value is computed for every output bit of the network. The error values are backpropagated through the network, and based on the value of error passing through each connection, the weight of that connection is updated.

Let d_{pj} be the desired output value for output bit j for the pth vector in the training set. In the same manner, let O_{pj} be the actual output value of output bit j for the pth pattern in the training set. Then the squared error for the pth vector of the training set is

$$E_p = \sum_j (d_{pj} - O_{pj})^2. \tag{3}$$

The total error for a training sweep is

$$E = \sum_p E_p. \tag{4}$$

Using ***delta rule*** [1], we reduce the value of E by implementing ***gradient descent*** [4]. By taking the partial derivative and using the chain rule with respect to s_{pj}, the summation value of neuron j for pattern p of training set, we get

$$\frac{\partial E_p}{\partial \omega_{ji}} = \frac{\partial E_p}{\partial s_{pj}} \cdot \frac{\partial s_{pj}}{\partial \omega_{ji}} \,. \tag{5}$$

Using (2), we get

$$\frac{\partial s_{pj}}{\partial \omega_{ji}} = \frac{\partial}{\partial \omega_{ji}} \sum_k x_{pk}\, \omega_{jk} = x_{pi} \,. \tag{6}$$

Now, let us define

$$\delta_{pj} = -\frac{\partial E_p}{\partial s_{pj}} . \tag{7}$$

Then, equation (5) becomes

$$-\frac{\partial E_p}{\partial \omega_{ji}} = \delta_{pj} x_{pi} . \tag{8}$$

This says that to implement gradient descent in E, we should make our weight changes according to

$$\Delta_p \omega_{ji} = \eta \delta_{pj} x_{pi} , \tag{9}$$

just as in the standard ***delta rule*** [1]. The trick is to find out what δ_{pj} should be for each unit in the network. It can be shown [I] that for neurons in the output layer

$$\delta_{pj} = (d_{pj} - O_{pj}) f'_j(s_{pj}), \tag{10}$$

and for the neurons in the hidden layer(s)

$$\delta_{pj} = f'_j(s_{pj}) \sum_k \delta_{pk} \omega_{kj}. \tag{11}$$

where δ_{pk} is the error propagating backwards in the network from neuron k.

A two stage (one hidden layer) backpropagation network was used for the classification problems mentioned. One major issue in backpropagation networks is to find the correct number of hidden neurons in the hidden layer(s). See Chapter 6 for more detail.

2.3 PSCNN and PSHNN

Both Parallel Self-organizing Consensual Neural Network (PSCNN) [2] and the Parallel Self-organizing Hierarchical Neural Network (PSHNN) [5-8] are modular networks (Figure 2.2). Each module may consist of a single

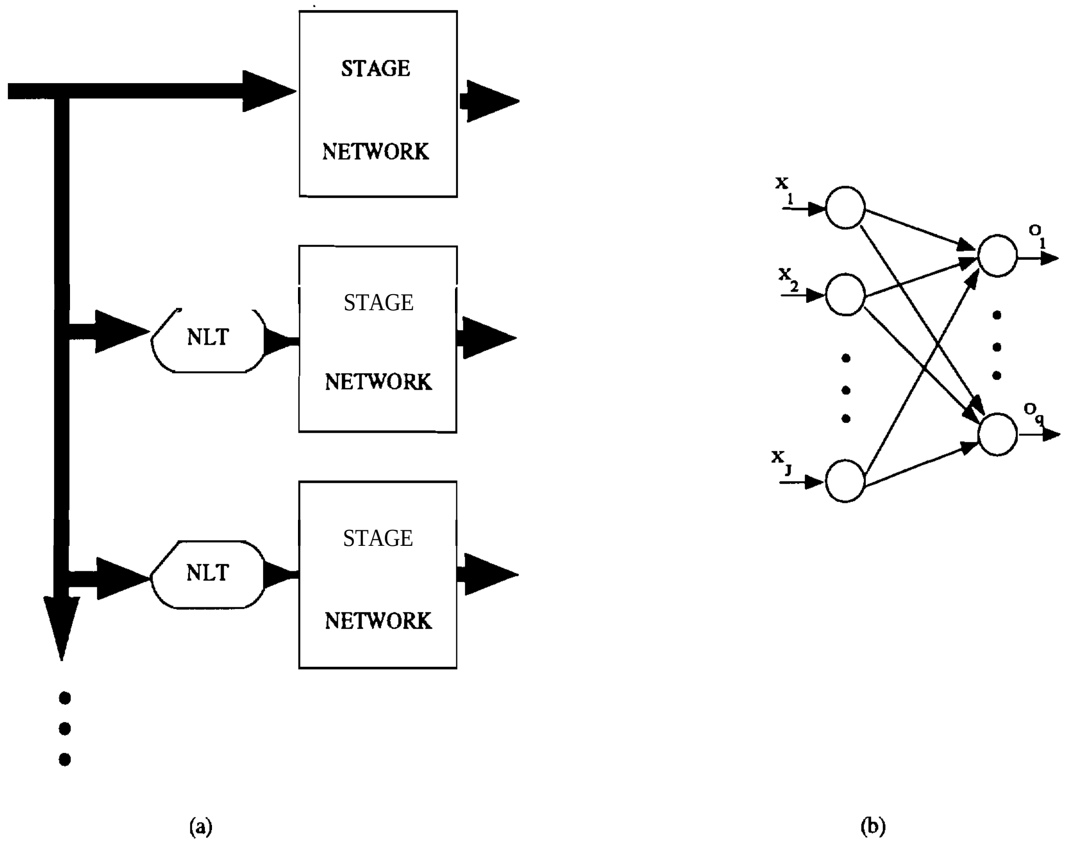


Figure 2.2. PSCNN, PSHNN Network.

stage fully connected feed-forward delta rule network (Figure 2.2 (a)). All except module one also have a Nonlinear Transformation (NLT) unit. Input data to each module is nonlinearly transformed and then fed into the stage network. In training, the system uses the stage network algorithm such as the delta rule to learn the input pattern.

In testing, it produces a classification output.

There is a rejection mechanism at the output of each output bit of each module. There are rejection boundaries (and certainty boundaries in PSCNN) which are learned similar to the weights during training. Learning rule for both PSCNN and PSHNN modules can be chosen to be any desired learning algorithm. Previously it has mostly been chosen to be the delta rule, which is similar to generalized delta rule described in the previous section.

In PSHNN, there is a hierarchy in training. In other words, module i is only trained with the data rejected by module $i-1$. In PSCNN on the other hand, modules are trained with all available data for training. This allows modules of PSCNN to be trained in parallel. During testing, each module of PSCNN votes for classification of input data. Then a consensus is taken based on the classification votes of all modules and the certainty of their votes. On the other hand, in PSHNN, the vote of module $i-1$ has precedence to that of module i. Thus if module $i-1$ classifies the incoming data (in other words, not rejects it), the classification of modules i and higher are ignored.

See chapter 6 for the classification results of PSCNN.

CHAPTER 3

PARALLEL, PROBABILISTIC, SELF-ORGANIZING, HIERARCHICAL NEURAL NETWORKS

In the previous chapter we discussed two complex classification problems in multi-dimensional spaces. In problems such as these, the high dimensionality of problem space, in addition to other factors, usually makes classification difficult. Due to the high dimensionality of this space, we need an extremely large data set for training, which in most cases is not available. We have also seen that in addition to the limited training data set in problems such as the remote sensing problem, some classes might be severely under-represented.

In Chapter 2, we have also seen some of the solutions to these problems which have been offered by neural networks (BP-networks, PSHNN, and PSCNN).

In this chapter we discuss a new type of neural networks, the ***Parallel Probabilistic Self-organizing Hierarchical Neural Network*** (PPSHNN), to reduce classification errors. The PPSHNN is designed especially for complex and high dimensional problems. Its major contributions are implementing a pre-rejection unit (P-unit) (see section 3.5) to reduce the complexity and possibly dimensionality of the classification space for the neural network unit (N-unit)* (Section 3.2), the bitwise Post-Rejection scheme (Section 3.3) which implements bit level statistical analysis to detect the errors made by the N-unit, and its parallel implementation in a SIMD fashion on MasPar MP-1 (Chapter 5). Because the P-unit and the postrejector units are adaptive, PPSHNN is very flexible as far as allowing the user to choose any type of network for P- and N-units. In our experiments, we have mainly used single stage delta rule networks for the P- and N-units. In some experiments we also used two stage backpropagation networks.

In the following sections, we shall see how PPSHNN is better equipped to address problems such as under-representation in training set, limited training data for very high dimensional problem spaces, highly non-linear and complex classification spaces, and so on. The PPSHNN also addresses the time complexity issues which back propagation networks have had. It can be shown that the time required for training a backpropagation network grows in the order of $O(n_h n_o)$ (see Section 5.3), where n_h is the size of the largest hidden layer. It is known that size of the hidden layer grows with the complexity of the application. For complex problems such as the 10-class remote sensing problem, backpropagation networks are painfully slow and sometimes require many days of training on an average work station.

On the other hand, due to the parallel nature of PPSHNN, we will show (Section 5.3) that the training time complexity of PPSHNN grows in the order of $O(n_i n_o)$. Note that both

* **In the basic PPSHNN module there are two neural network units, the pre-rejector and the neural network classification unit. By N-unit we mean the later. This unit is also not to be mistaken with the nearest neighbor classifier which is referred to just as the classifier.**

n_i and n_h are predetermined and not complexity dependent. Hence, the time complexity of PPSHNN grows only at a constant rate (i.e. $O(1)$) with respect to the complexity of the problem. Furthermore, by running the parallel version of PPSHNN, the SIMD-PPSHNN, on a SIMD machine such as MasPar MP-1, we can cut the training time by several orders of magnitude. In Section 5.3, we will make a time complexity analysis of the BP, PPSHNN, and SIMD_PPSHNN networks.

In chapter 4 we discuss the PNS module and the implementation of PPSHNN using these modules as its building blocks.

3.1 PPSHNN System Description

Figure 3.1 shows three modules of a PPSHNN network. Module 1 consists of four submodules and a communication link, and all the following modules consist of five submodules.

In the following, we describe briefly the function of each submodule and then the overall function of PPSHNN. In Sections 3.3 through 3.6 we will describe the details of each module.

The general idea behind the PPSHNN is to divide the problem space into polygons, and then perform the task of classification in each one of the polygons independently, rather than trying to do this in the entire problem space. The goal is to divide the problem space in such a manner that classification is easier in at least one of the resulting

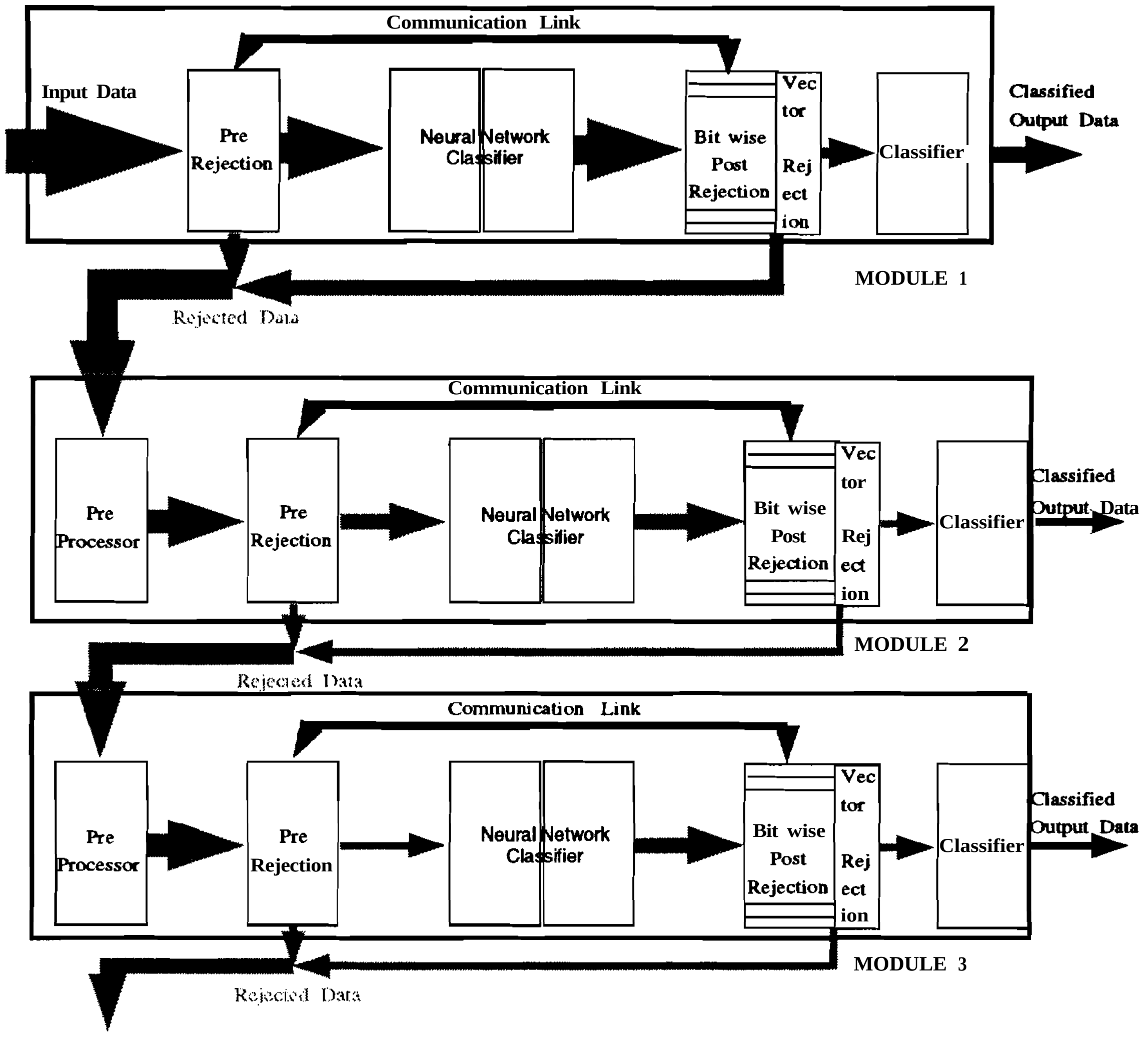


Figure 3.1. PPSHNN with 3 Modules.

polygons. The task of dividing the problem space into polygons is performed primarily by the pre-rejector (P-unit) (Section 3.5).

Once this is done, the neural network unit (N-unit) performs classification on data which fall into the easier regions. The rest of the data rejected by the P-unit is sent to the next (lower) module in the hierarchy. Since in complex problem spaces there are some data points which "pass" the pre-rejection test but still are difficult to classify, and are misclassified by the N-unit, a mechanism is required at the output of the N-unit to detect these data, reject them, and send them to the next module. This is done by a probabilistic mechanism at the output of the N-unit called the Post-Rejection (Section 3.3). This mechanism consists of two modules. The first performs a bit level probabilistic analysis of the individual output bits of the N-unit and is referred to as the Statistical unit or the S-unit. The second combines the results of the bit analyses and decides whether or not to reject the input pattern. This unit is referred to as the Vector Rejector or the VR.

There is a communication link between the P-unit and the postrejector. In many cases, one or more classes of data are too complex for the N-unit to classify. This results in an unusually low classification accuracy for these classes and most of the patterns belonging to these classes must be rejected. In such cases, instead of training the postrejector to reject each one of the individual patterns, the classes are communicated to the P-unit through the communication link. The P-unit is then retrained to reject these classes along with the ones it has already been trained to reject. If no P-unit exists for the module, one is created and is then trained to reject these classes.

During testing, if an input pattern is rejected by the postrejector, it joins the rejected data vectors from the P-unit and is sent to the next module. If accepted, the output data of the N-unit is sent to the distance *classifier* for a nearest neighbor match to a set of pre-set decoding patterns in order to convert the output vector of the N-unit to the required output format.

To determine the final P-unit, N-units and the postrejector, a number of retrainings of these units may be necessary. Initially, there is no P-unit. The class(es) rejected by the postrejector signal the creation of the P-unit. The P-unit is then created by training a neural network as a two-class classifier with the accepted and the rejected set of input vectors as determined by the postrejector. This leads to a reduced data set to be fed to the N-unit, which is then retrained. The postrejector is also retrained to determine whether or not more vectors or classes are to be rejected. If so, the classes are notified to the P-unit and the individual vectors are rejected by the postrejector itself. This process is repeated for a number of sweeps until all three units stabilize in terms of accepted and rejected vectors.

The process described above may be considered excessive in terms of learning time, due to the many sweeps which may be needed. In order to reduce this problem, two strategies are possible. The first is to limit the number of sweeps to a predetermined value. This could result in a higher number of rejected patterns and a higher number of modules required for proper classification. The second strategy is to decide to create a P-unit only if all or a predetermined high percentage of the input vectors from a class are rejected by the postrejector. In the latter strategy, the P-unit has the task of detecting classes which are difficult to classify as a whole since they may be underrepresented and so on. This strategy has been used in our computer simulations. The predetermined percentage was set to be 100%. With this strategy, only a single sweep is generated. The postrejector still rejects a number of input vectors which are accepted by the P-unit, but does not further notify the P-unit so that no more sweeps are generated.

The rejected data is sent to the next module to repeat the process. First, this data goes through the Pre-Processor. The function of this optional unit is comparable to that of the non-linear transformation performed in PSCNN or in PSHNN. This module non-linearly changes the way the sub-problem space is presented to the network. The non-linear transformation could be a neural network unit and thus learn the non-linear transformation during training. This transformation is problem-dependent, and not a preset transformation which may or may not work well on a given problem. For many problems this unit may be skipped, and only the P-unit is used.

For better understanding of the operation of the PPSHNN, we consider the 2-dimensional problem space shown in Figure 3.2.a. It contains three classes: A, B, and C. Figure 3.2.b shows how the P-unit of module 1 has divided the space into two polygons. The shaded area is the reject region, and data falling in this area is rejected. The remaining region of the space is the accept area, and data falling in this region are sent to the N-unit for classification. Figure 3.2.c shows the space which is passed to the N-unit of module one. We see that, since class C is not present in the data sent to the N-unit, the N-unit is only a two-class classifier. After this stage, the output of the N-unit is sent to the postrejector to reject the uncertain classifications. Data falling in the shaded area of Figure 3.2.d is rejected to the next module by the postrejector. Notice that the function of both the pre-rejector and the postrejector is to reject data which fall in the area of problem space where classification is difficult.(ie. near the border between two or more classes, etc.).

Figure 3.2.e shows the problem space that is introduced to the second module. This space consists of all the data rejected by the pre- and postrejectors of the previous

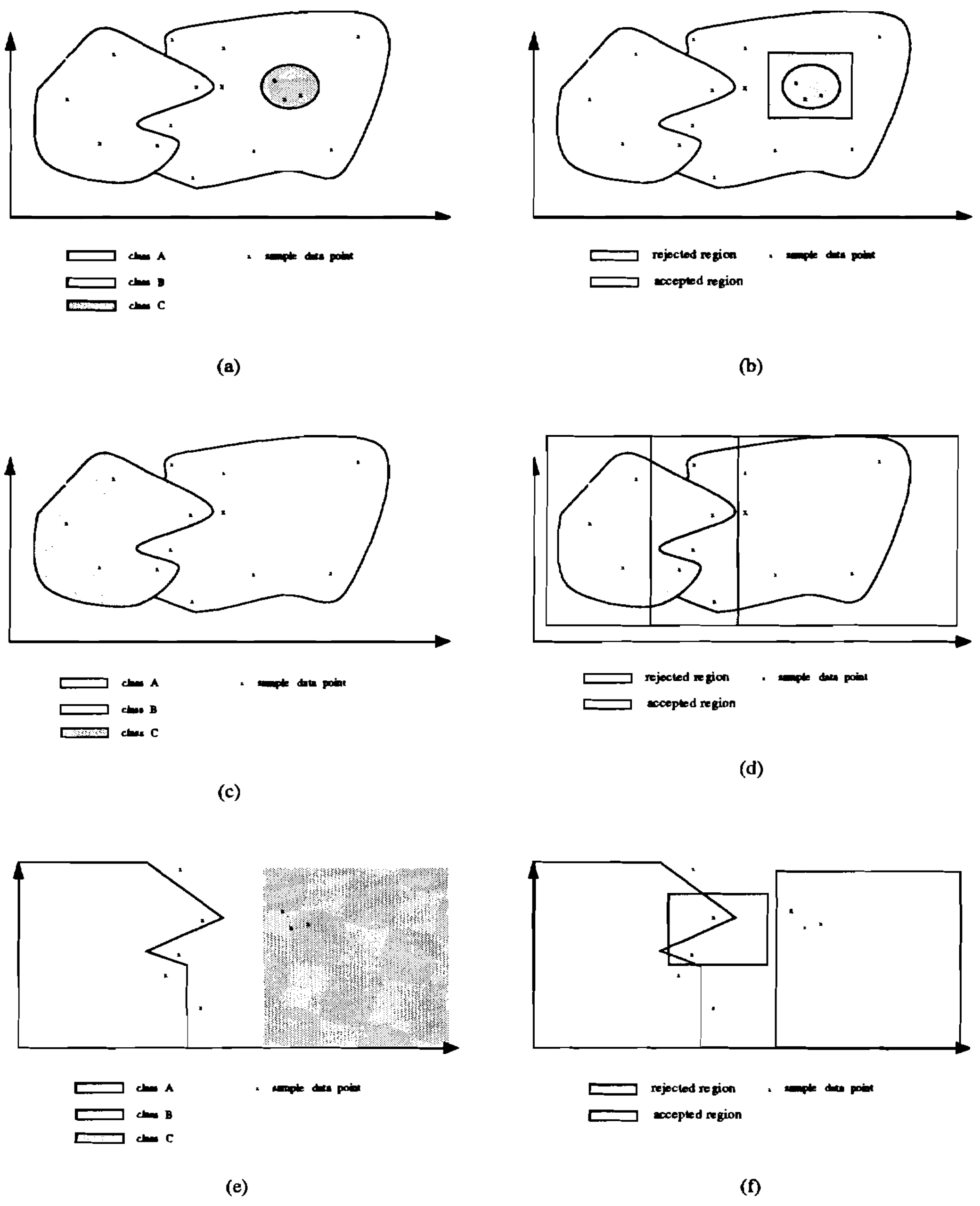


Figure 3.2 Sample Problem Space Initial Stage.

module. Notice that the new problem space is less complex than the original problem space. Also notice that, data belonging to any class which might have been under-represented for the first module, is not so for the second module. This is due to the fact

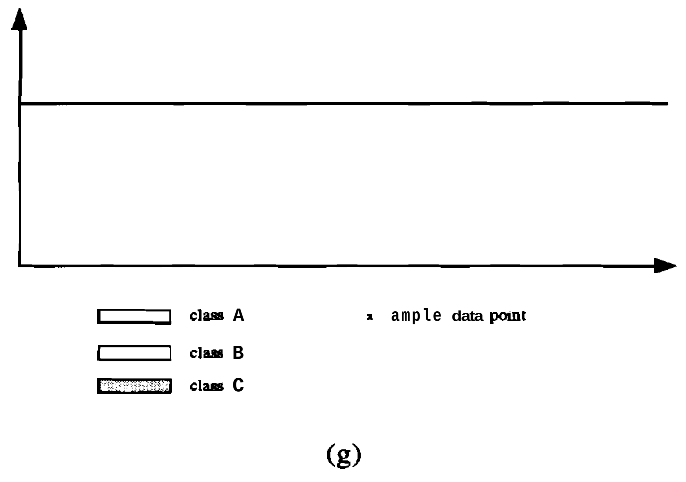


Figure 3.3 Sample Problem Space Final Stage.

that most of the data belonging to large classes are classified by the first module and do not exist in the problem space of the second module.

The second module repeats this procedure in its own problem space. Figure 3.2.f shows the reject area of the postrejector of second module. Note that since there are no under-represented classes present in the polygon of the second module, a P-unit is not needed for this module. Figure 3.3 shows the problem space introduced to the third module (again no P-unit is created for this module). Notice that, for the lower modules in the hierarchy, some of the classes present in the original problem space may vanish. This makes classification easier and opens possible avenues to reduce the dimensionality of the problem space. For example, in Figure 3.3, since the border separating class A from class B is horizontal (could also be vertical), one could perform classification simply by having a threshold on the Y-axis (or X-axis), thus, making it a one dimensional classification. A mechanism is needed to perform the reduction of dimensionality on the incoming data points in such cases. This task of dimensionality reduction could be performed by the pre-processor.

Training procedure: Figure 3.4 shows a flow chart of the training procedure of PPSHNN. In creating and training PPSHNN for a classification problem, first we start with no P-unit. This unit is created only after the postrejector has requested it through the communication link.

First the N-unit for module 1, named *N(1)*, is created. Then this network is trained, until there is little change in the classification accuracy. After which a bit level statistical analysis of the output is performed using output data from last sweep of training of N(1). After this point, there is a decision to be made as to whether or not a P-unit is needed for this module.

This decision is made based on the p_1^k calculated by each bit rejector, where p_1^k is the percentage of data correctly classified as class k (see section 3.4.1). There is a preset minimum percentage threshold. If p_1^k is less than this preset value for any k, a P-unit for that module is required.

If there is any rejected class, then it is signaled through the communication link to initiate the procedure of creating a P-unit. This procedure reduces the size of the output layer of the N-unit by eliminating the output bits corresponding to the class(es) which are to be rejected. Then, the P-unit is created. This unit is a two-class neural network classifier. It is trained with the training data set which the N-unit was trained with. It is trained to reject the classes determined by the postrejector. Other input data are classified as accept

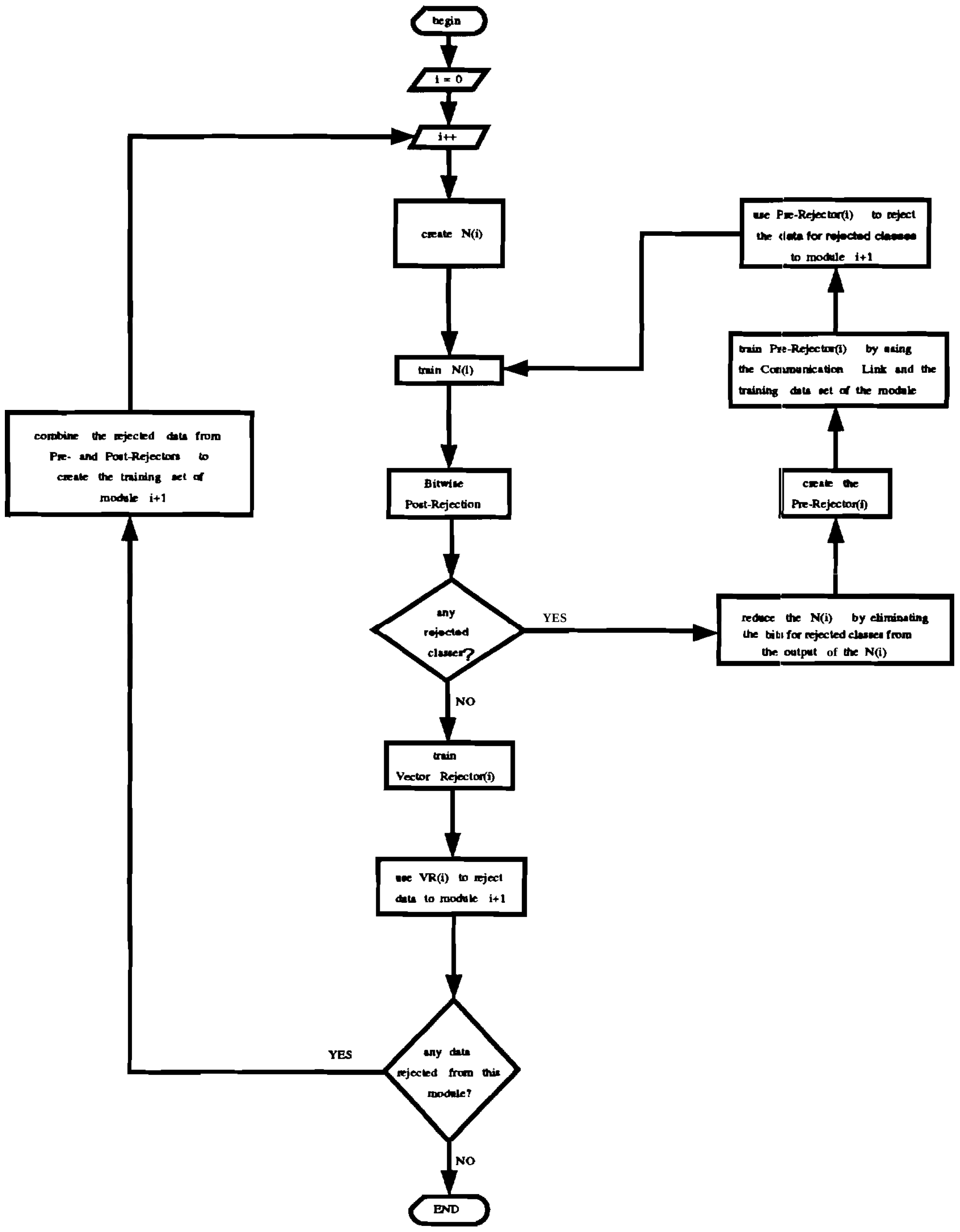

begin
i = 0
i++
create N(i)
train N(i)
Bitwise Post-Rejection
any rejected classes?
YES
NO
train Vector Rejector(i)
use VR(i) to reject data to module i+1
any data rejected from this module?
YES
NO
END
reduce the N(i) by eliminating the bits for rejected classes from the output of the N(i)
create the Pre-Rejector(i)
train Pre-Rejector(i) by using the Communication Link and the training data set of the module
use Pre-Rejector(i) to reject the data for rejected classes to module i+1
combine the rejected data from Pre- and Post-Rejectors to create the training set of module i+1

Figure 3.4 Training Procedure of PPSHNN.

anti are sent to the N-unit for classification. In other words, the P-unit eliminates the input vectors which are difficult to classify and, as a result, the N-unit is introduced only to a subregion of the original problem space. After the data set is divided into a rejected set and an accepted set, the retraining of the N-unit with the accepted data begins. If the number of classes is reduced, the size of the N-unit will be smaller. After retraining the N-unit, the process moves on to training the vector rejector with the output data of the N-unit and the S-unit. This unit is trained to decide based on the bitwise information, whether or not, to reject the input pattern to the next stage. After this point, all the data rejected by the postrejector and the P-unit (if present) are gathered together to build a training set for the next module. This process is repeated with succeeding modules until no, or few, data patterns are rejected.

An important feature of the PPSHNN modules is that modules become simpler as more of them are created. The P-unit is not created in most cases after the second module and the N-unit becomes smaller.

Testing procedure: In testing the hierarchical processing involved in creating modules is replaced by parallel processing. All modules are run in parallel, and each one classifies the incoming data into a class or rejects it. Due to the hierarchical nature of the training procedure, in testing, once module i has classified the incoming pattern into one of the possible classes (in other words it has not rejected the pattern), the classification results of modules i+1 and lower are ignored.

3.2 The Neural Network Classifier (N-unit)

This network is a neural network construct. We experimented with both backpropagation networks and single stage delta rule networks for this unit. In Section 2.2, backpropagation algorithm was described in some detail. The backpropagation network used complies with all the specifications given in that section and in [1]. The network has only one hidden layer and the layers are fully connected to each other without jumps over the hidden layer. The delta rule networks used are single stage backpropagation networks (no hidden layer). Therefore, Equation (11) does not apply, and all weights are updated according equations (9) and (10).

The design of PPSHNN is quite flexible, even allowing different types of networks to be used for the N-units of different modules. Due to the adaptive nature of the P-unit and the postrejector submodules, the system is able to adapt and function properly.

3.3 Post Rejection

This unit is a combination of a set of probabilistic classifiers (bitwise postrejectors) and a single stage Neural Network classifier (vector rejector). See Figure 3.5.

There is a bit classifier for every output bit. This classifier is a three-class Bayesian classifier which classifies the output bit into one, zero, or reject classes.

The vector classifier is a neural network construct which looks at classifications made by the bit classifiers and decides whether or not to reject that input pattern. If the vector is

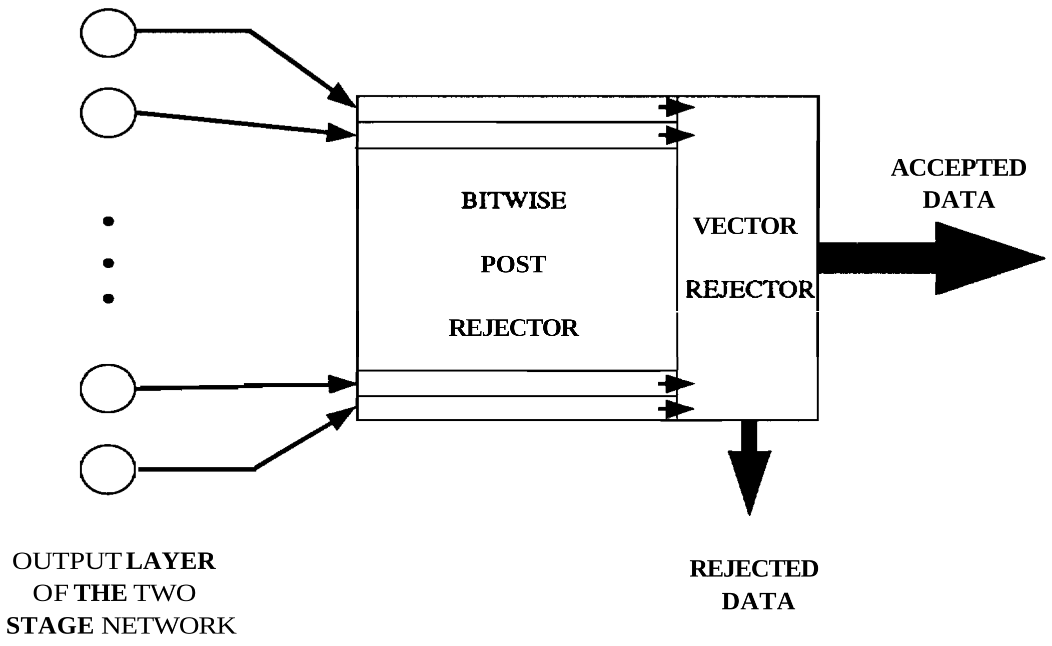


Figure 3.5 Post-Rejector.

not rejected, it is classified into one of the possible classes. If the data is rejected, it is sent to the next module for classification.

3.3.1 Bitwise Rejection (S-unit)

Bitwise rejection is performed by the bitwise classifiers. Each bitwise classifier is a three-class Maximum A Posteriori (MAP) Detector [9]. It is well-known from statistical decision theory that a Bayes receiver [10] minimizes the average cost of making a decision and is implemented by means of the likelihood ratio test. In the following we shall derive these ratio tests for a three class case. The idea is to look at neural network (N-unit) from a different point of view. Namely, we look at the network as part of a

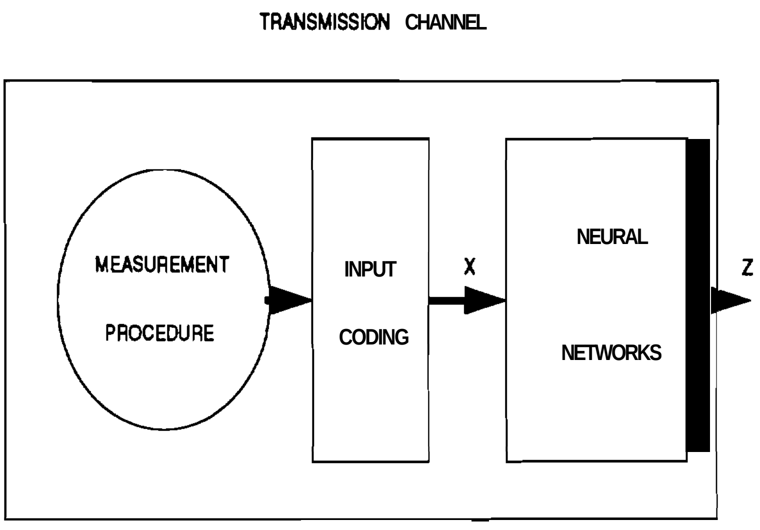


Figure 3.6 Transmission Channel Model.

transmission channel (see Figure 3.6) and we look at the output vector as the received signal from this channel. The transmission channel consists of the measurement procedure, coding the measurements into a pre-decided format and finally putting the signal through the network. All three stages of this channel can add noise to the signal. The measurement noise, the wrong coding scheme, an undertrained network, a wrong sized and/or structured network are all examples of potential noise-adding elements in the channel.

For the output bit k with the output value z of the N-unit, three hypotheses are possible:

H_0 = ***Bit k should be classijed as zero .***
H_1 = ***Bit k should be classijed as one .***
H_r = ***Bit k should be rejected .***

Notice that we consider the rejected data as a class by itself. This way we acknowledge the fact that some data points are not classifiable in their present representation. In

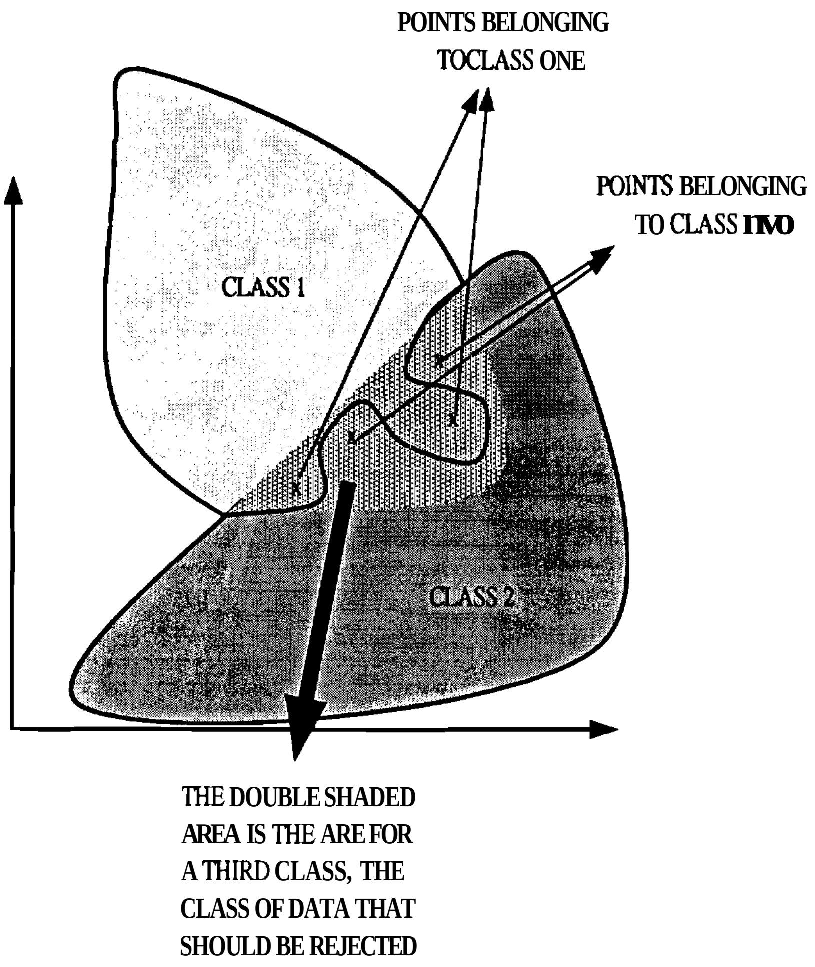


Figure 3.7 Sample Nonlinear Problem Space.

Figure 3.7 a simple example of this in 2-D space is shown.

We establish the following notation:

$f_z^k(z\,|\,H_i)$ = **probability density function of the output value of bit k given that H_i is true.**

z^k = **output value of the k^{th} output bit of the N-unit.**

C_{ij}^k = cost of deciding hypothesis H_i is true when H_j was actually true for bit k.

$p_i^k = p^k(H_i)$ = a priori probability for bit k that hypothesis H_i is true (i.e. $p_1^k = 1 - p_0^k - p_r^k$).

$P^k(H_i \mid z)$ = probability of hypothesis H_i being true for bit k, given the output value z from the N-unit.

The a posteriory probability $P^k(H_i \mid z)$ can be computed from $f_z^k(z \mid H_i)$ using Bayes rule [10]:

$$P^k(H_i \mid z) = \frac{f_z^k(z \mid H_i)\, p^k(H_i)}{f_z^k(z)} \tag{12}$$

Suppose that we observe a particular z on output bit k and that we decide it belongs to hypothesis H_i. If the true classification is H_j, the expected loss associated with chosing H_i is merely

$$R^k(H_i \mid z) = \sum_j C_{ij}^k\, P^k(H_j \mid z) \qquad i,\ j\ \varepsilon\ \{\,0,\ 1,\ r\,\} \tag{13}$$

Thus, the expected loss for choosing H_0 given output value z at bit k is

$$R^k(H_0 \mid z) = C_{00}^k P^k(H_0 \mid z) + C_{01}^k P^k(H_1 \mid z) + C_{0r}^k P^k(H_r \mid z)\,. \tag{14}$$

The expected loss for choosing H_1 given output value z at bit k is

$$R^k(H_1 \mid z) = C_{10}^k P^k(H_0 \mid z) + C_{11}^k P^k(H_1 \mid z) + C_{1r}^k P^k(H_r \mid z)\,, \tag{15}$$

and the expected loss for choosing H_r given output value z at bit k is

$$R^k(H_r\,|\,z) = C^k_{r0}P^k(H_0\,|\,z) + C^k_{r1}P^k(H_1\,|\,z) + C^k_{rr}P^k(H_r\,|\,z). \tag{16}$$

In decision theoretic-terminology, an expected loss is called a *risk,* and $R^k(H_i\,|\,z)$ is known *as* the *conditional risk.* Whenever we encounter a particular output **z,** we can minimize our expected loss by selecting the hypothesis that minimizes the conditional risk.

Now we can show that this is the same as the optimal Bayes decision procedure:

Let us define a decision function $\zeta^k(z)$ which chooses a hypothesis for output value z at output bit k. The overall risk R is the expected loss associated with a given decision rule. Since $R^k(H_i\,|\,z)$ is the conditional risk associated with chosing H_i, and since the decision rule specifies the hypothesis chosen, the overall risk is given by

$$R = \int R(\zeta^k(z)\ |\ z)\, f^k_z(z)\, dz \tag{17}$$

Where dz is the notation for a d-space volume element, and where the integral extends over the entire feature space. Clearly, if $\zeta^k(z)$ is chosen so that $R(\zeta^k(z)\ |\ z)$ is as small as possible for every z, then the overall risk will be minimized. This justifies the following statement of the Bayes decision rule: To minimize the overall risk, compute the conditional risk

$$R^k(H_i\,|\,z) = \sum_j C^k_{ij}\, P^k(H_j\,|\,z) \qquad i,\ j\ \varepsilon\ \{\,0,\ 1,\ r\,\}$$

and select the H_i for which $R^k(H_i\,|\,z)$ is minimum.

Thus, for every output value **z** at every bit k there are three tests to perform. Using results of these tests we define the following decision rule which has minimum ***risk:***

$$\zeta^k(z) = \begin{cases} \text{if } R^k(H_0|z) < R^k(H_1|z) \;\&\; R^k(H_0|z) < R^k(H_r|z) & \textit{chose } H_0 \\ \text{if } R^k(H_1|z) < R^k(H_0|z) \;\&\; R^k(H_1|z) < R^k(H_r|z) & \textit{chose } H_1 \\ \qquad\qquad \textit{otherwise} & \textit{chose } H_r \end{cases} \tag{18}$$

TEST 1:

The first test is between H_0 and H_1:

$$R^k(H_0|z) \underset{H_0}{\overset{H_1}{\gtrless}} R^k(H_1|z) \tag{19}$$

Now let $C^k_{00} = C^k_{11} = C^k_{rr} = 0$. This means that there is no cost for guessing the correct hypothesis, which is the case in most classification problems. Then the inequality reduces to

$$C^k_{01}P^k(H_1|z) + C^k_{0r}P^k(H_r|z) \underset{H_0}{\overset{H_1}{\gtrless}} C^k_{10}P^k(H_0|z) + C^k_{1r}P^k(H_r|z), \tag{20}$$

$$C^k_{01}P^k(H_1|z) - C^k_{10}P^k(H_0|z) \underset{H_0}{\overset{H_1}{\gtrless}} (C^k_{1r} - C^k_{0r})\, P^k(H_r|z). \tag{21}$$

Assuming $f^k_z(z) \neq 0$, we can multiply both sides by $f^k_z(z)$. Thus we get

$$C_{01}^{k}P^{k}(H_1\,|\,z)f_z^{k}(z)-C_{10}^{k}P^{k}(H_0\,|\,z)f_z^{k}(z)\ \underset{H_0}{\overset{H_1}{\gtrless}}\ (C_{1r}^{k}-C_{0r}^{k})P^{k}(H_r\,|\,z)f_z^{k}(z)\,. \qquad (22)$$

Using Bayes rule (12) and assuming $P_i^k \neq 0$, Eq. (22) becomes

$$C_{01}^{k}f^{k}(z\,|\,H_1)P^{k}(H_1)-C_{10}^{k}f^{k}(z\,|\,H_0)P^{k}(H_0)\ \underset{H_0}{\overset{H_1}{\gtrless}}\ (C_{1r}^{k}-C_{0r}^{k})f^{k}(z\,|\,H_1)P^{k}(H_r). \qquad (23)$$

Choosing $C_{10}=C_{01}$ and $C_{1r}=C_{0r}$ and $C_{r0}=C_{r1}$ leads to the following:

$$\frac{f_z^{k}(z\,|\,H_1)}{f_z^{k}(z\,|\,H_0)}\ \underset{H_0}{\overset{H_1}{\gtrless}}\ \frac{C_{10}\,P^{k}(H_0)}{C_{01}\,P^{k}(H_1)}\,, \qquad (24)$$

or

$$p_1^{k}\,f_z^{k}(z\,|\,H_1)\ \underset{H_0}{\overset{H_1}{\gtrless}}\ p_0^{k}\,f_z^{k}(z\,|\,H_0)\,. \qquad (25)$$

TEST 2:

The second test is between H_0 and H_r

$$R^{k}(H_0\,|\,z)\ \underset{H_0}{\overset{H_r}{\gtrless}}\ R^{k}(H_r\,|\,z)\,. \qquad (26)$$

Using (14) and (16) and choosing $C_{00}=C_{rr}=0$, yields

$$C^k_{01}P^k(H_1\,|\,z) + C^k_{0r}P^k(H_r\,|\,z) \underset{H_0}{\overset{H_r}{\gtrless}} C^k_{r0}P^k(H_0\,|\,z) + C^k_{r1}P^k(H_1\,|\,z) \tag{27}$$

Using Bayes rule (12) and applying the same conditions as in Test 1, we obtain

$$C^k_{01}P^k(H_1\,|\,z)f^k(z)+C^k_{0r}P^k(H_r\,|\,z)f^k(z) \underset{H_0}{\overset{H_r}{\gtrless}} C^k_{r0}P^k(H_0\,|\,z)f^k(z)+C^k_{r1}P^k(H_1\,|\,z)f^k(z)\,, \tag{28}$$

$$C^k_{0r}f^k(z\,|H_r)P^k(H_r)-C^k_{r0}f^k(z\,|H_0)P^k(H_0) \underset{H_0}{\overset{H_r}{\gtrless}} (C^k_{r1}-C^k_{01})f^k(z\,|H_1)P^k(H_1)\,, \tag{29}$$

$$C^k_{0r}f^k(z\,|H_r)p^k_r - C^k_{r0}f^k(z\,|H_0)p^k_0 \underset{H_0}{\overset{H_r}{\gtrless}} (C^k_{r1} - C^k_{01})f^k(z\,|H_1)p^k_1\,. \tag{30}$$

TEST 3:

The third test is between H_r and H_1:

$$R^k(H_1\,|\,z) \underset{H_1}{\overset{H_r}{\gtrless}} R^k(H_r\,|\,z)\,. \tag{31}$$

With the same assumptions as in the previous two tests and the same operations, a final inequality for test 3 can be reached:

$$C^k_{10}f^k(z\,|H_0)p^k_0 + C^k_{1r}f^k(z\,|H_r)p^k_r \underset{H_1}{\overset{H_r}{\gtrless}} C^k_{r1}f^k(z\,|H_1)p^k_1 + C^k_{r0}f^k(z\,|H_0)p^k_0 \tag{32}$$

$$C^k_{1r}f^k(z\,|H_r)p^k_r - C^k_{r1}f^k(z\,|H_1)P^k_1 \;\overset{H_r}{\underset{H_1}{\gtrless}}\; (C^k_{r0} - C^k_{10})f^k(z\,|H_0)P^k_0\,. \tag{33}$$

The final three inequalities resulting from the above three test are as follows:

$$p_1\, f^k_z(z\,|H_1) \;\overset{H_1}{\underset{H_0}{\gtrless}}\; p_0\, f^k_z(z\,|H_0)\,,$$

$$C^k_{0r}f^k_z(z\,|H_r)p^k_r - C^k_{r0}f^k_z(z\,|H_0)p^k_0 \;\overset{H_r}{\underset{H_0}{\gtrless}}\; (C^k_{r1} - C^k_{01})f^k_z(z\,|H_1)p^k_1\,,$$

$$C^k_{1r}f^k_z(z\,|H_r)p^k_r - C^k_{r1}f^k_z(z\,|H_1)P^k_1 \;\overset{H_r}{\underset{H_1}{\gtrless}}\; (C^k_{r0} - C^k_{10})f^k_z(z\,|H_0)P^k_0\,.$$

Moving all the terms to left side of the inequalities, we get

$$p_1\, f^k_z(z\,|H_1) - p_0\, f^k_z(z\,|H_0) \;\overset{H_1}{\underset{H_0}{\gtrless}}\; 0\,, \tag{34}$$

$$C^k_{0r}f^k_z(z\,|H_r)p^k_r - C^k_{r0}f^k_z(z\,|H_0)p^k_0 + (C^k_{01} - C^k_{r1})f^k_z(z\,|H_1)p^k_1 \;\overset{H_r}{\underset{H_0}{\gtrless}}\; 0\,, \tag{35}$$

$$C^k_{1r}f^k_z(z\,|H_r)p^k_r - C^k_{r1}f^k_z(z\,|H_1)P^k_1 + (C^k_{10} - C^k_{r0})f^k_z(z\,|H_0)P^k_0 \;\overset{H_r}{\underset{H_1}{\gtrless}}\; 0\,. \tag{36}$$

For simplicity, let us define the following three functions:

$$\Gamma_1^k(z) = p_1^k \, f_z^k(z \mid H_1) - p_0^k \, f_z^k(z \mid H_0) \tag{37}$$

$$\Gamma_2^k(z) = C_{0r}^k f_z^k(z \mid H_r) p_r^k - C_{r0}^k f_z^k(z \mid H_0) p_0^k + (C_{01}^k - C_{r1}^k) f_z^k(z \mid H_1) p_1^k \tag{38}$$

$$\Gamma_3^k(z) = C_{1r}^k f_z^k(z \mid H_r) p_r^k - C_{r1}^k f_z^k(z \mid H_1) P_1^k + (C_{10}^k - C_{r0}^k) f_z^k(z \mid H_0) P_0^k \tag{39}$$

The inequalities (34), *(35)* and *(36)* can be written simply as

$$\Gamma_1^k(z) \underset{H_0}{\overset{H_1}{\gtrless}} 0, \tag{40}$$

$$\Gamma_2^k(z) \underset{H_0}{\overset{H_r}{\gtrless}} 0, \tag{41}$$

$$\Gamma_3^k(z) \underset{H_1}{\overset{H_r}{\gtrless}} 0. \tag{42}$$

Hence the decision rule of (18) becomes

$$\zeta^k(z) = \begin{cases} \text{if } \Gamma_1^k(z) \;\&\; \Gamma_2^k(z) < 0 & \textit{choose } H_0 \\ \text{if } \Gamma_1^k(z) > 0 \;\&\; \Gamma_3^k(z) < 0 & \textit{choose } H_1 \\ \textit{otherwise} & \textit{choose } H_r \end{cases} \tag{43}$$

From (43), if we had the three Γ_0^k, Γ_1^k and Γ_r^k functions we could compute regions on the z axis for every output bit and for every hypothesis such that the expected loss would be minimal. To do so we need to have all the conditional probability density functions (ie. $f_z^k(z \mid H_i)$) as well as all the a priori probabilities p_i^k required in (37), *(38)* and in *(39)*. These probabilities are different for every output bit, and need to be computed for every bit separately.

Estimation of the Conditional Density Functions ($f_z^k(z \mid H_i)$):

There are two general approaches to density estimation, ***parametric*** and ***nonparametric*** [10]. If we can assume we have a density function that can be characterized by a set of parameters, we can design a classifier using estimates of the parameters. Unfortunately, we often can not assume a parametric form for the density function, and in order to perform the test in *(43)* we have to estimate the conditional probability density functions using a different and not so structured approach called ***nonparametric estimation***. Since, in nonparametric approach, the density function is estimated locally by a small number of neighboring samples, the estimation is less reliable with larger bias and variance than the parametric counterpart.

The two main nonparametric estimation techniques are: the ***Parzen density estimate*** [10] and the ***k-nearest neighbor density estimate***(kNN) [10]. They are fundamentally very similar, but exhibit some different statistical properties. The kNN approach can be interpreted as the Parzen approach with a uniform kernel function whose size is adjusted automatically, depending on the location. We have decided to use the Parzen approach since a Gaussian distribution function instead of the uniform kernel can be used, which in practice gives a smoother estimate.

It is extremely difficult to obtain an accurate density estimate nonparametrically, particularly in high-dimensional spaces. But since we are performing bitwise analysis, all our density functions are in a one dimensional space stretching only from 0 to 1 (since output of all neurons are between 0 and 1). Because the number of training patterns are limited, this method has higher accuracy of estimation compared to the multidimensional density estimation.

Now let us consider a random variable Z and its probability density function p(z). In order to estimate the value of the density function at a point z, we may set up a small local region around z, L(z). Then, the probability coverage (or probability *mass)* of L(z) may be approximated by $p(z)v$, where v is the length if L(z). This probability may be estimated by drawing a large number of samples, N, from p(z), containing the number of samples, m, falling in L(z), and computing m/N. Equating these two probabilities, we may obtain an estimate of the density function as

$$\hat{p}(z)v = \frac{m(z)}{N} \quad or \quad \hat{p}(z) = \frac{m(z)}{Nv} \tag{44}$$

Note that, with a fixed v, m is a random variable and is dependent on z. A fixed v does not imply the same v throughout the entire space, and v could still vary with z. However, v is a preset value and is not a random variable.

Kernel expression: The estimate of (44) has another interpretation. Suppose that three samples, z_3, z_4, and z_5, are found in L(z) as shown in Figure 3.8. With v and N given, $\hat{p}(z)$ becomes $\frac{3}{Nv}$. On the other hand, if we setup a uniform kernel *function,* $\kappa(.)$, with

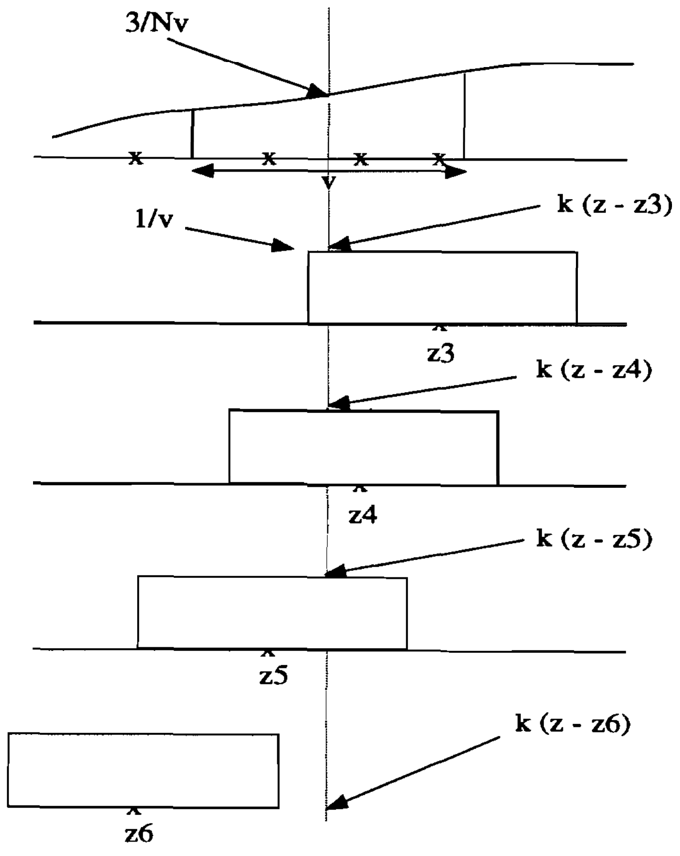


Figure 3.8 Parzen Density Estimation.

length v and magnitude of $\frac{1}{v}$ around all existing samples, the average of the values of these kernel functions at z is also $\frac{3}{Nv}$. That is

$$\hat{p}(z) = \frac{1}{N}\sum_{i=1}^{N}\kappa(z - z_i) \qquad (45)$$

As seen in Figure 3.8, only the kernel functions around the three samples, z_3, z_4, and z_5, contribute to the summation of (45).

Once (45) is adopted, the shape of the kernel function could be selected more freely, under the condition $\int \kappa(z)\,dz = 1$. For one-dimensional cases such as ours, we may seek optimally and select a complex shape. However to keep computations simple and yet to be accurate enough, we have chosen a normal kernel with the mean of zero ($\mu_z = 0$) for all the experiments:

$$\kappa(z) = \frac{1}{\sqrt{2\pi}\,\sigma_z} e^{-\frac{z^2}{2\sigma_z^2}} \tag{46}$$

Convolution expression: Equation **(45)** can be rewritten in convolution form as

$$\hat{p}(z) = \hat{p}_s(z) * \kappa(z) \equiv \int \hat{p}_s(\Upsilon)\,\kappa(z-\Upsilon)\,d\Upsilon \tag{47}$$

where $\hat{p}_s$ is an impulse density function with impulses at the **locations** of existing N **samples.**

$$\hat{p}_s(\Upsilon) = \frac{1}{N}\sum_{i=1}^{N}\delta(\Upsilon - z_i) \tag{48}$$

That is, the estimated density $\hat{p}(z)$ is obtained by feeding $\hat{p}_s(z)$ through a linear (noncausal) filter whose impulse response is given by $\kappa(z)$. **Therefore,** $\hat{p}(z)$ is a smoothed version of $\hat{p}_s$**(z).**

Moments of $\hat{p}(z)$: The first and second order moments of **(47)** can be easily computed. First, let us compute the expected values of $\hat{p}_s(z)$ as

$$E\left\{\hat{p}_s(z)\right\} = \frac{1}{N}\sum_{i=1}^{N}\int\delta(z-\xi)p(\xi)d\xi = \frac{1}{N}\sum_{i=1}^{N}p(z) = p(z) \tag{49}$$

That is, $\hat{p}_s(z)$ is an unbiased estimate of p(z). Then, the expected value of $\hat{p}(z)$ of (47) may be computed as

$$E\left\{\hat{p}(z)\right\} = \int E\left\{\hat{p}_s(Y)\right\} \kappa(z - Y)\, dY = \int p(Y)\kappa(z - Y)\, dY = p(z) * \kappa(z) \quad (50)$$

Also,

$$E\left\{\hat{p}^2(z)\right\} = \frac{1}{N^2}\left[\sum_{i=1}^{N}\int \kappa^2(z-\xi)p(\xi)d\xi + \sum_{i=1_{i\neq j}}^{N}\sum_{j=1}^{N}\iint \kappa(z-Y)\kappa(z-\xi)p(Y)p(\xi)dY\, d\xi\right] \quad (51)$$

$$= \frac{1}{N}p(z) * \kappa^2(z) + (1 - \frac{1}{N})\left[p(z) * \kappa(z)\right]^2 \quad (52)$$

Therefore, the variance of $\hat{p}(z)$ is

$$Var\left\{\hat{p}(z)\right\} = \frac{1}{N}\left[p(z) * \kappa^2(z) - \left[p(z) * \kappa(z)\right]^2\right] \quad (53)$$

Even though we only need to estimate $\hat{f}^k(z\,|H_i)$, for $i \varepsilon \{0, 1, r\}$, we have also computed $m_{z^k} = E\left\{z^k\,|H_i\right\}$ and $\sigma^2_{z^k} = var\left\{z^k\,|H_i\right\}$ as well for future analysis of output bits.

For every bit k, we use the following procedure to estimate $\hat{f}^k(z\,|H_0)$:

Consider the training set $\Omega = \left\{X_1, X_2, \ldots, X_N\right\}$ with N data samples.

1. Find the set Ω_0^k of data samples in Ω which have a desired output value of zero for

bit k: $\Omega_0^k = \left\{ X_1 , X_2 , \ldots , X_{M_0} \right\}$ with M_0 samples.

2. Find the subset Ω_{00}^k of Ω_0^k for which the actual output value at bit k is less than 0.5 ($z^k < 0.5$): $\Omega_{00}^k = \left\{ X \mid z_x^k < 0.5 \right\} = \left\{ X_1 , X_2 , \ldots , X_{r_0} \right\}$ with r_0 samples.

3. For the set Ω_{00}^k, we build a corresponding output set Ξ_{00}^k which contains all the output values for bit k for input samples of Ω_{00}^k: $\Xi_{00}^k = \left\{ z \mid X_z \varepsilon\, \Omega_{00}^k \right\} = \left\{ z_1 , z_2 , \ldots , z_{r_0} \right\}$

4. Form a normal kernel around each $z_i \varepsilon\, \Xi_{00}^k$:

$$\kappa_i(z - z_i) = \frac{\alpha_i}{\sqrt{2\pi}\,\sigma_i} e^{-\frac{(z-z_i)^2}{2\sigma_i^2}} \left[U(z) - U(z-1) \right] \tag{54}$$

Where U(z) and U(z-1) are unit step functions. They are used to limit the probability density function to the interval from 0 to 1. α_i is a constant calculated by

$$\alpha_i = \frac{\sqrt{2\pi}\,\sigma_i}{\int_0^1 e^{-\frac{(z-z_i)^2}{2\sigma_i^2}} dz} \tag{55}$$

It compensates for the fact that the pdf is only valid over the interval [0 , 1] instead of $(-\infty , +\infty)$.

5. Use (47) to form an estimate

$$\hat{f}^k(z\,|H_0) = \frac{1}{r_0}\sum_{i=1}^{r_0} \kappa_i(z - z_i) = \frac{1}{r_0}\sum_{i=1}^{r_0} \frac{\alpha_i}{\sqrt{2\pi}\,\sigma_i} e^{-\frac{(z-z_i)^2}{2\sigma_i^2}} \left[U(z) - U(z-1)\right] \quad (56)$$

The above procedure is the same for estimating $\hat{f}^k(z\,|H_1)$ and $\hat{f}^k(z\,|H_r)$ except for steps 1 and 2. To estimate $\hat{f}^k(z\,|\mathbf{H}_1)$, steps 1 and 2 change to:

1. Find the set Ω_1^k of data samples in Ω which have a desired output value of one for bit **k** $\Omega_0^k = \left\{X_1, X_2, \ldots, X_{M_1}\right\}$ with M_1 samples.

2. Find the subset Ω_{11}^k of Ω_1^k for which the actual output value at bit **k** is greater than 0.5 ($z^k > 0.5$): $\Omega_{11}^k = \left\{X\,|\,z_x^k{>}0.5\right\} = \left\{X_1, X_2, \ldots, X_{r_1}\right\}$.

For $\hat{f}^k(z\,|H_r)$, step 1 is not performed and step 2 is as follows:

2. Find the subset of Ω_0^k for which the actual output value at bit **k** is greater than 0.5 and find the subset of Ω_1^k for which the actual output value at bit **k** is less than 0.5. Take the union of the two subsets to get Ω_{rr}^k:

$$\Omega_{rr}^k = \left\{X\,|\,(X\varepsilon\Omega_0^k \ \&\ z_x^k{>}0.5)\ or\ (X\varepsilon\Omega_1^k \ \&\ z_x^k{<}0.5)\right\} = \left\{X_1, X_2, \ldots, X_{r_r}\right\}, \quad (57)$$

where for every bit **k**. r_0, r_1, r_r satisfy

$$r_0 + r_1 + r_r = N . \qquad (58)$$

Estimation of the a priori probabilities p_i^k: The estimation of the a priori probabilities is much simpler and can be computed by the following simple equations:

$$\hat{p}_0^k = \frac{r_0^k}{N} \quad \hat{p}_1^k = \frac{r_1^k}{N} \quad \hat{p}_r^k = \frac{r_r^k}{N} . \qquad (59)$$

Cost of error (C_{ij}^k): Though it is possible to have different cost criterions for different bits, we decided to have one criterion for all bits. Then, C_{ij}^k, simplifies to C_{ij}. There are several conditions in our criterion which were mentioned before:

1. $C_{ii} = 0$ Normally the cost of guessing the correct hypothesis is zero.
2. $C_{r0} = C_{r1}$ The costs of rejecting an output when it should have been classified 0 or 1, are the same.
3. $C_{0r} = C_{1r}$ The costs of chosing H_0 or H_1 when H_r should have been chosen, are equal.
4. $C_{01} = C_{10}$ The cost of chosing H_0 when H_1 was true, and the cost of chosing H_1 when H_0 was true are equal.

There are two more relational conditions which should be mentioned here:

5. $C_{r0} = C_{r1} < C_{0r} = C_{1r}$ The consequences of classifying H_0 or H_1 as H_r is less severe than classifying H_r as H_0 or H_1. (Rejected information still has a chance of being classified correctly in the next module.)

6. $C_{01} = C_{10}$ w $C_{0r} = C_{1r} > C_{r0} = C_{r1}$ The consequences of classifying H_0 as H_1 or reverse is much higher than that of any other error.

In our research we experimented primarily with $\left\{ C_{r0} = C_{r1} = 1\ ,\ C_{0r} = C_{1r} = 2\ ,\ C_{01} = C_{10} = 5 \right\}$ and sometimes with $\left\{ C_{r0} = C_{r1} = 1\ ,\ C_{0r} = C_{1r} = 2\ ,\ C_{01} = C_{10} = 10 \right\}$. The results were similar, except the fact that the second criterion makes reject region to slightly grow and zero and one regions to slightly shrink.

Now using the above a posteriory and a priori estimates in (37), (38), and *(39)* we can estimate $\Gamma_1^k(z)$, $\Gamma_2^k(z)$, and $\Gamma_3^k(z)$. Using these estimates in (43), we can decide on one of the three hypotheses H_0, H_1, or H_r.

This procedure is performed for every output bit. The decision for every bit is then sent to the vector rejector which in turn decides whether to reject the input pattern and send it to the next stage or accept it and send it to the nearest neighbor classifier for classification.

The decision rule of *(43)* is carried out by performing the following:

For test *1*, set $\Gamma_1^k(z) = 0$, and use *(37)* to find

$$z_{01}^k = \Gamma_1^{k^{-1}}(0) \qquad (60)$$

Thus dividing the interval $\iota = \left[0 \cdots 1 \right]$, into two subintervals, $\iota_1^{k0} = \left[0 \cdots z_{01}^k \right]$ the interval for H_0 for test 1 of bit k, and $\iota_1^{k1} = \left[z_{01}^k \cdots 1 \right]$ the interval for H_1 for test 1 of bit k.

In the same manner we compute z_{0r}^k and z_{r1}^k, from test 2 and 3, using (38) and (39). Although in theory it is possible for each test to divide the interval ι into several subintervals, in practice, in all our experiments, ι is divided only into two sub-intervals by each test (ie. $\Gamma_1^k(z)$, $\Gamma_2^k(z)$, and $\Gamma_3^k(z)$ have only one root each). Figure 3.9 shows a typical outcome of the three tests. Namely

$$0 < z_{0r}^k < z_{01}^k < z_{r1}^k < 1 \qquad (61)$$

The decision strategy governed by (43) corresponds to a voting strategy among the three tests. For output value z, when two of the three tests are in agreement, that decision is accepted. If no tests agree, the decision is reject, and that bit is rejected. For example, assuming the order shown in Figure 3.9, if the output value of bit k falls in the interval $\left[0 , z_{0r}^k \right]$ (tests 1 and 3 agree on H_0), the bit is classified as zero, if the output value falls in $\left[z_{r1}^k , 1 \right]$ (tests 1 and 2 agree on H_1), the bit is classified as one, and finally, if it falls in $\left[z_{0r}^k , z_{r1}^k \right]$ (tests 2 and 3 agree on H_r), that bit is rejected.

It is also possible that the order in (61) not hold. A current working hypothesis is that, any network that defies the order of (61), is either severely under-trained or is not large enough to handle the complexity of the problem. If this is proven to be a correct hypothesis, then one can have an idea as to how the size of the network matches up with the complexity of the problem, early in training procedure. This can avoid further training of a network that can not handle the complexity of the problem for purely topological reasons.

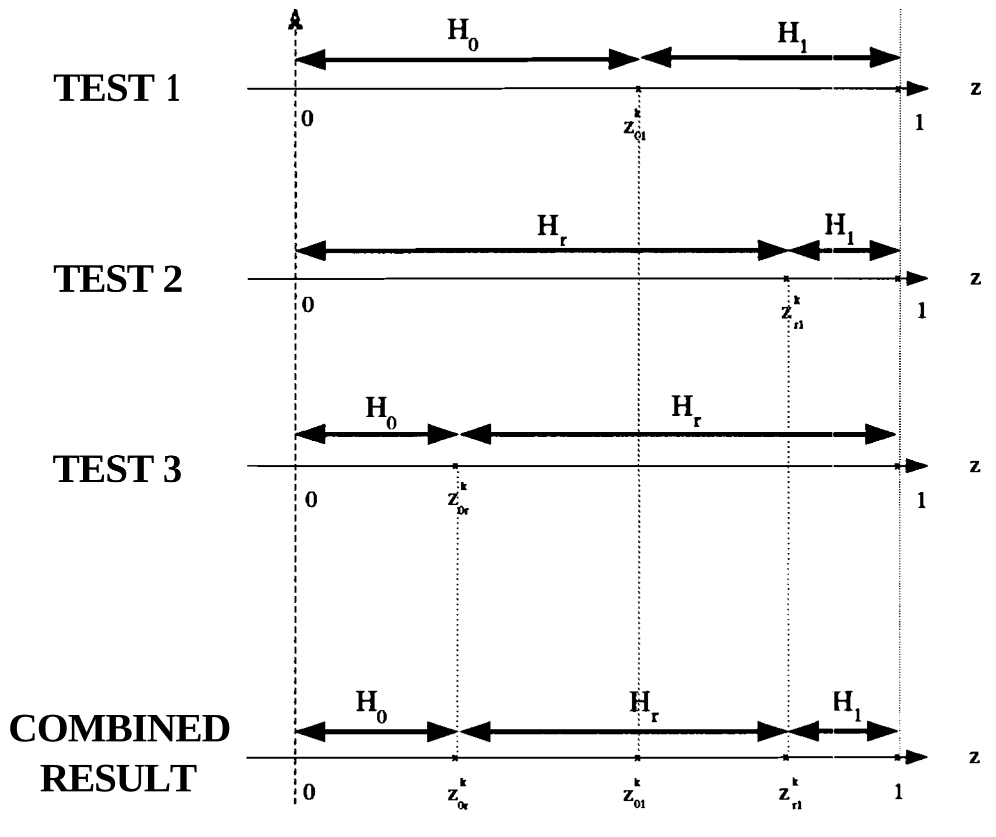


Figure 3.9 Sample Rejection Boundaries.

In the above discussion and in (61), it is assumed that the equations

$$\Gamma^k_1(z) = 0 \quad \Gamma^k_2(z) = 0 \quad \Gamma^k_3(z) = 0 \tag{62}$$

have only one root. The expected behavior of $f_z(z \,|H_0)$, $f_z(z \,|H_1)$, and $f_z(z \,|H_r)$ are shown in Figure 3.10. In order for equations in (62), have one root, the following conditions must be satisfied (These conditions assume probability behavior as shown in

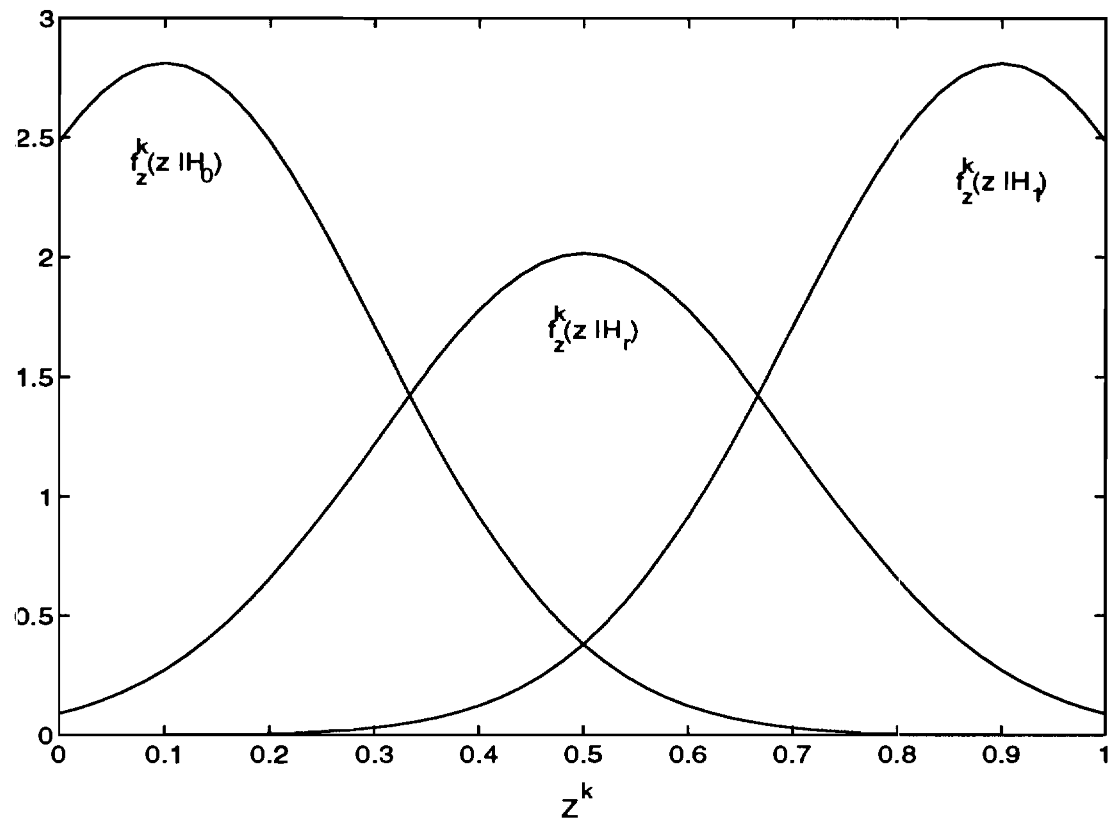


Figure 3.10 Expected Conditional Density Functions.

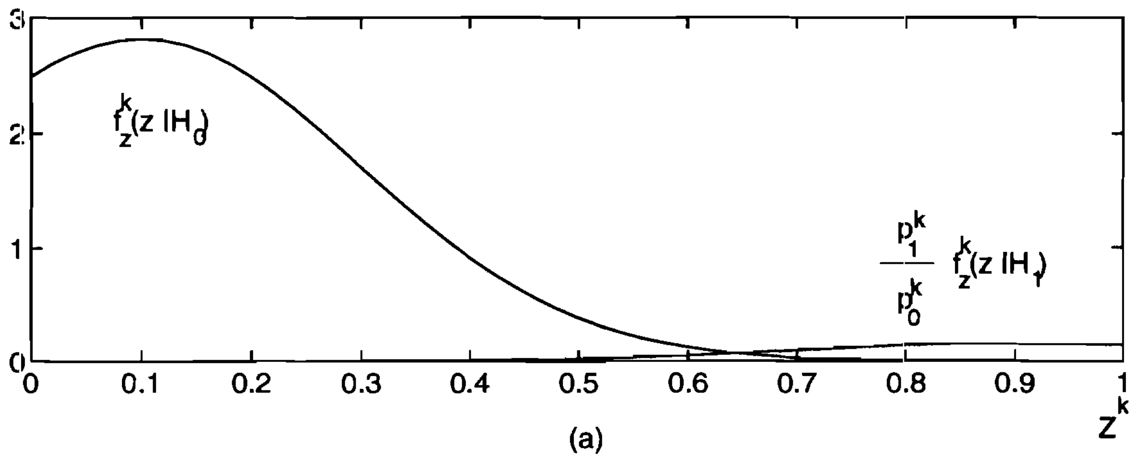
Figure 3.10) :

From Test 1 we get two conditions (see Figures 3.11 b and 3.11 a),

$$f_z^k(0|H_0) > \frac{p_1^k}{p_0^k} f_z^k(0|H_1)\,, \tag{63}$$

$$f_z^k(1|H_1) > \frac{p_0^k}{p_1^k} f_z^k(1|H_0)\,. \tag{64}$$

From Test 2 we get (see Figures 3.12 b and 3.12 a)



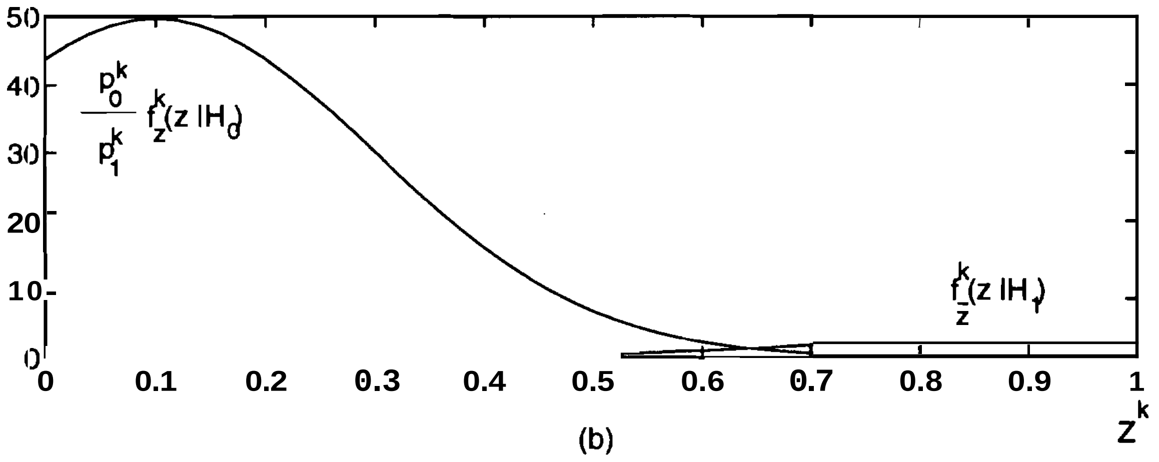


Figure 3.11 Density Functions for Test I.

$$f_z^k(0\,|\,H_0) > \frac{C_{0r}\,p_r^k\,f_z^k(0\,|\,H_r) + (C_{01} - C_{r1})p_1^k\,f_z^k(0\,|\,H_1)}{C_{r0}\,p_0^k}, \tag{65}$$

$$f_z^k(1\,|\,H_1) > \frac{C_{0r}\,p_r^k\,f_z^k(1\,|\,H_r) - C_{r0}\,p_0^k\,f_z^k(1\,|\,H_0)}{(C_{r1} - C_{01})p_1^k} \tag{66}$$

From Test 3 we get (see Figures 3.13 a and 13 b),

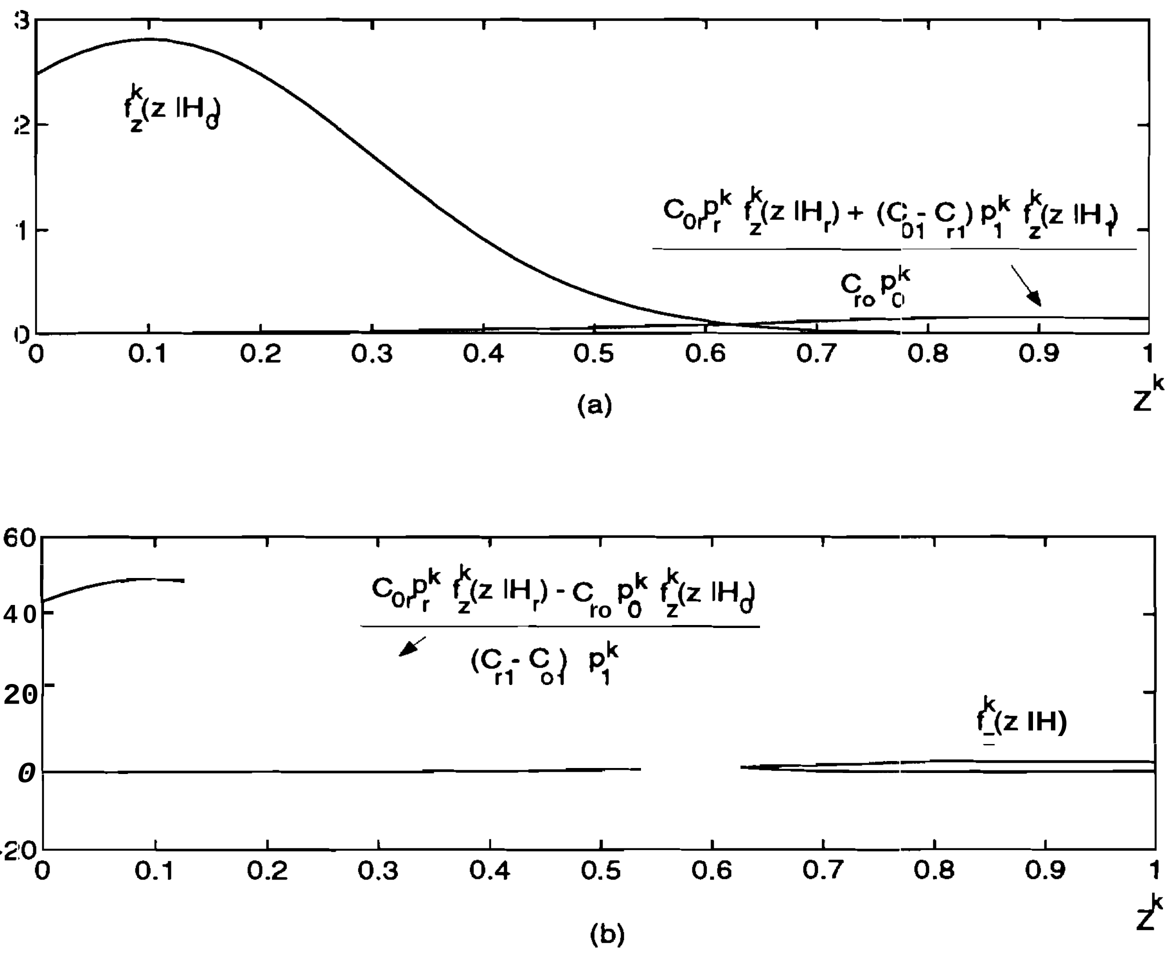


Figure 3.12 Density Functions for Test 2.

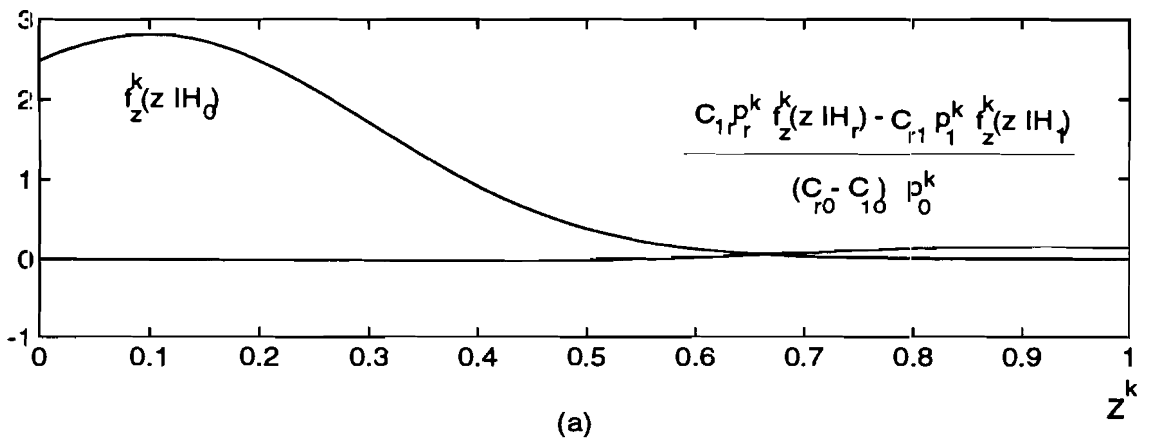
$$f_z^k(0\,|\,H_0) > \frac{C_{1r}\,p_r^k\,f_z^k(0\,|\,H_r) - C_{r1}\,p_1^k\,f_z^k(0\,|\,H_1)}{(C_{r0} - C_{10})p_0^k}\,, \tag{67}$$

$$f_z^k(1\,|\,H_1) > \frac{C_{1r}\,p_r^k\,f_z^k(1\,|\,H_r) + (C_{10} - C_{r0})p_0^k\,f_z^k(1\,|\,H_0)}{C_{r1}\,p_1^k} \tag{68}$$

Figures 3.11 through 3.13 are the actual probabilities from one of our experiments with the following *cost* values:

$$C_{10} = C_{01} = 5\,,\quad C_{1r} = C_{0r} = 2\,,\quad C_{0r} = C_{r1} = 1\,.$$

The conditions of (63-68) were satisfied in all experiments.



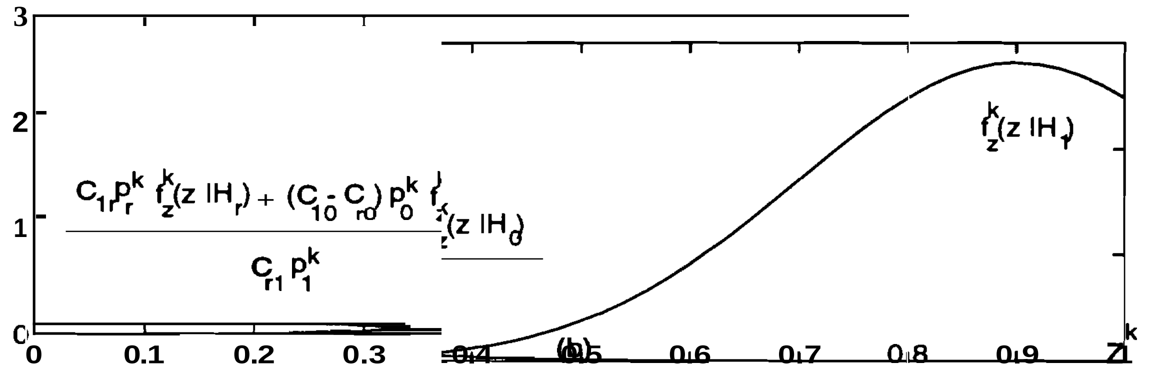


Figure 3.13 Density Functions for Test 3.

Side property: The same procedure can be used to estimate the threshold of each output neuron in parallel (in which case we would only have H_0 and H_1):

Let y^k be the sum of weighted input activation levels for output neuron k,

$$y^k = \sum_{i=1}^{n_h} W_{ki}\, z_h^i \,, \tag{69}$$

where W_{ki} is the weight connecting the ith hidden neuron to the kth output neuron, z_h^i is the output (activation level) of i^{th} neuron of the previous level. Therefore the activation of the kth output neuron is,

$$z^k = \frac{1}{1 + e^{-(y^k + \theta^k)}} \,. \tag{70}$$

By estimating $f_y^k(y\,|H_0)$, and $f_y^k(y\,|H_1)$ using the density estimations similar to the procedure above, we can estimate the threshold (θ_{01}^k) using:

$$\Gamma_y^k(\theta_{01}^k) = P_1^k(\theta_{01}^k)\, f_y^k(\theta_{01}^k\,|H_1) - P_0^k(\theta_{01}^k)\, f_y^k(\theta_{01}^k\,|H_0) = 0\,, \tag{71}$$

or

$$\theta_{01}^k = \left[\Gamma_y^k\right]^{-1}(0)\,. \tag{72}$$

Remarks:

- The procedure described above is parallel in nature and can be performed for all the output bits at the same time. Since the steps performed in parallel are the same, the above procedure is ideal for an ***SIMD (Single Instruction Multiple Data)*** [11-12] machine such as MasPar MP-1 (see Chapter 5).

- There are some fundamental and philosophical differences between the system described here and other probabilistic networks. The procedure above looks at the problem of classification using neural networks from a different point of view as follows:

1. A fundamental difference between this method and other probabilistic neural networks is that others estimate $P(H_i | X)$, where X is the input vector to the network and $i \in \left\{ 1,2,...,n_o \right\}$ where n_o is the number of output bits. This means estimating the probability of hypothesis i being true given the input vector of X.

   Our procedure discussed above is estimating $f_z^k(z | H_i)$, where z is the output value of bit k. Use of Bayes rule (12) then allows the estimation of $P^k(H_i | z)$ $(i \in \left\{0,1,r\right\})$.

   In other probabilistic networks, using Bayes rule (12) yields

$$P(H_i | X) = \frac{f_x(X | H_i) P(H_i)}{f_x(X)}, \tag{73}$$

   The estimation is not a single bit estimation, but rather a hypothesis estimation using vector estimation. It is well known that high dimensionality is the main source of inaccuracy in classification problems. As an example, consider the one dimensional case in which 1000 training patterns are available between 0 and 1. 1000 samples distributed in the interval [0 , 1] gives a an accurate estimation of the probability density function. Now consider a 7 dimensional space with the same number of training patterns available in the unit hypercube at the origin. The data points will be so sparse that an accurate estimation of the

7-dimensional probability density function will be almost impossible.

This is the case with the 10-class Colorado problem. There are 1188 training patterns available in a 7-dimensional space belonging to 10 classes. It is clear that the accuracy of the single dimensional estimation will be: much higher than that of the 7-dimensional one. This is of course assuming that all 1188 data patterns are made available to all single dimension estimators. This is the case in our procedure.

2. A second important difference is that other probabilistic networks force a classification even for data which in their current format are impossible or very difficult to classify. We, on the other hand recognize the fact that this kind of data can form a class which is to be rejected. The network is organized so that it recognizes data belonging to this class, classifies it as such and sends it to the next stage for preprocessing (possibly including change of format, using a different non-linear coding scheme, etc.) and another attempt at classification (see Figure 3.2 and 3.3).

   This rejection classification is first performed at the bit level by bit classifiers, and then the information is combined in the vector rejector (optional) for vector rejection (see section 3.3.2).

   In other words, other networks try to divide the classification space shown in Figure 3.7 into two regions, one for each class. However, the system described here will divide the space into three regions, an additional region for the class of data which are difficult to classify. Data falling in this region are sent to the next module. In that module, this region is nonlinearly transformed (optional) and the process is repeated. This divide and conquer procedure continues until we reach a desired accuracy.

Comparative analysis:

In order to better point out the fundamental differences between PPSHNN and other probabilistic networks, we would like to briefly describe and and conduct a comparative study of Donald Specht's recently published [5] ***Probabilistic Neural Networks*** (PNN).

Figure 3.14 shows Specht's neural network organization for classification of input patterns X into two possible categories, A, and B.

In Figure 3.14, the input units are merely distribution units that supply the same voltage values to all of the pattern units. The pattern units (shown in Figure 3.15) each form a dot product of the input pattern vector X with a weight vector W_i, $Y_i = X \cdot W_i$, and then perform a nonlinear operation on Y_i before outputting their activation level to the summation unit. Instead of the Sigmoid activation function commonly used for backpropagation, the nonlinear operation used here is

$$f_Y(y) = e^{\frac{(y-1)}{\sigma^2}}$$

Assuming that both X and W_i are normalized to unit length, this is equivalent to using

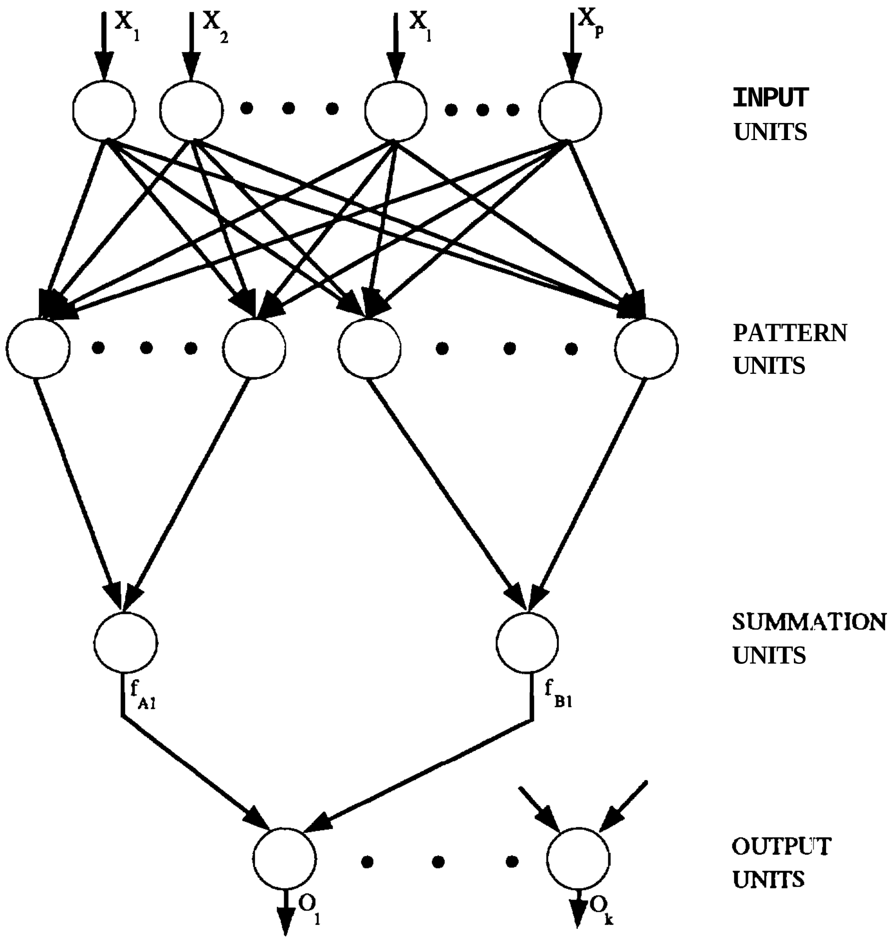


Figure 3.14 Specht's Network.

$$e^{-\frac{(W_i - X)^t (W_i - X)}{2\sigma^2}} \qquad (74)$$

The summation units simply sum the inputs from the pattern units that correspond to the category from which the training pattern was selected.

The output, or decision, units are two-input neurons, as shown in Figure 3.16. These units produce binary outputs and have only a single variable weight C_k where

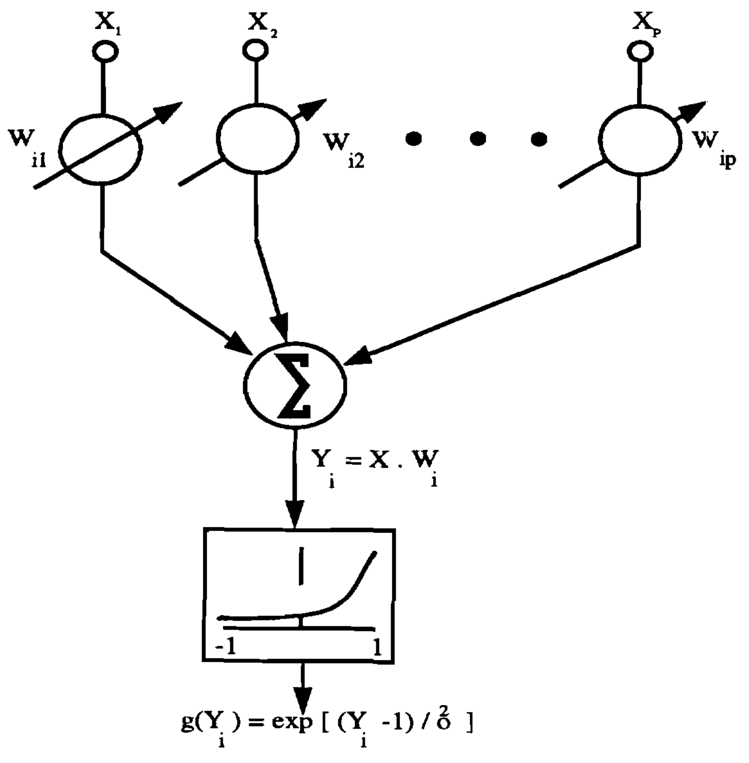


Figure 3.15 Pattern Units of Specht's Network.

$$C_k = -\frac{h_{Bk} l_{Bk}}{h_{Ak} l_{Ak}} \cdot \frac{n_{Ak}}{n_{Bk}} . \qquad (75)$$

h_{Ak} is the a priori probability of occurrence of patterns from category A for output neuron k, h_{Bk} is the a priori probability of occurrence of patterns from category B for output neuron k, l_{Ak} is the loss associated with the decision "X belongs to class A for output neuron k", l_{Bk} is the loss associated with the decision "X belongs to class B for output neuron k", n_{Ak} is the number of training patterns from category A for output neuron k, and n_{Bk} is the number of training patterns from category B for output neuron k.

Note that C_k is the ratio of a priori probabilities, divided by the ratio of samples, and

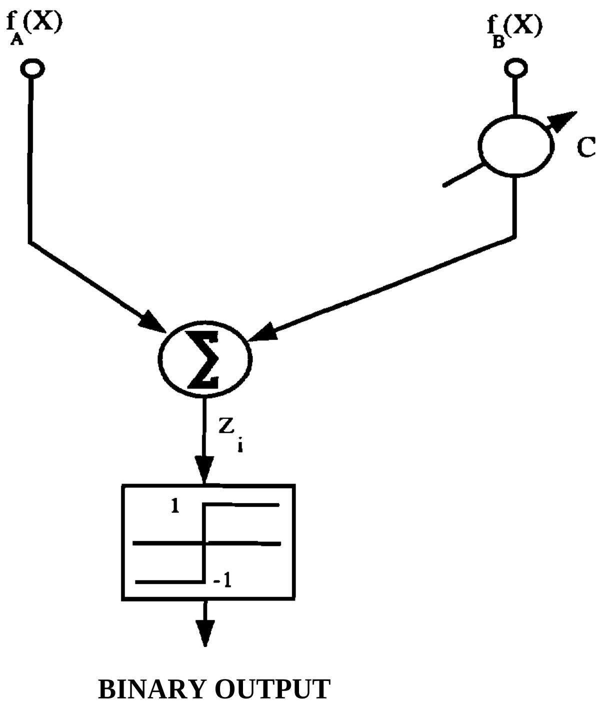


Figure 3.16 Output Neuron of Specht's Network.

multiplied by the ratio of losses. In any problem in which the numbers of training samples from categories A and B are obtained in proportion to their a priori probabilities, $C_k = -\frac{l_{Bk}}{l_{Ak}}$. This final ratio cannot be determined from the statistics of the training samples, but only from the significance of the decision. If there is no particular reason for biasing the decision, C_k may simplify to -1 (an inverter).

The network is trained by setting the W_i weight vector in one of the pattern units equal to each X pattern in the training set, and then connecting the outputs of the pattern units to the appropriate summation unit. A separate neuron (pattern unit) is required for every training pattern. As indicated in Figure 3.14, the same pattern units can be grouped by different summation units to provide additional pairs of categories and. additional bits of information in the output vector.

In other words, the pattern units in Specht's network form a normal distribution around their respective W_i. This means that the pattern layer builds a normal distribution function around each pattern of training set (X^i). Then the summation units, by adding up these distribution functions for each class, form a global distribution function for each class. Therefore, every incoming pattern X is compared to these global distribution functions and, according to Bayes ***minimum risk*** criterion, a class for X is chosen. This final step is performed in the output unit.

Comparison: There are several issues that have to be addressed in order to point out the important differences between the ***PPSHNN*** and Specht's or any other probabilistic neural networks:

1. Estimation Accuracy: Specht uses Parzen density estimation to estimate $P(H_i|X)$, where X is the input pattern and H_i is the hypothesis that X belongs to class i.

   It is well known that non-parametric methods become exceedingly difficult and inaccurate as the dimensionality of problem space increases. In most real world problems such as the speech synthesis problem (X is a vector in 70 dimensional space, see section 2.1.1), or the remote sensing problem (X belongs to a 7 dimensional space, see section 2.1.2), Specht's network tends to estimate the distribution functions inaccurately, and as a result, decreases chances of correct classification.

On the other hand, in PPSHNN, distribution estimates are always performed at the bit level, in other words, always in a one dimensional space, resulting in improved estimation and classification accuracy.

2. Training data: Let us assume we have n pieces of data for the one dimensional case. In order to have a comparable estimation accuracy in the p-dimensional space, Parzen's or any non-parametric method requires on the order of n^p sample points. Normally this many pieces of sample data does not exist; therefore a reasonable multi-dimensional estimate in problems which have limited number of sample data is quite difficult.

3. Training and Testing time: A big advantage of Specht's PNN is the short period of time required for training, as compared to backpropagation networks. But with SIMD_PPSHNN (see chapter 5) running on MasPar, this time has been cut in several orders of magnitude (see section 5.4), making the speed advantage of other networks negligible.

   The testing time, which is more crucial, is increased substantially by Specht's PNN. For every input vector X, PNN has to perform the inner product $< X , X^i >$ for all X^i's in training set. Because X and X^i belong to a high dimensional space as in the speech synthesis problem, and since, a large training set is needed to satisfy the requirements for high accuracy estimation in higher dimensional spaces, testing takes much longer than PPSHNN or a backpropagation network. If testing is not performed on a high performance machine, it cannot meet the real time requirement of most problems. On the other hand, PPSHNN or backpropagation networks are able to meet this requirement on almost any machine. An example such as the speech synthesis problem can clarify this further. Below, we compare the testing time complexity of the PNN and a backpropagation network in this problem.

Since most of the time a network requires is used to perform floating point additions and multiplications, counting the number of floating point addition and multiplication operations gives us an idea of the total time required by each network relative to each other:

k=1600	number of training patterns.
p=70	number of input neurons.
n_h=40	number of hidden neurons in the backpropagation network.
n_o=14	number of possible classes .

Specht's PNN:

Pattern units perform

$$pk \quad \textit{multiplications} \quad \text{and} \quad (p-1)k \quad \textit{additions.}$$

Summation units perform at least

$$0 \quad \textit{multiplications} \quad \text{and} \quad (k-n_o) \quad \textit{additions.}$$

And finally the output units require

$$\sum_{i=1}^{n_o-1} i = \frac{n_o(n_o-1)}{2} \quad \textit{multiplications} \quad \text{and} \quad \sum_{i=1}^{n_o-1} i = \frac{n_o(n_o-1)}{2} \quad \textit{additions.}$$

Thus Specht's PNN requires

$$pk+\frac{n_o(n_o-1)}{2} = 112091 \quad \textit{multiplications} \quad \text{and}$$

$$(p-1)k+(k-n_o)+\frac{n_o(n_o-1)}{2} = 112077 \quad \textit{additions.}$$

Backpropagation network:

Hidden neurons perform

$$pn_h \quad \textit{multiplications} \quad \text{and} \quad n_h(p-1) \quad \textit{additions.}$$

Output neurons perform

$$n_h n_o \quad \textit{multiplications} \quad \text{and} \quad n_o(n_h-1) \quad \textit{additions.}$$

Thus backpropagation only requires

$$n_h(p+n_o) = 3360 \quad \textit{multiplications} \quad \text{and}$$

$$n_h(p+n_o)-(n_h+n_o) = 3306 \quad \text{additions.}$$

If we assume that multiplication takes twice as much time as addition, we see that backpropagation is more than 33 times faster than Specht's PNN during testing:

$$\frac{2\times 112091 + 112077}{2\times 3360 + 3306} = 33.5 \,.$$

In addition, the inaccuracy in the estimation of distribution functions in a 70 dimensional space with only 1600 sample patterns should be a concern with the PNN network.

Therefore, one can train a PPSHNN or backpropagation network on a high performance machine, but use it on any machine in near real time for testing. Whereas, for Specht's PNN, one needs a high performance machine to use it for testing.

3.3.2 Vector Rejection

Vector rejection can be performed by a neural network. Such a network is trained to perform two-class classification (See Figure 3.17).

This network has $2n_o$ input neurons and 1 output neuron. The Vector Rejector (VR), receives two values from each Bit Rejector (BR). The first value is simply the output value of the corresponding output bit of the N-unit (z^k). The second value is the hypothesis (H_i^k), to which z^k has bin classified by the k-th BR. Note that H_i^k is 1, if z^k is

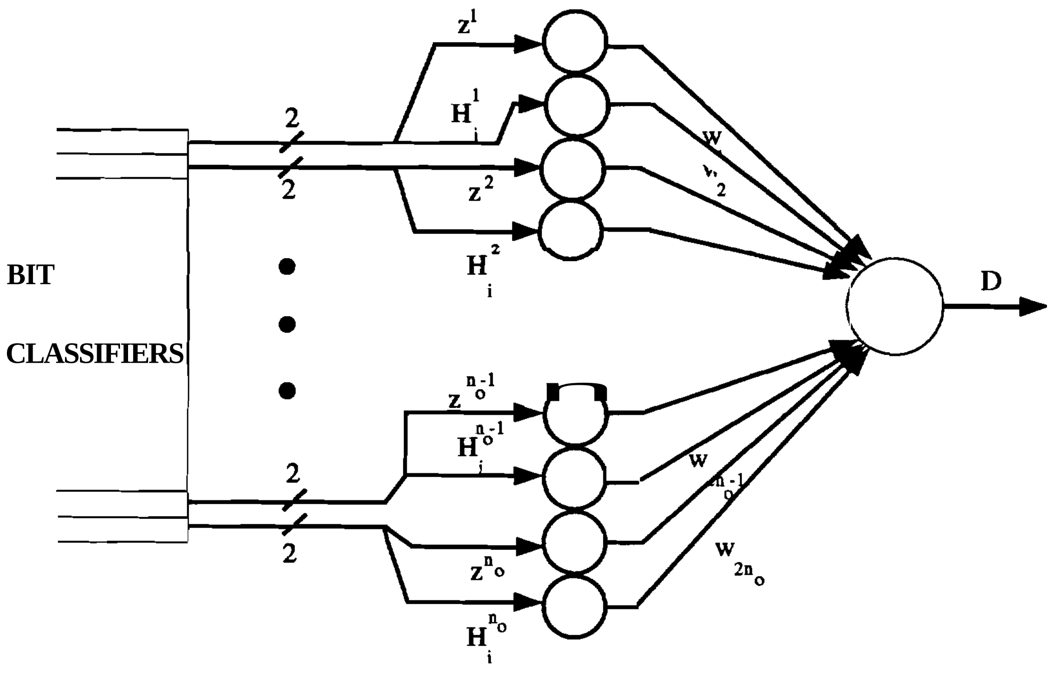


Figure 3.17 A Delta Rule Network as the Vector Rejector.

classified to H_1. It is 0 if z^k is classified to H_0, and it is 0.5 if z^k is classified to H_r. The output of VR is trained to go high for vectors that should be accepted. It is trained to go low for vectors which should be rejected and sent to the next stage for classification.

This network is trained with the output data which is gathered from the last training sweep of the N-unit and the bit-rejectors. Its desired output data is created by generating a 1 for all the input patterns for which the classification of the PPSHNN module was correct and their classification should be accepted, and a 0 for all the patterns whose classification by the PPSHNN module was incorrect or uncertain and should be rejected.

The vector rejector can be a single stage delta rule network. In some cases, the task of classifying vectors into accepted and rejected classes might be too complex for a single stage network to handle. In that case, the VR can be chosen as a two-stage network or a PNS network (see Chapter 4). As in all other modules, the VR can also be chosen as any special network such as a competitive learning network.

The bitwise classifiers, together with the vector rejector, address several problems and offer solutions for them as follows:

Most classifiers look at the entire vector and make the classification decision (eg. is a minimum distance classifier). By doing so, the classifier could overlook detailed information encoded in the individual bits which might be crucial for classification. The following example from the 10 class remote sensing problem using backpropagation is one such case:

Output vector =[0.53 0.62 0.40 0.32 0.67 0.37 0.32 0.45 0.47 0.351

A typical classifier such as a Bayesian vector classifier or any neural network classifier at the output would most probably classify this vector as class 5 or if the reject option is present as reject.

The output of the bitwise classifier is as follows:

[1 R R 0 R R 0 R R R].

The thresholds for the bit one classifier are as follows:

1.00000	***<-->***	***0.41987***	***Class 1***
0.41987	***<-->***	***0.30100***	***Rejected***

0.30100 <--> 0.00000 Class 0

In other words, bit one is classified as 1. Similarly, bits four and seven are classified as zero according to their thresholds. The rest of the bits are rejected again according to their thresholds. The problem with this data set is that some classes are very underrepresented during training, therefore making it difficult and unlikely for the network to learn them. In the 10-class Colorado problem, we have

# of total data pieces in the training set : 1188

# of data pieces for class 9 in the training set : 25

bit 9:

H1= 0 H0= 1163 Hr= 2.5

p1= 0.0000+00 p0= 9.7895-01 pr= 2.1043-02

We see that class 9 is very underrepresented (%2.1 of the training set) and its data are all rejected. Thus, any class 9 data in testing is going to be rejected. The problem is that this will also cause the 9th bit of some data from other classes to be rejected as well. If there are several such underrepresented classes, they will cause rejection of a vector belonging to another class, due to the uncertainty of the undertrained bits.

A bitwise classifier combined with a neural network vector rejector can detect these cases and allow exceptions. In the above mentioned case the vector rejector can learn to overlook the underrepresented bits when there is a definite classification for other bits, and correctly classify the above vector as class 1.

3.4 The Classifier (Minimum Euclidian Distance Classification unit)

This unit is a simple nearest neighbor classifier. It simply compares the incoming vector to desired vectors and finds the desired vector which is the closest to the incoming vector. The incoming vectors to this unit are the output vectors of the N-unit of the module which have not been rejected.

Let $V = \left[v_1 , v_2 , \cdots , v_n \right]$ be theincoming vector and $D_i = \left[d_{i1} , d_{i2} , \cdots , d_{in} \right]$ be the ith desired vector for $i = 1, \ldots , m$. The classification is according to

$$C = \min_i \left\{ \| D_i - V \| \right\}. \tag{76}$$

This unit is the final step in the classification process. The output of this unit is the number of a class to which the incoming pattern has been classified.

3.5 The Pre-Rejector (P-unit)

This unit as described in Section 3.1 is a two class classifier. It classifies the data belonging to the under- and unproportionally represented classes as "reject" and classifies the rest as "accept". In other words, it divides the problem space into two subregions and allows the N-unit to learn only the simpler region of the two.

This unit can be any type of neural network network. For example, it can be a single stage delta rule network. If this unit is a two stage network, because of it being only a two class classifier, it is normally much smaller than the N-unit of the corresponding module. For example, for the 10-class Colorado data, the pre-rejector of the first module (if chosen to be a two-stage backpropagation network) has only four hidden neurons (see Figure 3.18).

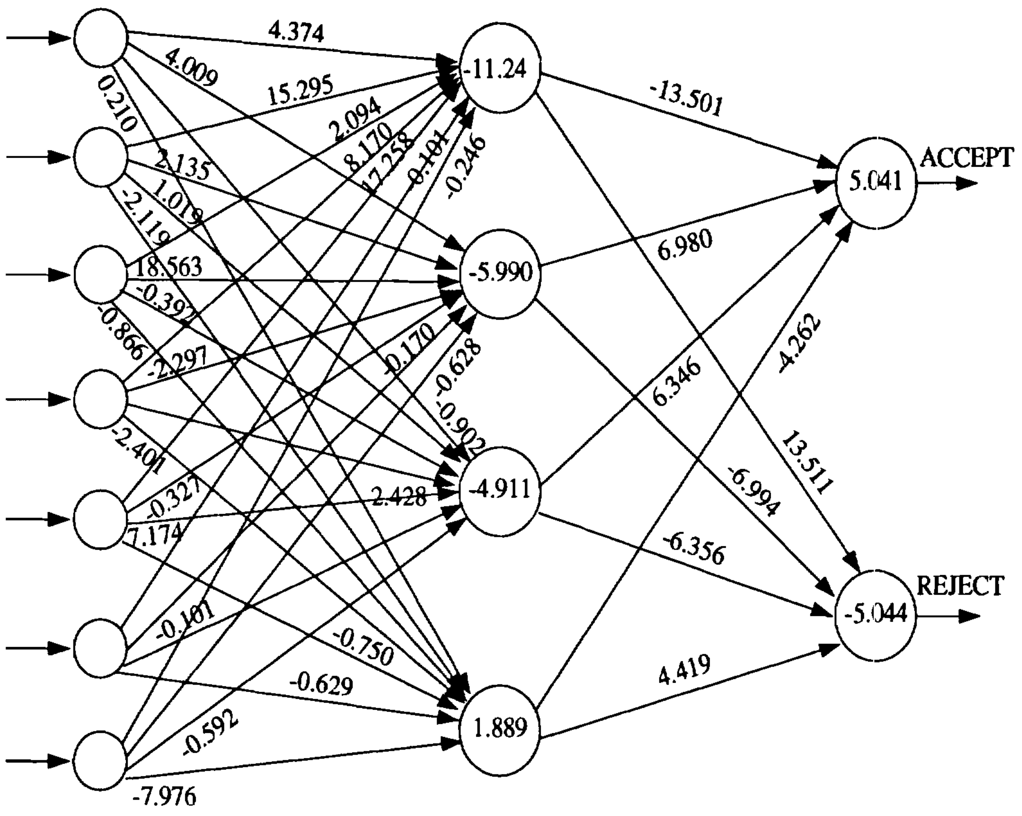


Figure 3.18 Pre-Rejector of Module 1 of PPSHNN.

The pre-rejector is perhaps the most important unit in the PPSHNN module. Care must be taken in choosing the classes that it should reject or accept. Hence, the design and operation of the S-unit is of great importance. With an accurate pre-rejector and the optimal selection of reject and accept classes, a complex problem space can be divided into two simpler and perhaps even linearly separable regions. This could not only decrease the training time by simplifying the problem space and hence reducing the size of the N-unit, but also increase the classification accuracy by allowing the N-units to learn a simplified problem space rather than a large and complex one.

Unlike any other neural network units in the system, the pre-rejector has to always have a very high classification accuracy. In most cases, the accuracy of the pre-rejector should not be lower than 90%. The accuracy of the unit shown in Figure 3.18 is around 95.5%.

Much of the success and failure of PPSHNN in achieving higher classification accuracies than other networks is due to this unit. Most of the classification error occurring in PPSHNN is due to a pre-rejector accepting a pattern which should have been rejected. This type of error leads, almost always, to misclassification. We call this type of error "fatal". The second type is called "nonfatal" due to the fact that over %50 of this type of error is corrected in the following stages of the network. For simplicity, sometimes we also call the pre-rejector the *P-unit.* The operation of this unit and its theoretical interpretation is further discussed in the next Chapter.

3.6 The Pre-Processor

The pre-processor is the least researched unit in the system and future research should be heavily concentrated on this unit. The sole purpose of this unit is to simplify the way problem space is presented to the respective module. In some experiments, we used simplistic pre-processors, whose task was only spreading out the data in the problem space so that the boundaries between classes could be more flexible and easily found. To do this, the pre-processor finds the statistical mean of all the data it is presented during training and memorizes that mean. Then every datum point in testing (or training) is nonlinearly pushed away from the mean, thus spreading the problem space further out. By enlarging the distance between the data points, one hopes to allow the boundaries to become so flexible that a piece of a highly nonlinear boundary can be simulated by a linear one.

CHAPTER 4

SPECIAL TOPICS IN PPSHNN

In this chapter, we discuss a special variation of the PPSHNN modules called the PNS module. We discuss its behavior and its features. In the first section, we discuss the architecture of the PNS module. In the second section, we discuss the training algorithm for this module and, finally, in Section 3, we analyze the features of this new module.

4.1 The PNS Module

In this Section, we discuss the PNS module as the basic building block for the synthesis of PPSHNN. The PNS consists of a prerejector (P-unit), a neural network classifier (N-unit), and a statistical analysis unit (S-unit). In some cases, we will refer to the combination of N-and S-units as NS-unit. The optional pre-processor and vector rejector units are not included, but they can be included in future developments of the module. While the P- and the N-units can be any type of neural network, we have chosen them to be a single stage delta rule network. The P- and NS-units are fractile in nature, meaning that each such unit may itself consist of a number of PNS modules. As before, through a mechanism of statistical acceptance or rejection of input vectors for classification, the sarnple space is divided into a number of subregions (polygons if the single-stage delta rule network is chosen). The input vectors belonging to each polygon are classified by a dedicated set of PNS modules. Since the delta rule network is used to generate the N-unit, each polygon approximates a linearly separable region*. In this sense, the total system becomes similar to a piecewise linear model.

4.2 The PNS Algorithm

The block diagram for a PNS module is shown in Figure 4.1.

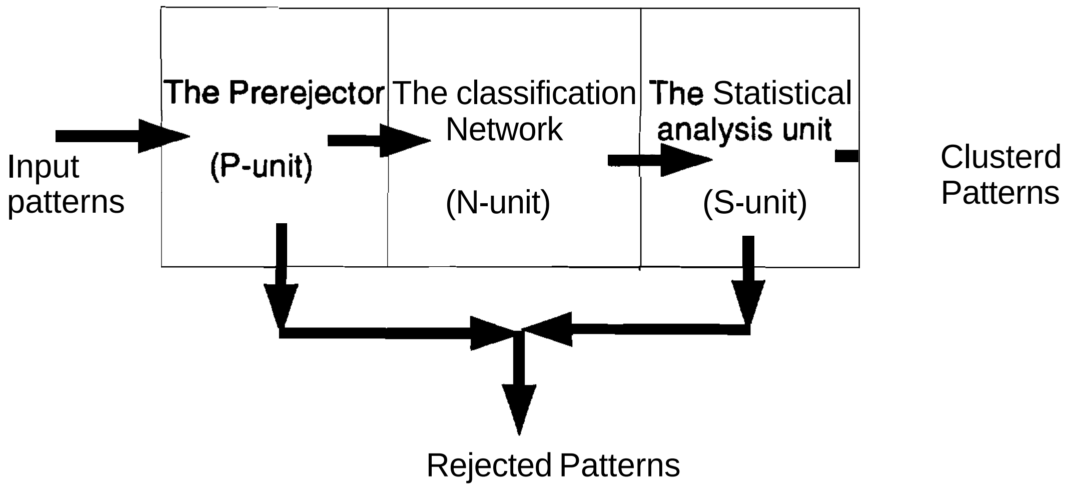


***Figure 4.1.** The Block Diagram of a PNS Module.*

The N-unit can be any type of neural network, but it is chosen as a delta rule network

* *By linearly separable region we mean part of the original problem space which is separated from the rest of the space by a combination of linear boundaries.*

with output nonlinearity in this thesis.

The procedure for the creation of the PNS modules is shown in the flow charts of Figures 4.2 and 4.3. Initially, the total network consists of a single N-unit. It has as many input neurons as the length of an input pattern, and as many output neurons as the number of classes. The number of input and output neurons may also be chosen differently depending on how the input patterns and the classes are represented. The N-unit is trained by using the present training set (each N-unit will be presented a different training set depending on where in the hierarchy its module lies). After the N-unit converges, the S-unit is created. The S-unit of the PNS module is identical to that of the PF'SHNN module. It is a parallel statistical classifier which performs bit-level three-class Bayesian analysis on the output bits of the N-unit. It was discussed in detail in Section 3.3.1. One result of this analysis is the generation of the probabilities p_1^k, $k=1,2, \cdots$ M, M being the number of classes. p_1^k signifies the probability of classifying an input pattern belonging to class k correctly. Like before, if this probability is equal to or smaller than a small threshold δ for one or more classes, a P-unit is created to reject the patterns belonging to these classes. In other words, if $p_1^k \leq \delta$, the corresponding class is either geometrically small or undersampled, or has highly nonlinear boundaries such that the present network cannot learn it.

As before, the rejection of such classes before they are fed to the N-unit is achieved by the creation of the P-unit. The P-unit is a two-class classifier trained to reject the input patterns belonging to the classes initially determined by the S-unit. In this way, the P-unit divides the sample space into two regions, allowing the N-unit to be trained with

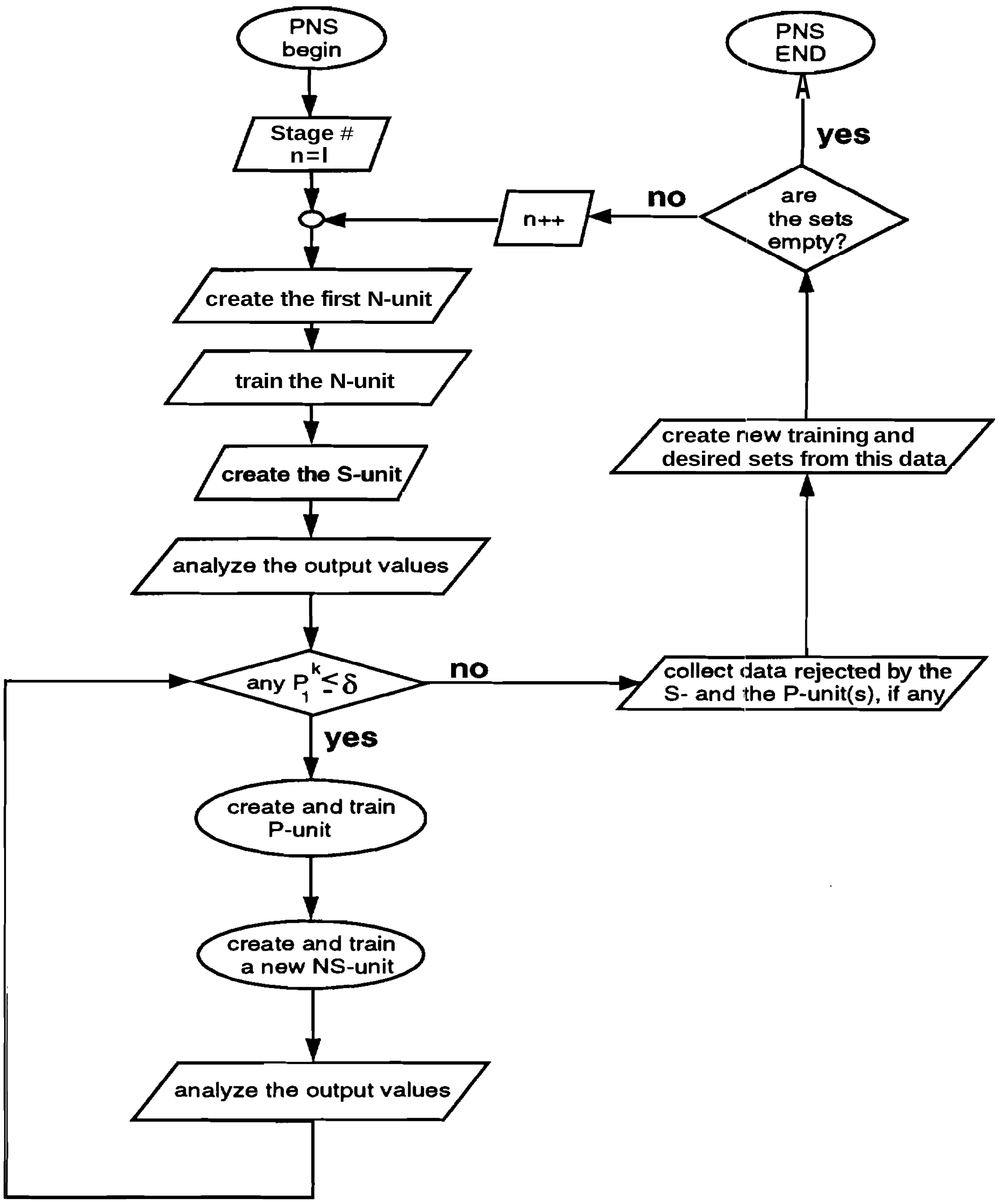
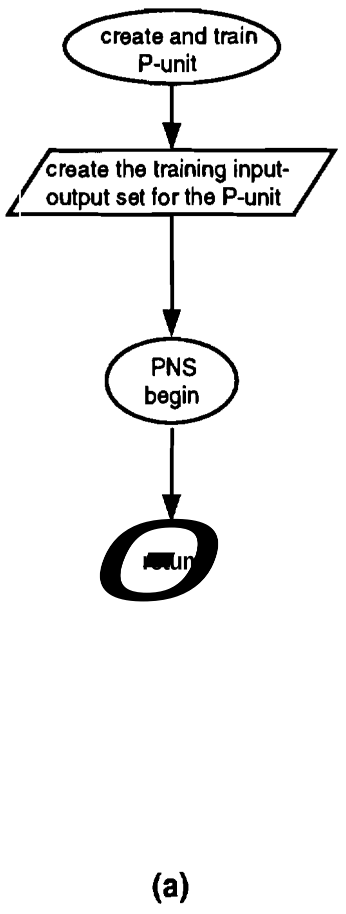

PNS begin
Stage # n=l
create the first N-unit
train the N-unit
create the S-unit
analyze the output values
any $P_1^k \leq \delta$
yes
no
create and train P-unit
create and train a new NS-unit
analyze the output values
collect data rejected by the S- and the P-unit(s), if any
create new training and desired sets from this data
are the sets empty?
yes
no
n++
PNS END

Figure 4.2. Flow Chart for Learning of a PNS Module.



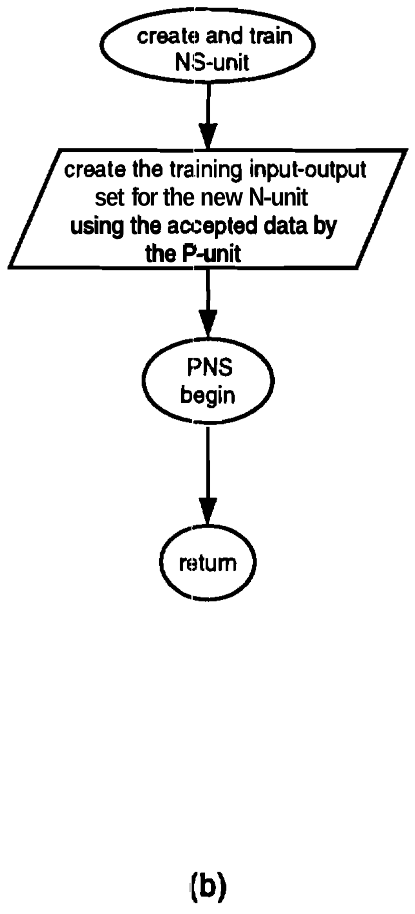

create and train NS-unit
create the training input-output set for the new N-unit using the accepted data by the P-unit
PNS begin
return
(b)

Figure 4.3. *(a)The Recursive Procedure to Create a P-unit.*
(b)The Recursive Procedure to Create a NS-unit.

patterns belonging to the classes which are easier to classify.

If a P-unit is created, the N-unit is retrained only with the patterns that are accepted by the P-unit. The process discussed above is repeated as necessary. The S-unit is regenerated; it may again reject some classes. Then, another P-unit has to be created to reject these classes. This results in a recursive procedure.

If there are no more classes rejected by the S-unit, the PNS module :is completed. The input patterns rejected by it are fed to the next PNS module.

The complicating factor in the discussion above is that there may be more than one P-unit generated. Each P-unit is a two-class classifier. Depending on the difficulty of the two-class classification problem, the P-unit may itself consist of a. number of PNS modules. The same is true with the NS-unit. The flow diagrams of the: procedure for the generation of the P-unit and the NS-unit are shown in Figure 4.3. A particular example is shown in Figure 4.4, which shows the PNS modules generated for the 10-class Colorado problem discussed in detail in Section 2.1.2. In the first stage, the P-unit required 3 PNS modules and 1 NS module to reach desired performance. Similarly, the NS-unit has actually developed into one PNS and one NS module. In this sense, the P- and the NS-units are like fractals.

Like the PPSHNN module, the S-unit also generates certain other thresholds for the acceptance or the rejection of an input pattern, as discussed in Section 3.3.1. Thus, the input pattern may be rejected by the P-unit or the S-unit. The rejected vectors become input to the next stage of PNS modules. This process of creating stages continues until all.(or a desired percentage of) the training vectors are correctly classified. For example, for the Colorado problem discussed in Section 2.1.1, two stages were required, as seen in Figure 4.4.

The recursive nature of the algorithm becomes evident when a P-unit or a NS-unit is to be created. Either unit starts as a single NS structure and builds up further, if necessary, into several parallel PNS modules. In order to create a new P- or NS-unit, it is necessary

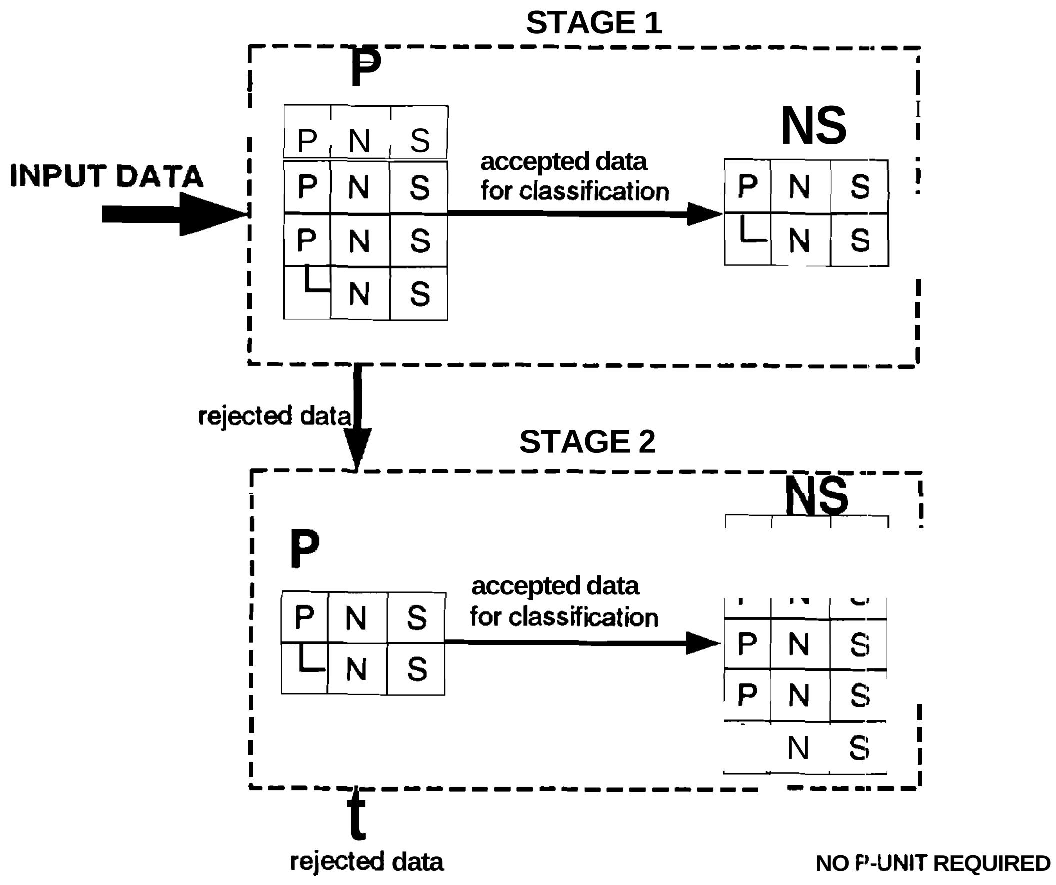


Figure 4.4. The PNS Modules in the PSHNN Designed for the 10-Class Colorado Problem.

to generate the particular training data for its learning, as shown in Figure 4.3.

Figure 4.3 shows the procedures which create the P- and the NS-units. Before the creation of the P-unit, the appropriate input-output training set has to be created. The input training set is simply the set presented to the PNS module which is being created. The corresponding desired output set is created by entering the vector $\begin{bmatrix} 1 & 0 \end{bmatrix}$ for all the patterns which should be accepted by the P-unit and the vector $\begin{bmatrix} 0 & 1 \end{bmatrix}$ for all the patterns which should be rejected by the unit. Before the creation of the NS-unit , a new input-output training set for this unit must also be created. The input set contains patterns from the original training set which are not rejected by the P-unit, and the desired output set is the collection of the corresponding desired output vectors from the original desired set.

If no more P-unit is needed, the main program branches up to train the next stage of PNS modules, as shown in Figure 4.2. To do so, the program gathers all. the rejected data from the first stage. If there are no more rejected data, or if their number is less than a preset threshold, the algorithm terminates.

In brief, the total network begins as a single PNS module and grows during training in a way similar to fractal growth. The P- and the NS-units may themselves create PNS modules. The delta rule network is used to generate the N-units. We will show that the net result is the separation of nonlinear classes into regions which are linearly separable. This separation continues until the resulting PNS network can approximate the nonlinear class boundaries using a piecewise linear model accurately. This procedure is similar to modeling of a nonlinear system by a collection of piecewise linear systems.

Remarks:

It can be shown [5] that the output values of a network based on Least-squares error minimization, such as the delta rule neural network, can be interpreted as the estimation of the conditional pdf $f(H_i|X)$, where X is the input pattern. Therefore, one can perform density estimation by such a network, which can be chosen as a PNS network. Then, the total network consists only of PNS modules.

4.3 System Features And Proof of Piecewise Linearity

As mentioned in the previous Section, the learning procedure divides the problem space into linearly separable spaces, based on the learnability of the classes by the present N-unit. Referring to Figure 3.9, this will be proven below.

Proof of Linearity:

For now let us assume that the N-unit has only one output neuron. In Section 3.3.1, we showed how to compute two rejection boundaries for every bit. In Figure 3.9, these rejection boundaries are marked as z^k_{0r} and z^k_{r1}. Since the N-unit is a single stage delta rule network with sigmoidal output nonlinearity, as described in Section 2.2, the output value of the k^{th} neuron is computed by

$$y^k = \frac{1}{1 + e^{-\sum_{i=0}^{n_i} x_i \omega_{ki}}} \tag{77}$$

Where, n_i is the number of input neurons, x_i is the value at the i^{th} input neuron, and ω_{ki} is the weight connecting the i^{th} input neuron to the k^{th} output neuron. Using (77), the equation describing the boundary imposed by the S-unit at bit k between the zero and the reject regions is

$$\frac{1}{1+e^{-\sum_{i=0}^{n_i} x_i \omega_{ki}}} = z_{0r}^k. \tag{78}$$

The above equation can be written as

$$e^{-\sum_{i=0}^{n_i} x_i \omega_{ki}} = \frac{1}{z_{0r}^k} - 1, \tag{79}$$

which leads to

$$\sum_{i=0}^{n_i} x_i \omega_{ki} = \ln\left[\frac{z_{0r}^k}{1 - z_{0r}^k}\right]. \tag{80}$$

The right hand side is a constant, making the above a linear equation. It describes a hyperplane in the n_i-dimensional space. Hence, the boundary between the "zero" and the "reject" region is linear. The same argument can be used to show that the boundary between the "reject" and the "one" region is also linear and can be described by

$$\sum_{i=0}^{n_i} x_i \omega_{ki} = \ln\left[\frac{z_{r1}^k}{1 - z_{r1}^k}\right]. \tag{81}$$

Notice that, since the equations of the two boundaries differ only in the value of the constant on the right hand side, the boundaries are parallel to each other.

QED

In the same way, every output neuron in combination with its S-unit bit-rejector, creates two linear (hyper-plane) boundaries in the n_i-dimensional space. Data falling between these boundaries are rejected by the bit-rejector. Data whose output falls outside of this region is accepted by the bit-rejector and classified, for example, based on a minimum mean square criterion [9-10]. If the certainty of classification for a class grows, the two boundaries move closer to each other, making the reject region smaller. If the certainty is one, the two boundaries lie on top of each other and there is no reject region. This is the case for bit one in the 10-class problem (see Figure 4.5).

Proof of Piecewise Linearity:

Now let the network have n_o output neurons. Each output neuron and. its corresponding bit-rejector create two linear boundaries and three regions: zero, one, and the reject regions. This results in $2n_o$ boundaries in the n_i-dimensional problem space, which divide the space into a number of polygons. A loose upper bound for this number can be expressed as:

$$A_{n_o} = \begin{cases} 3^{n_o} & n_o \leq n_i \\ 3^{n_o} - 3^{n_o - n_i} + 1 & n_o > n_i \end{cases} . \tag{82}$$

Proof:

We will prove this in two steps:

1. For now let us assume the S-unit is not existent. In other words, for every output neuron, only one boundary is created. Hence we have n_o boundaries and

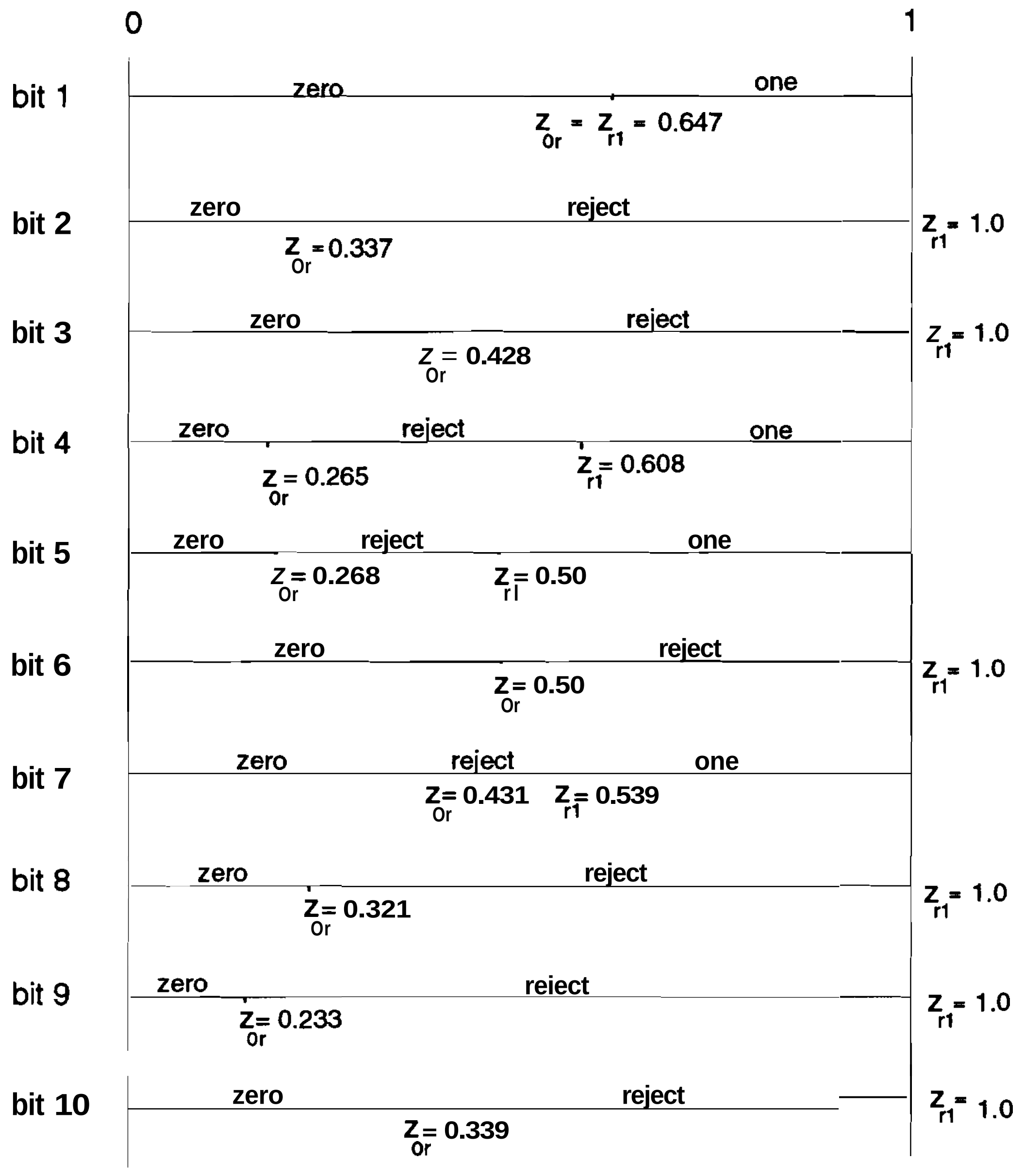


Figure 4.5. The S-unit boundaries created for the 10-Class Colorado Problem.

$$A_{n_o} = \begin{cases} 2^{n_o} & n_o \le n_i \\ 2^{n_o} - 2^{n_o - n_i} + 1 & n_o > n_i \end{cases} . \qquad (83)$$

The first case is relatively straight forward. If $n_o \le n_i$, then the maximum number of polygons are created when the boundaries share at least one point. Because it is assumed that the activation level of every output neuron represents a different feature of the classification problem, it is assumed that the weight vectors of these neurons are different from each other. And because these weight vectors are normal vectors of their respective hyperplanes, it is therefore assumed that these hyperplanes are not parallel to each other. Hence, it is perceivable that they all share a common point. Therefore, 2^{n_o} is actually the maximum number of polygons created and is the tightest upper bound. In such a case with every new boundary, we can divide every existing polygon into two sub polygons. In other words,

$$A_{n_o} = 2A_{n_o-1} = 2^{n_o} \qquad \text{for } n_o \leq n_i . \tag{84}$$

The second statement of (83) can be proven as follows. Let us assume that we have n_i boundaries and they have divided the problem space into $A_{n_i} = 2^{n_i}$ polygons using n_i boundaries. Every additional boundary will not be able to divide all of the polygons because of the linearity property of the bounclaries. This means that the n_i+1 st boundary will cut at most $A_{n_i} - 1$ regions. This means:

$$A_{n_{i+1}} = 1 + 2\,(A_{n_i} - 1\,) . \tag{85}$$

The "1" in the equation is for the one region not touched by the new boundary. The rest of the equality is for all the regions that are divided into two subregions. The same argument can be made for every additional boundary, resulting in the general difference equation:

$$A_{n_i} = 2^{n_i}$$
$$A_{n_o} = 1 + 2\,(A_{n_o - 1} - 1) \qquad n_o > n_i\ . \tag{86}$$

Using induction, we can now show that this is the same as the second statement of (83).

Induction basis: For $n_o = n_i$, from (83) we get:

$$A_{n_o} = 2^{n_i} - 2^{n_i - n_i} + 1 = 2^{n_i} \tag{87}$$

Induction hypothesis: $A_{n_o} = 1 + 2\,(A_{n_o - 1} - 1) = 2^{n_o} - 2^{n_o - n_i} + 1$ for $n_o > n_i$.

Induction proof for $A_{n_o + 1} = 2^{n_o + 1} - 2^{n_o - n_i + 1} + 1$. Using (87) and the induction hypothesis, we can write:

$$A_{n_o + 1} = 1 + 2\,(A_{n_o} - 1) = 1 + 2\,(2^{n_o} - 2^{n_o - n_i} + 1 - 1), \tag{88}$$

or

$$A_{n_o + 1} = 2^{n_o + 1} - 2^{n_o - n_i + 1} + 1\ . \tag{89}$$

QED

. Now let us add the S-unit in. This will cause two boundaries to be created for every output neuron. The two boundaries are parallel hyperplanes because of the fact that for both planes the weight vectors are the same. Hence, the normal vector to both hyperplanes are the same, and the planes are parallel. The only difference between the two vectors is on the right hand side of the equation of the hyperplane as seen in (80) and (81).

Now let us consider (82). Here, the same argument used for (83) can be applied, except that now with every additional neuron, we are adding two parallel boundaries rather than one. This means that now every polygon that the new set of boundaries enters will be divided into three subpolygons rather than two. This is true for both cases in (82). Therefore, by following the same argument as before and by keeping in mind that every set of boundaries divides the regions into three subregions, the upperbounds of (82) will follow.

QED

Introduction of the P-unit to the problem space: The P-unit is chosen as a single stage, delta rule, two-class classifier network. It introduces at least one additional linear boundary to the problem space (the argument for linearity is identical to that of the N-unit). The additional boundary(ies) serves to divide the problem space. into further reject and accept regions. The difference here is that the reject region is completely dropped out of the problem space of the N-unit, and the N-unit does not learn it.

As an example, Figure **4.6** shows the problem space of the XOR problem as it is learned by the PNS module. Figure **4.6** (a) shows the PNS module developed for this problem. Due to the simplicity of the problem, the P-unit consists of only one neuron. The N-unit consists of two neurons. Figure **4.6** (b) shows the two boundaries which the N-unit imposed upon the problem space. The "one" regions of the boundaries overlap in the

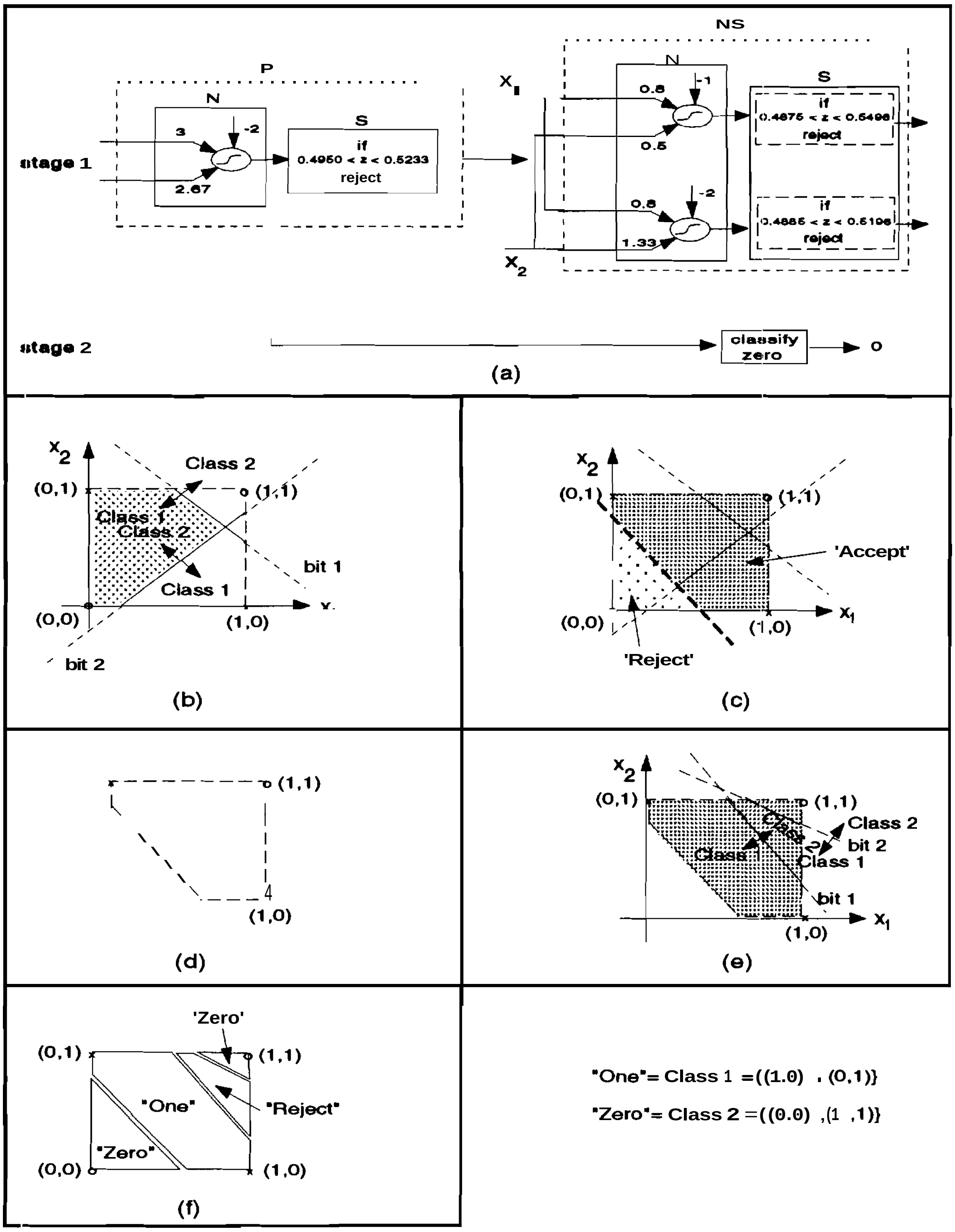


Figure 4.6. A PNS Module for the XOR Problem and its Problem Space.

dotted area. Notice that the boundaries are only of consequence in the problem space (the square shown in the figure). Hence the boundaries are finite lines (solid boundary lines in the figure). Figure 4.6 (c) includes the boundary imposed by the P-unit. Figure 4.6 (d) demonstrates the problem space introduced to the N-unit after implementing the P-unit. This space is linearly separable and can be learned by a single stage delta rule network.

The N-unit is retrained to separate the classes in the new space. It creates the boundaries shown in Figure 4.6 (e). Notice that the two boundaries accomplish the same task and thitt one can be eliminated. In other words it is sufficient to have only one neuron as the N-unit. In general, this process of elimination can be achieved by introducing a new unit to the output of the N-unit. The job of this unit would be to compare the weight vectors of output neurons after training. It would compare these vectors two at a time, and if it detected a linear dependence between any two vectors, it would eliminate one of them by eliminating its corresponding output neuron from the network. To follow up the argument presented above, however, we keep both neurons.

It is important to mention here that at this stage the boundaries of the retrained N-unit are no longer merely confined to the boundaries of the original problem space, but are also bounded by the boundaries which the P-unit imposes. In other words, all the boundaries are bounded by the current problem space at hand (dotted area in Figure 4.6 (e)), and not by the boundaries of the original problem space (shown in Figure 4.6 (b) as a dotted square).

The final space division by the PNS network is shown in Figure 4.6 (f). Notice that the region marked "Reject" also will be classified "Zero" because of the automatic classification of all rejected vectors as "Zero". In the above discussion we have ignored the S-units. Introduction of the S-units changes the space division in the manner shown in Figure 4.7. As we see, every boundary of Figure

'One'= Class 1 = {(1,0) , (0,1)}

'Zero'= Class 2 = {(0,0) , (1,1)}

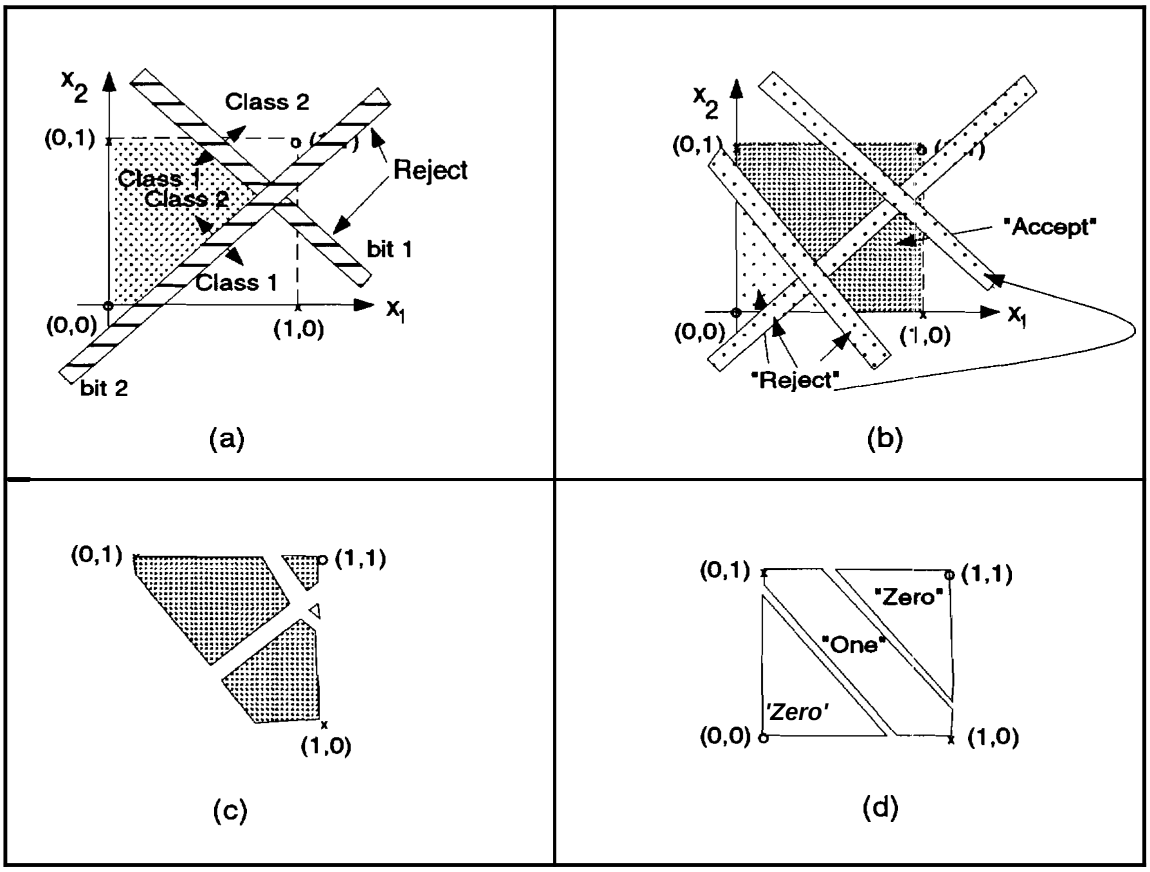


Figure 4.7. The Division of XOR Problem Space after Introduction of S-units.

4.6 has been replaced by a region of uncertainty ("reject" region). Data falling in these regions are rejected to the second module and classified automatically as "Zero". The final result shows that, due to this fact, in this case, introduction of the S-units only causes the "one" region to shrink and the "zero" region to expand.

Hence, as we have seen, the function of the P-unit is to divide the problem space into simpler polygons by introducing new boundaries to the space. This division of space can result in complete elimination of one or more class(es) from the problem space (polygon) of some modules. In the 10-class problem, the P-unit of the first module eliminates six out of ten classes from the problem space of the N-unit of the first module. This results in only four output neurons for this N-unit and thereby $2 \times 4 = 8$ boundaries, or by using the upper bounds of (82), at most $3^4 = 81$ polygons (subregions) versus ten output neurons, $2 \times 10 = 20$ boundaries, and an upper bound of $3^{10} - 3^3 + 1 = 59023$ polygons (subregions).

As the result of the above argument, the problem space introduced to the N-unit only contains $n_o' < n_o$ which is the number of classes accepted by the P-unit. Therefore, the NS-unit creates $2n_o'$ linear boundaries and $A_{n_o'} = O\left[3^{n_o'}\right]$ polygons*. These new linear boundaries are confined to the boundaries of the polygon passed to the N-unit rather than the limits of the original problem space.

+ *From now on, we use the word "polygon" to indicate that region of the problem space that is parsed down through the hierarchy to a certain unit in the network for classification. In other words, by "polygon of the N-unit", we mean that region of the space which the particular N-unit is responsible for.*

The above discussion is valid only under the fundamental assumption that the polygon of the: N-unit is linearly separable (i.e. A single stage delta-rule network can accurately classify patterns from this region). The same assumption should be valid for the P-unit. The problem space of the N-unit may not be linearly separable*, even after simplification of space by the P-unit. The polygon of the P-unit may also not be linearly separable. In such cases, the P- or the N-unit or both is replaced by an entire PNS module. If this is still not sufficient, the P- and/or N-unit(s) of the new PNS module is also replaced by a PNS module. In this way, the PNS modules are created in a way similar to fractals until the performance of the overall network is satisfactory. The fractile architecture will have several P-units which will serve to further divide the space. Their respective N-units impose linear boundaries upon these polygons. The polygon of each N-unit is the accept subregion of its corresponding P-unit and the boundaries it creates are confined to this subregion.

In summary, the problem space is divided into as many polygons as necessary to reach linearly separable polygons. This division is performed by the P-units. Then the NS-units create linear boundaries which are only defined within the confines of their respective polygons. The whole process results in the separation of linearly separable regions of a nonlinear classification space by hierarchically organized piecewise linear subsystems which are structured within each other like fractals.

+ *By a linearly separable region, we mean that the classes of the region (polygon) can be separated from each other by a linear boundary.*

Since we desire for any given input pattern, only one output bit to go high, we shall desire the one region of each output bit to fall on top of the "zero" region of the other bits. It can easily be shown that this is not possible for more than one linear boundary unless they all lie on top of each other (identical boundaries). Hence, we will most likely have regions of the n_i-dimensional space which are classified "one" by more than one bit. Since the problem space is a subspace of the n_i-dimensional space, one hopes that such regions fall outside of the problem space. An example of this is shown in Figure 4.8. Figure 4.8 (a) shows an example of the overlapped "one" region in the problem space. Figure 4.8 (b) shows the opposite, where the overlapped region is outside the problem space.

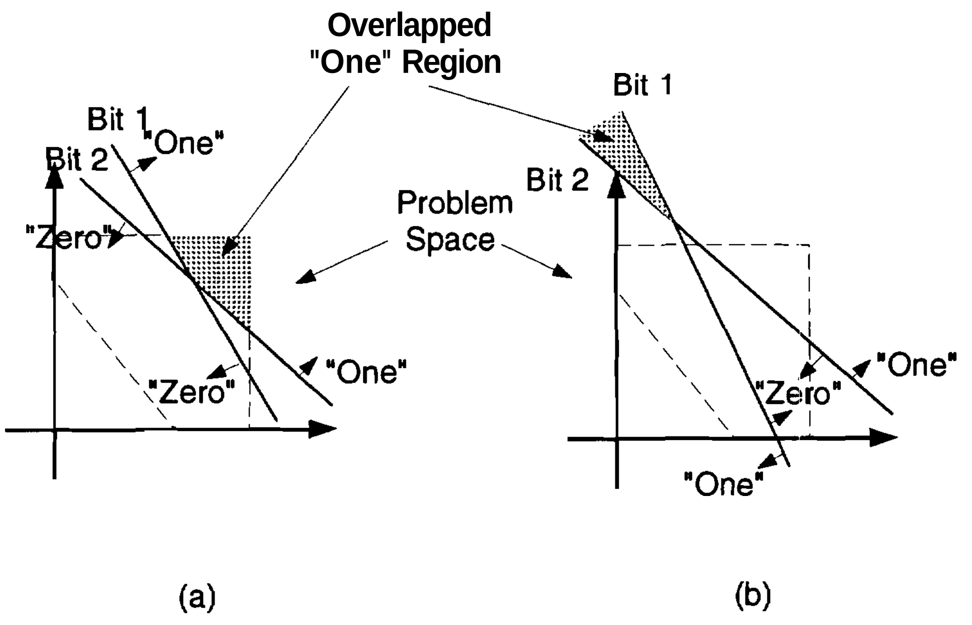


Figure 4.8. An example of overlapped "one" regions.

If this special case occurs, the classification accuracy is extremely high. However, since almost every bit creates two boundaries, this phenomenon rarely occurs. Therefore, we will have overlapped regions in the problem space, and for patterns falling in these regions more than one output bit will go high. We need a mechanism. to serve as a "tie breaker." In other words, we need a mechanism which decides which one of the "high" bits is dominant, thereby choosing its respective class over the others. One could simply decide to let the minimum mean square error mechanism at the output perform this task. It can be shown that this mechanism chooses the class for which the pattern sample is farthest away from its boundary. In other words, the class that is chosen is the class that the sample output is deeper in its "one" region. (see Figure 4.9)

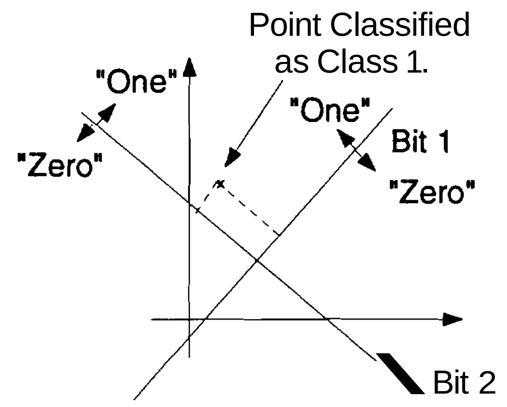


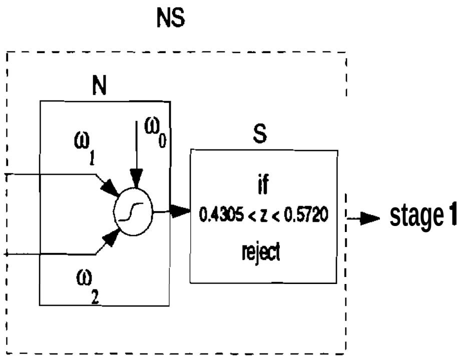
Figure 4.9. ***Minimum Mean Squared Error Decision in 2-D Space.***

This method works well in an unnoisy problem space such as the XOR problem. But in noisy situations, since the output of the N-unit is shifted, this measure could prove to be inaccurate. Introducing a vector rejector after the bit-rejectors is one solution to this problem. The vector rejector is a neural network unit. This unit introduces new boundaries to the polygons of the problem space which have been created by the P- and the NS-units. These boundaries act as tie breakers and, since they are (adaptive, they can take the noisy characteristics of the problem space into account.

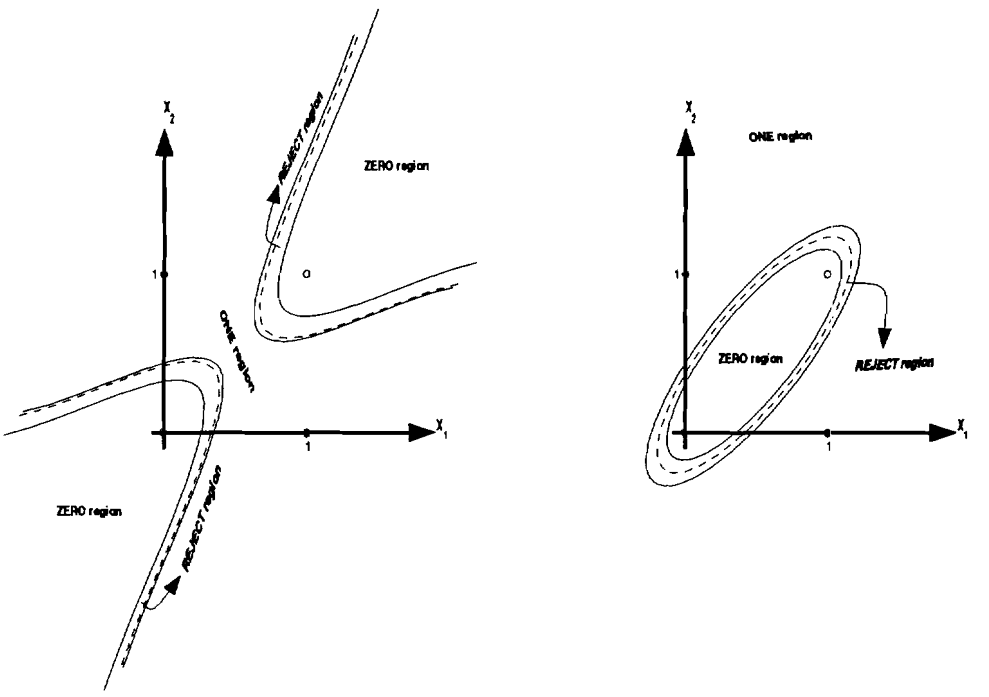
It should be mentioned here that there could also be regions in the problem space whose data patterns are classified as zero in every bit. In other words, in some regions of the space, the zero regions of all the bits overlap. The vector rejector could also be trained to work as a tie breaker in such cases as well.

From the above discussion, the following important result follows: A network of PNS modules divides the problem space into linearly separable regions, as in a piecewise linear model. The reject regions also impose additional boundaries to separate the "hard" to classify patterns from the "easy" to classify patterns. These additional boundaries are also linear due to the fact that all networks used in the PNS experiments (in the P- and the N-units) were single stage delta rule networks. Each PNS module contributes to the task of approximating the class boundaries by building a linear piece of the overall model.

It is important to mention that, by using other types of networks instead of the single stage delta rule network, or by using different types of neurons, the piecewise linear model could become a piecewise nonlinear model. For example, the results obtained with the use of quadratic neurons for the XOR problem is shown in Figure 4.10. The only difference here is that the input values are squared before inputting to the output neuron. The k^{th} output neuron has the output given by



(a)



(b) (c)

Figure 4.10. *a) A Second Order Polynomial Network for the XOR Problem, b) and c) Possible Accept and Reject Regions.*

$$y^k = \frac{1}{1 + e^{-\sum_{i=0}^{n_i} x_i^2 \omega_{ki}}} \tag{90}$$

The equation of the boundaries can be derived in a way similar to the linear case and is given by

$$\sum_{i=0}^{n_i} x_i^2 \omega_{ki} = \ln\left[\frac{z_{0r}^k}{1 - z_{0r}^k}\right]. \tag{91}$$

This may result in a hyperbolic or an elliptic boundary as shown in Figures 4.10 b) and c). In this case, only one stage is generated to correctly classify the XOR problem with no P-unit, and the N-unit is a 2-1 unit as in Figure 4.10 a).

The change to quadratic neurons had little effect in the overall accuriicy of the system, leading us to believe that the total network consisting of PNS modules based on the delta rule is very effective in overall classification accuracy while rernaining relatively inexpensive.

CHAPTER 5

A PARALLEL SIMD ALGORITHM FOR MASPAR; THE SIMD-PPSHNN

In this chapter we describe the parallel implementation of the PPSHNN with two-stage backpropagation networks as its P- and N-units and with PNS modules with single stage delta rule networks. In particular we describe the SIMD versions of these networks implemented on MasPar MP-1.

For simplicity, we refer to the PPSHNN network with two-stage: backpropagation networks as the PPSHNNl and with single-stage delta rule PNS modules as the PPSHNN2. We refer to the parallel SIMD version of their respective algorithms as SIMD-PPSHNN1 and SIMD-PPSHNN2. We also refer to the process of producing an output vector for an input pattern by the N-unit as throughput.

We first describe the architecture of MasPar *MP-1*[12-14], and then describe the SIMD [11-12] version of PPSHNN and how it was adapted to MasPar MP-1 architecture to take advantage of its features. Section 5.2 is the general parallel algorithm description for both networks. In section 5.3, the time complexities of the serial and parallel versions of the PPSHNNl and PPSHNN2 algorithms are analyzed and estimated. Section 5.4 offers a theoretical speed up comparison between the SIMD-PPSHNN1 and SIMD-PPSHNN2 ant3 their respective serial algorithms. In Section 5.5 the parallel testing procedure is discussed.

5.1 Introduction to MasPar MP-1

Massively parallel computers normally use more than 1024 processors to obtain computational performances unachievable by conventional computers. The MasPar Computer Corporation has designed and implemented a high performance, massively parallel computer system called the MP-1. The MasPar MP-1 system is scalable from 1024 to 16384 processors and its peak performance scales linearly with the number of processors. A 16K processor system delivers 30,000 MIPS peak performance where a representative instruction is a 32-bit integer add. In terms of peak floating point performance, the 16K processor system delivers 1,500 MFLOPS single: precision (32-bit) and 650 MFLOPS double precision (64-bit), using the average of add and multiply times.

Because massively parallel systems focus on data parallelism, all the processors can execute the same instruction stream. The MP-1 has a Single Instruction Multiple Data (SIMD) architecture that simplifies the highly replicated processors by eliminating their instruction logic and instruction memory, thus saving millions of gates and hundreds of megabytes of memory in the overall system. The processors in a SIMD system are called Processing Element (PE) to indicate that they contain only the data path of a processor.

Unique characteristics of the MP-1 architecture include the combination of a scalable architecture in terms of the number of Processing Elements (PEs), system memory, and system communication bandwidth, "RISC-like" instruction set design that leverages optimizing compiler technology, adherence to industry standard floating point design, and an architectural design amenable to a VLSI implementation.

Figure 5.1 shows a block diagram of the MasPar system with five major subsystems. The following describes each of the major components:

The Array Control Unit (ACU): The ACU is a 14 MIPS scalar processor with a RISC-style instruction set. It fetches and decodes MP-1 instructions, computes addresses and scalar data values, issues control signals to the PE array, and monitors the status of the PE array; but most of the scalar ACU instructions execute in one 70 nsec clock. The ACU occupies one printed circuit board.

The ACU performs two primary functions: either PE array control or independent program execution. The ACU controls the PE array by broadcasting all PE instructions. Independent program execution is possible since it is a full control processor capable of independent program execution.

The ACU is a custom designed processor with the following major architectural characteristics:

— Separate instruction and data spaces

— 32-bit, two address, load/store, simple instruction set

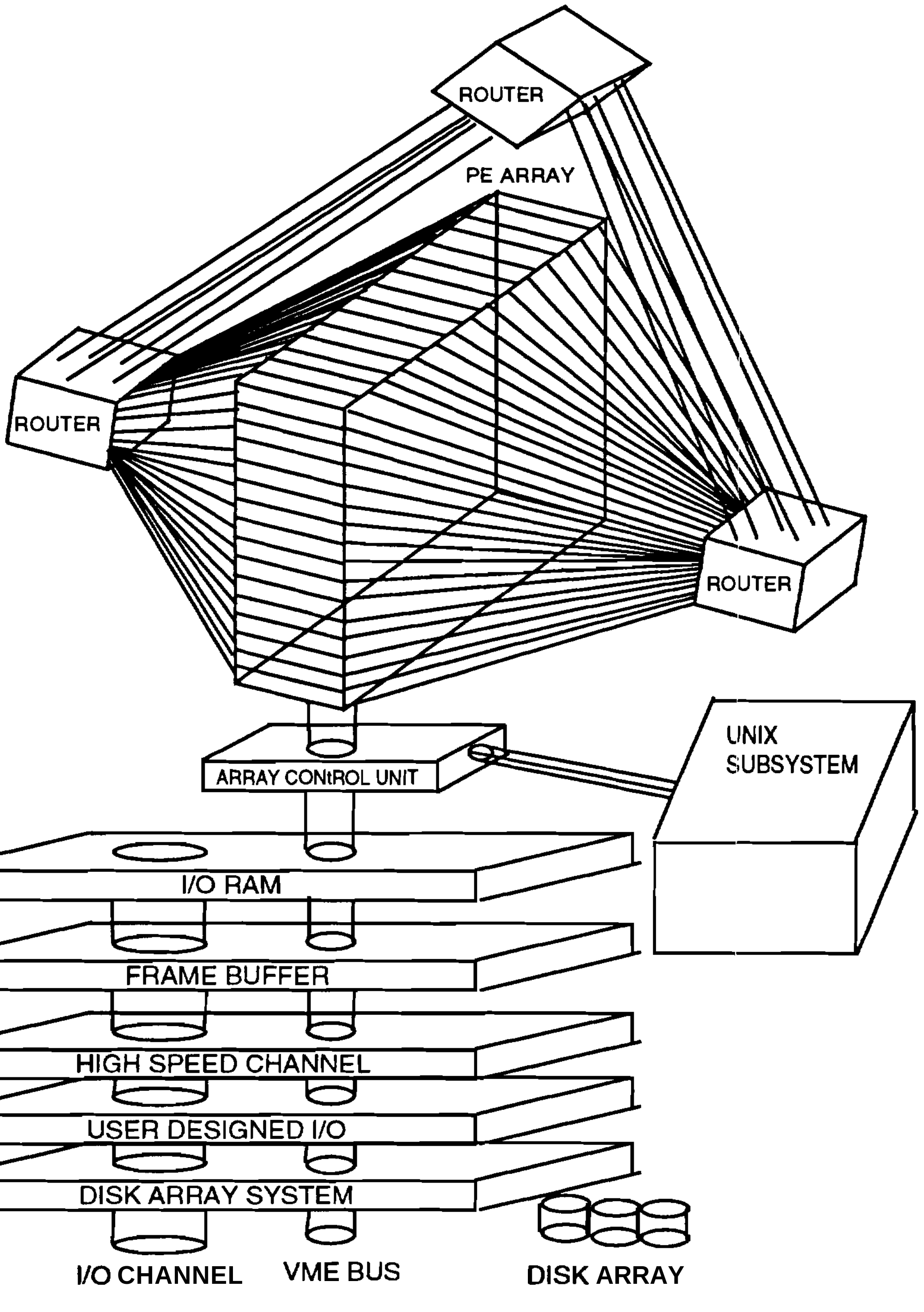


Figure 5.1 Bbck Diagram of MasPar MP-1.

— 4 Gigabyte, virtual, instruction address space, using 4K bytes per page.

The ACU has a microcoded implementation of its RISC-like instruction set due to the additional control requirements of the PE array. PE instructions typically require more than one clock cycle, including floating point instructions which are well suited to a microcode implementation.

Processor Array: The MP-1 processor array (Figure 5.2)

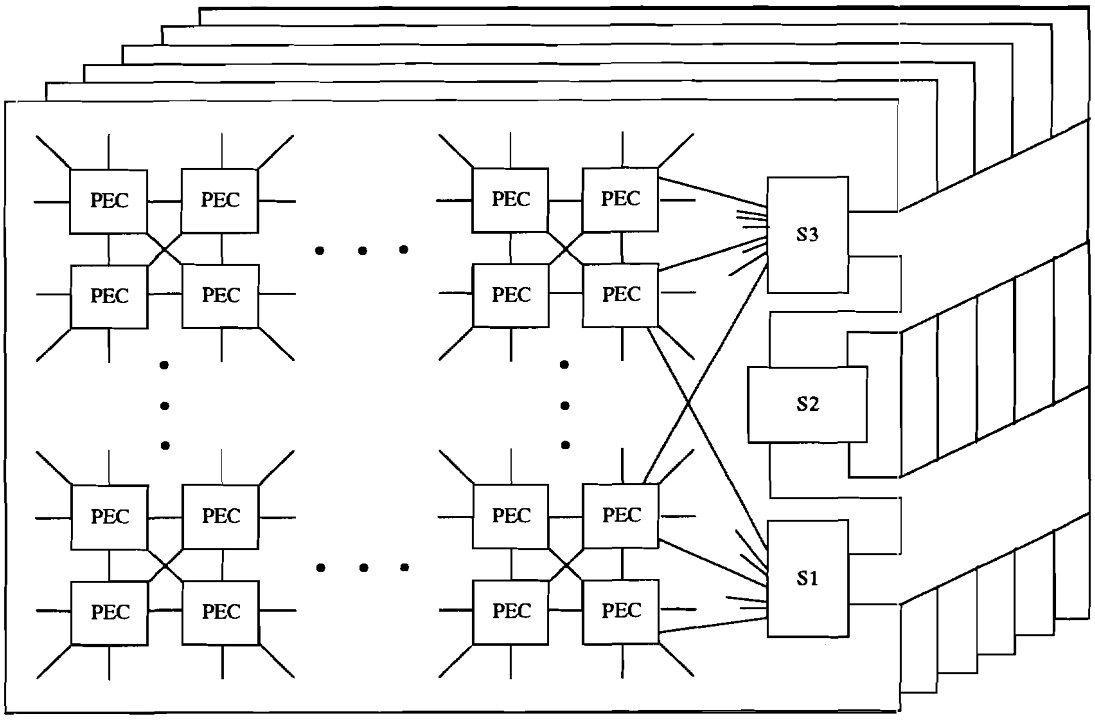


Figure 5.2. Physical Organization of the Array Processor of MP-1. 1024 PEs on each Board, Organized in Clusters of 16 PEs.

is configurable from 1 to 16 identical processor boards. Each processor board has 1024 PEs and associated memory arranged as 64 PE clusters (PECs) of 16 PEs per cluster. The processors are interconnected via the X-Net neighborhood mesh and the global multistage crossbar router network. A processor board dissipates less than 50 watts; a full 16K PE array and ACU dissipate less than 1,000 watts.

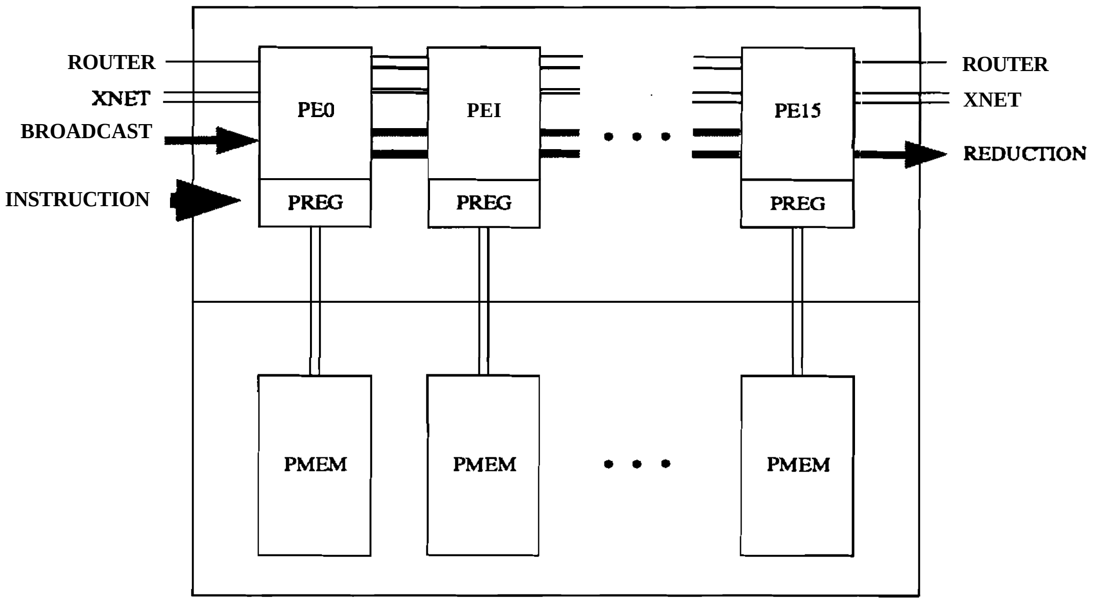


Figure 5.3 A PE Cluster of MasPar.

A PE cluster (Figure 5.3) is composed of 16 PEs and 16 processor memories (PMEM). The PEs are logically arranged as a 4 by 4 array for the X-Net two-dimensional mesh interconnection. Each PE has a large internal register file shown in the figure as PREG. Load and store instructions move data between PRES and PMEM. The ACU broadcasts instructions and data to all PE clusters, and the PEs all contribute to an inclusive-OR reduction tree received by the ACU. The 16 PEs in a cluster share an access port to the multistage crossbar router.

The MP-1 processor chip is a full custom design that contains 32 identical PEs (2 PE clusters) implemented in two-level metal 1.6μ CMOS and packaged in a cost effective 164 pin plastic quad flat pack. The die is 11.6 mm by 9.5 mm, and has 450,000 transistors. A conservative 70 nsec clock cycle yields low power and robust timing msugins.

Processor memory, PMEM, is implemented with 1Mbit DRAM's that are arranged in the cluster so that each PE has 16 Kbytes of data memory. A processor board has 16 Mbytes of memory, and a 16 board system has 256 Mbytes of memory. The MP-1 instruction set supports 32 bits of PE number and 32 bits of memory addressing per PE, so the memory system size is limited only by cost and market considerations.

As an MP-1 system is expanded, each increment adds PEs, memory, and communication resources, so the system always maintains a balance between processor performance, memory size and bandwidth, and communications and I/O bandwidth.

The MP-1 processor element (PE) design is different than that of a conventional processor because a PE is mostly data path logic and has no instruction fetch or decode logic. Like present RISC processors, each PE has a large on-chip register set (PREG) and all computations operate on the registers. Load and store instructions move data between the external memory (PMEM) and the register set. The register architecture substantially improves performance by reducing the need to reference external memory. The compilers optimize register usage to minimize load/store traffic.

Each PE has forty 32-bit registers available to the programmer and eight additional 32-bit registers that are used internally to implement the MP-1 instruction set- With 32 PEs per die, the resulting 48 Kbits of register occupy about 30% of the die area, but represent 75% of the transistor count. Placing the registers on-chip yields an aggregate PE/PREG bandwidth of 117 gigabytes per second with 16K PEs. The registers are bit and byte addressable.

Each PE provides floating point operations on 32 and 64 bit IEEE or VAX format operands and integer operations on 1, 8, 16, 32, and 64 bit operands. The PE floating point/integer hardware has a 64-bit MANTISSA unit, a 16-bit EXPONENT unit, a 4-bit ALU, a 1-bit LOGIC unit, and a FLAGS unit; these units perform floating point, integer, and boolean operations. The floating pointlinteger unit uses more than half of the PS silicon area, but provides substantially better performance than the bit-serial designs used in earlier massively parallel systems.

Most data movement within occurs on the internal PE 4-bit NIBBLE BUS and the BIT BTJS (Figure 5.4). During a 32-bit or 64-bit floating point or integer instruction, the ACU microcode engine steps the PEs through a series of operations on successive 4-bit nibbles to generate the full precision result. Because the MP-1 instruction set focuses on conventional operand sizes 8, 16, 32, and 64 bits, MasPar can implement subsequent PEs with smaller or larger ALU widths without changing the programmers instruction model. The internal 4-bit nature of the PE is not visible to the programmer, 'but does make the PB flexible enough to accommodate different front-end workstation data formats. The PI3 hardware supports both little-endian and big-endian format integers, VAX floating point F, D, and G formats, and IEEE single and double precision floating point formats.

UNIX Subsystem (USS): An important aspect of the system is the use of an existing computer system (specifically a VAX station 3520 *ULTRIX*^{TM} workstation) that follows existing industry standards (e.g. X windows, TCPIP, etc.). The USS provides a complete

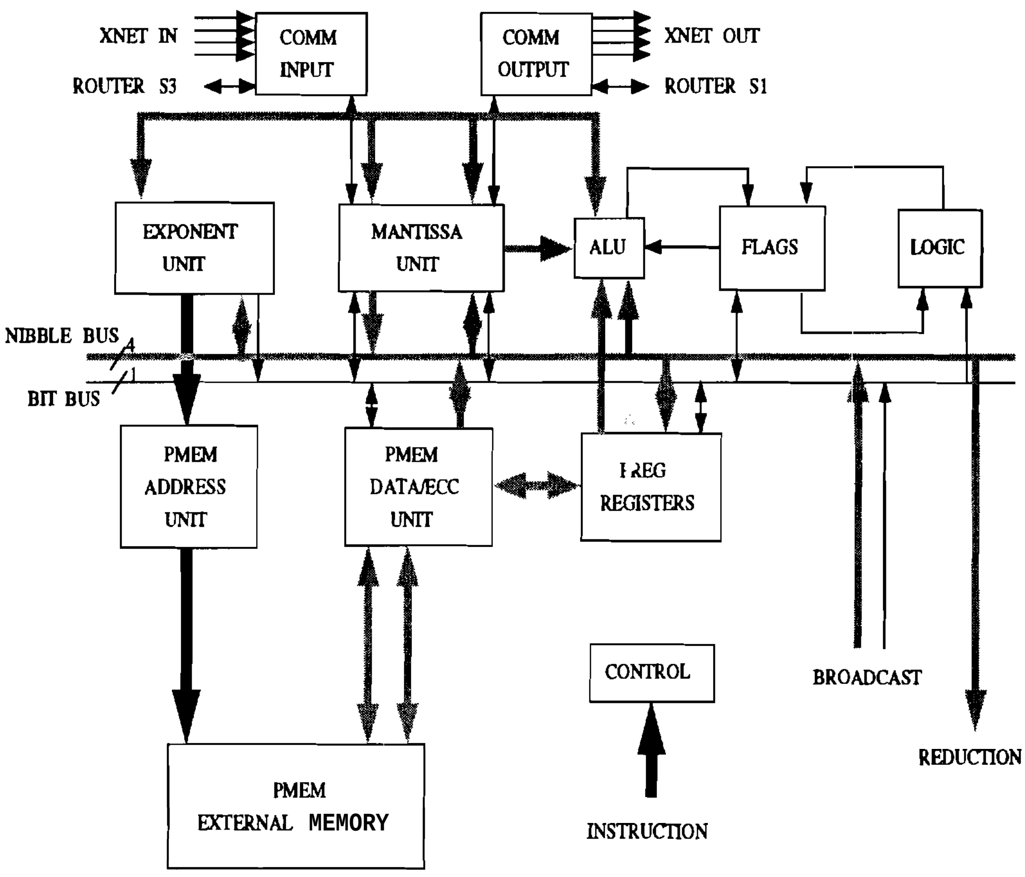


Figure 5.4 Internal Architecture of a PE.

network and graphic based software environment in which all the MasPar tools and utilities (e.g. compilers) execute. Part of the application executes as a conventional workstation application; most of the "operating system" functions are provided by the workstation's UNIX software.

Communication Mechanism: The following sections describes the five major communications mechanisms.

1. USS to ACU: Three different interactions occur between the USS and the ACU,

which use three different hardware supports. All are based on a standard bus interface (VME). The following describes each mechanism:

I. Queues: Hardware queues are provided which allows the USS process to quickly interact with the process running on the ACU. The programming model is similar to UNIX pipes but with hardware assist.

II. Shared memory: The shared memory mechanism overlaps ACU memory addresses with USS memory addresses. This provides, a strait forward mechanism for processes to share common data structures like file control block etc.

III. DMA: A DMA mechanism is provided that permits fast bulk data transfers without using programmed I/O.

- ACU to PE array: Two basic capabilities are required for data movement between ACU and PE array: data distribution, DIST, and array consensus detection which uses a global OR, GOR.

I. PE array: XNet XNet communications provide all PEs with direct connection to its eight nearest neighbors. Processors on the physical edge of the array have toroidal wrapped edge connections.

Three basic instruction types are provided to use the nearest neighbor connections:

a. XNET: The XNET instruction moves an operand from source to destination a specified distance in all active PEs. The instruction time is proportional to the distance times the operand size, since all communication is done using single wire connections.

b. XNETP: The XNETP instruction is pipelined so that a collection of PEs move an operand from source to destination over a specified distance. However, the pattern of active and inactive PEs is very important since active PEs transmit data and inactive PEs act as pipeline stages. The instruction time is proportional to distance plus the operand size due to its pipelined nature.

c. XNETC: The XNETC instruction is pipelined and is very similar to XNETP instruction, except that a copy of the operand is left in all PEs acting as a pipeline stage. Again the instruction time is proportional to the distance plus the operand size.

II. PE array: Global Router The global router is a circuit switched style network organized as a three stage hierarchy of crossbar switches. This mechanism provides direct point to point bidirectional conimunications. The network diameter is $\frac{1}{16}$ the number of PEs, which requires a minimum of 16 communication cycles to do a permutation with all PEs. The basic instruction primitives are:

a. ropen: open a connection to a destination PE

b. rsend: move data from the originator PE to the destination PE

c. rfetch: move data from the destination PE to the originator PE

d. rclose: terminate the communication

III. PE array to UO subsystem: Since the global router provides high performance random PE to PE communication, the global router is also used to provide a high performance communication mechanism to the UO subsystem. The interface is achieved by connecting the last stage of the global router to an UO device, the I/O RAM. The programming model is identical to the model for using the global router.

3. Array UO system: Referring back to Figure 5.1, the UO subsystem uses the following key components: the global router connection into the PE array (over 1 $\frac{\text{GB}}{sec}$), a large UO RAM buffer (up to 256 MB), and a high speed (230 $\frac{\text{MB}}{sec}$) data communication channel between peripheral devices, a bus for device control (not for data movement). Using output as an example, the model for using the UO subsystem follows these steps:

a. Device is opened by the USS (all UO devices are UNIX controlled)

b. The ACU moves data into the UO RAM through the global router.

c. Either the USS or an I/O processor (IOP) schedules data movement from the I/O RAM to the device (e.g. Disk) (data through the MPIOC (MP I/O Channel) and control through the VME bus).

d. The USS is notified when the transaction is complete.

Note that all transactions from the I/O Ram to external I/O systems can occur asynchronously from PE array actions. This is a key attribute since data can move into the YO RAM at speeds over 1 $\frac{GB}{sec}$ then move at YO device speeds, typically in the tens of megabytes per second or less, without effecting the performance of the PE array. These hardware mechanisms can support either typical synchronous UNIX I/O or newer (and faster) asynchronous software models.

5.2 Algorithm Description and Machine Adaptation

In this section, we discuss the parallel version of the PPSHNN1 and the PPSHNN2 algorithms in detail. Training procedure of the SIMD versions are the same as the serial versions shown in Figures 3.4, 4.2 and 4.3 , except that training of the N-unit, the P-unit, and the postrejector is done in parallel in a SIMD fashion. Since the training procedures of these modules are very similar, we will concentrate on the training procedure of the N-unit. Since the N-unit is chosen to be a two-stage backpropagation network or a single stage delta rule network, we concentrate on the parallelization of these learning procedures.

5.2.1 The Weight Batching and the Stochastic Backpropagation Algorithms

The backpropagation algorithm, also referred to as the generalized delta rule algorithm, is the generalization of the delta rule algorithm to multiple stages [1]. For this reason we first concentrate on parallelizing the backpropagation algorithm and then use this result to parallelize the delta rule algorithm.

The parallel version of the backpropagation algorithm (referred to as SIMD-BP) is designed for MasPar MP-1 with 16K PEs. Our design included backpropagation networks with one and no hidden layer. Without any hidden layer, the algorithm is the sarne as the delta rule with output layer nonlinearities and is further discussed later.

In standard backpropagation, an input pattern is presented to the network. Based on that pattern, the network computes an output pattern. The output pattern is compared to a desired pattern and an error vector is computed. The error is backpropagated through the network; based on the amount of error passing through each connection, the weights are changed. After that, the next pattern is presented to the network and this procedure is repeated for the new pattern. In the SIMD version of this algorithm, the weights are not changed after each pattern. The weight changes are stored; after the completion of a sweep they are added together, and only then the weights are updated (weight *batching*) baaed on the total weight change computed. Figure 5.5 shows the training procedures of the serial version of the backpropagation algorithm (BP) and its SIMD version (SIMD-BP).

The following is the derivation of the backpropagation algorithm to clarify the difference

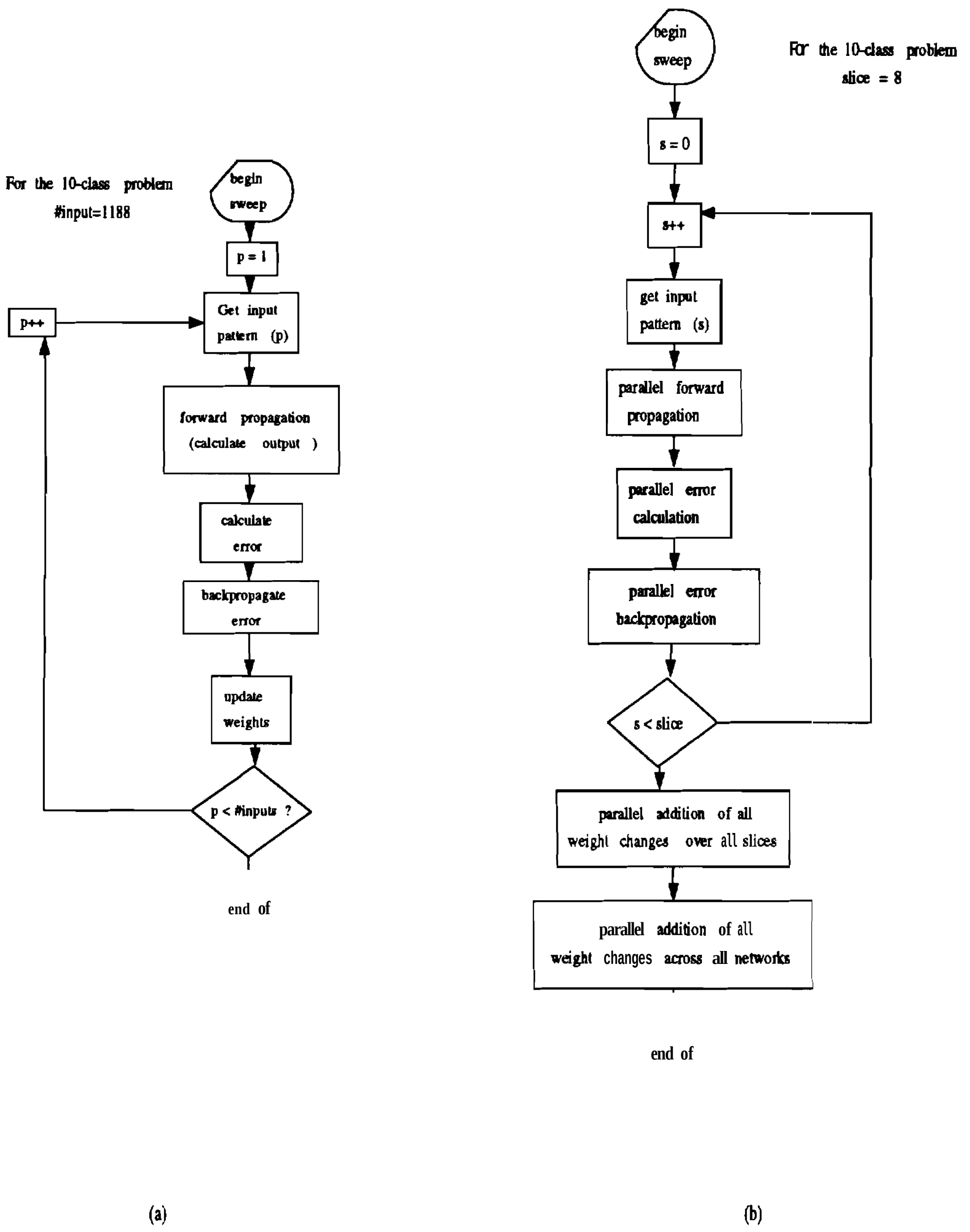

For the 10-class problem
#input=1188
begin sweep
p = 1
Get input pattern (p)
p++
forward propagation (calculate output)
calculate error
backpropagate error
update weights
p < #inputs ?
end of
(a)
For the 10-class problem
slice = 8
begin sweep
s = 0
s++
get input pattern (s)
parallel forward propagation
parallel error calculation
parallel error backpropagation
s < slice
parallel addition of all weight changes over all slices
parallel addition of all weight changes across all networks
end of
(b)

*Figure 5.5 Flow Charts of (a) Serial BP or Delta Rule Algorithm.
(b) SIMD-BP or SIMD-A.*

between the SIMD-BP version and the serial version. Let us assume a network with N output neurons in a problem with P training patterns. The total squared error defined for one training sweep is defined as

$$E = \frac{1}{2P} \sum_{p=1}^{P} \sum_{n=1}^{N} (d_n^p - o_n^p)^2. \tag{92}$$

Where d_n^p is the desired output value of the n^{th} output neuron for the p^{th} training pattern, anti the o_n^p stands for the actual output of the n^{th} neuron for the p^{th} training pattern.

Below, we first discuss the weight changes between the hidden and the output layers. Then, we describe the weight changes between the input and the hidden layer. The results can be easily generalized to more than one hidden layer. When there is no hidden layer, the first discussion is valid. Then, the hidden layer is the same as the input layer.

Using the chain rule we can find the rate of change of E with respect to w_{ij}, the weight connecting the j^{th} hidden neuron to the i^{th} output neuron, as

$$\frac{\partial E}{\partial w_{ij}} = \sum_{p=1}^{P} \frac{\partial E}{\partial o_i^p} \times \frac{\partial o_i^p}{\partial w_{ij}}. \tag{93}$$

where

$$\frac{\partial E}{\partial o_i^p} = -\frac{1}{P} (d_i^k - o_i^k). \tag{94}$$

We: assume a sigmoidal activation function in the form

$$o_i^p = \frac{1}{1 + e^{-\left[\sum_{m=1}^{M} x_m^p w_{im} + \theta_i\right]}}, \tag{95}$$

where M is the number of hidden neurons, and x_j^p is the j^{th} input to the output neuron, in other words, the output of the j^{th} hidden neuron. We get

$$\frac{\partial o_i^p}{\partial w_{ij}} = \frac{x_j^p \, e^{-\left[\sum\limits_{m=1}^{M} x_m^p w_{im} + \theta_i\right]}}{\left[1 + e^{-\left[\sum\limits_{m=1}^{M} x_m^p w_{im} + \theta_i\right]}\right]} = x_j^p o_i^p (1 - \text{of}). \qquad (96)$$

Using Eqs. (94) and (96) in Eq. (93) gives

$$\frac{\partial E}{\partial w_{ij}} = -\frac{1}{\text{P}} \sum_{p=1}^{P} x_j^p o_i^p (1 - o_i^p)(d_i^p - \text{of}). \qquad (97)$$

Therefore, using the gradient descent (steepest descent) algorithm [4], the weight change for w_{ij} is given by

$$\Delta w_{ij} = \text{p} \sum_{p=1}^{P} x_j^p o_i^p (1 - o_i^p)(d_i^p - \text{of}) \qquad (98)$$

where p is a small constant called the step size.

For the weights connecting the input layer to the hidden layer, the derivation is slightly more complicated. Let us assume that v_{jk} is the weight connecting the k^{th} input neuron to the j^{th} hidden neuron. Then, we have

$$E = \frac{1}{2P} \sum_{p=1}^{P} \sum_{n=1}^{N} (d_n^p - o_n^p)^2 = \frac{1}{2P} \sum_{p=1}^{P} \sum_{n=1}^{N} \left[d_n^p - \frac{1}{1 + e^{-\left[\sum\limits_{m=1}^{M} x_m^p w_{im} + \theta_i\right]}} \right]^2 \qquad (99)$$

where x_j^p is the output of the j^{th} hidden neuron for the p^{th} training pattern and is given by

$$x_j^p = \frac{1}{1 + e^{-\left[\sum_{k=1}^{K} i_k^p v_{jk} + \theta_j\right]}} \tag{100}$$

where K is the number of input neurons (ie. the length of the input pattern), and i_k^p is the kth bit* of the pth training pattern. Using the chain rule again, we get

$$\frac{\partial E}{\partial v_{jk}} = \sum_{p=1}^{P} \frac{\partial E}{\partial x_j^p} \times \frac{\partial x_j^p}{\partial v_{jk}}. \tag{101}$$

Using

$$\frac{\partial E}{\partial x_j^p} = \frac{1}{P} \sum_{n=1}^{N} (d_n^p - o_n^p) \frac{-w_{nj} e^{-\left[\sum_{m=1}^{M} x_m w_{nm} + \theta_n\right]}}{\left[1 + e^{-\left[\sum_{m=1}^{M} x_m w_{nm} + \theta_n\right]}\right]^2} = \frac{-1}{P} \sum_{n=1}^{N} w_{nj} o_n^p (1 - o_n^p)(d_n^p - o_n^p) \tag{102}$$

and from (96),

$$\frac{\partial x_j^p}{\partial v_{jk}} = i_k^p x_j^p (1 - x_j^p), \tag{103}$$

we get

$$\frac{\partial E}{\partial v_{jk}} = -\frac{1}{P} \sum_{p=1}^{P} i_k^p x_j^p (1 - x_j^p) \sum_{n=1}^{N} w_{nj} o_n^p (1 - o_n^p)(d_n^p - o_n^p). \tag{104}$$

The weight change for steepest descent is

* *In binary representation of t k input pattern, the k^{th} bit h as a value of 1 or 0, whereas in continuous number representation, this input is the k^{th} component on the analog input pattern vector.*

$$\Delta v_{jk} = \rho \sum_{p=1}^{P} i_k^p x_j^p (1 - x_j^p) \sum_{n=1}^{N} w_{nj} o_n^p (1 - o_n^p)(d_n^p - o_n^p). \tag{105}$$

In other words, the network has to calculate the weight changes due to all the training patterns, add them up and update the weights based on the total weight change accumulated over the entire sweep. In practice, however, the weight update in the serial implementation is performed after each training pattern (stochastic method). In other words, using (98) and (105) , the weight changes are computed as

$$\Delta w_{ij} = \rho x_j^p o_i^p (1 - o_i^p)(d_i^p - o_i^p) \tag{106}$$

anti

$$\Delta v_{jk} = \rho i_k^p x_j^p (1 - x_j^p) \sum_{n=1}^{N} w_{nj} o_n^p (1 - o_n^p)(d_n^p - o_n^p). \tag{107}$$

It can be shown that if the step size ρ is sufficiently small, the weight update can be performed after each pattern and reach a minimum of the error function E after a series of very small steps. While this approach is proved to work, its speed is very slow. Figure 5.6 shows the descent steps taken to move to the minimum of a paraboloid by the exact algorithm (weight batching) [1] and the approximate version (stochastic method) [1].

5.2.2 The SIMD-BP and the SIMD-A Algorithms

The SIMD-BP and the SIMD-A use the exact method, mainly because it allows data parallelism. In these algorithm we create, in parallel, as many networks as the number of

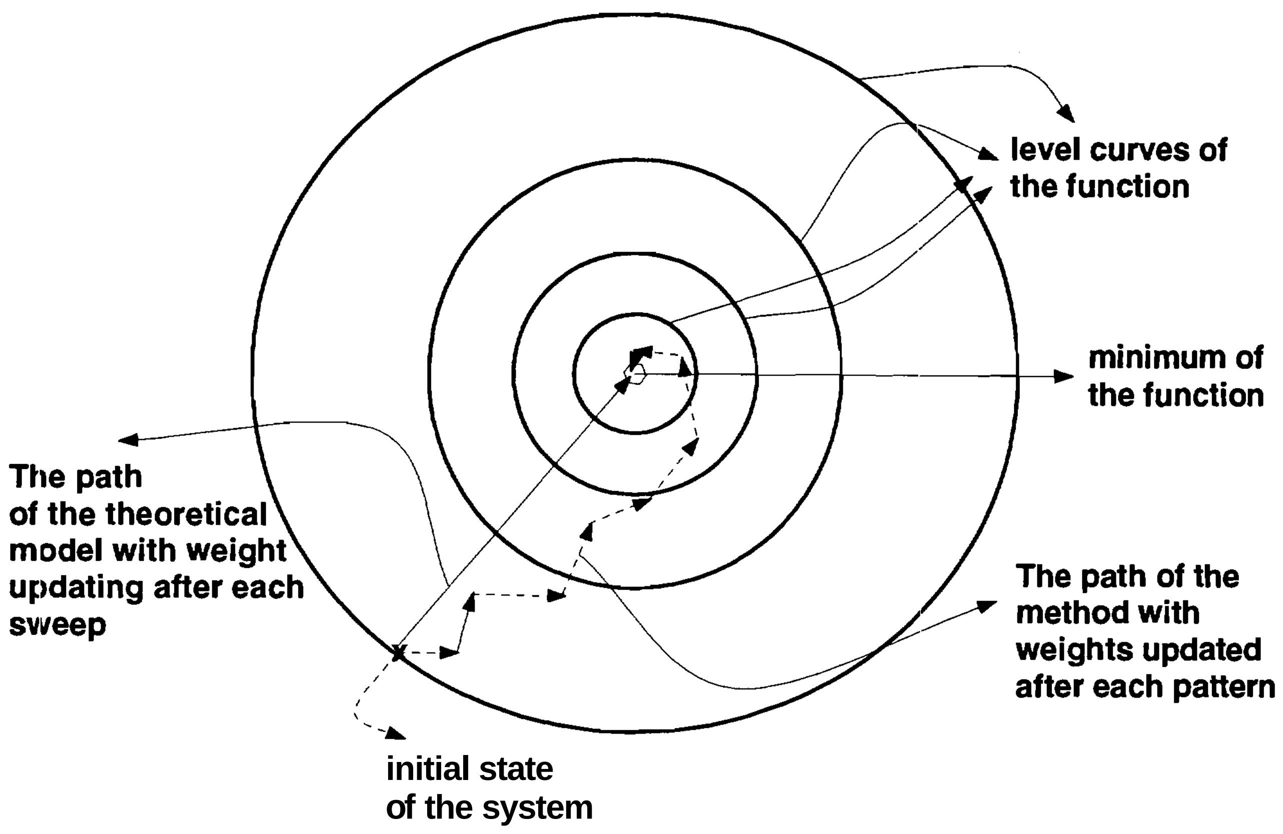

level curves of the function
minimum of the function
The path of the theoretical model with weight updating after each sweep
The path of the method with weights updated after each pattern
initial state of the system

Figure 5.6 The Descent Paths toward the Minimum of a Paraboloid Function for the Weight Batching Technique (Solid Line), and the Stochastic Technique (Dotted Line).

training patterns. Each network is given a training pattern and computes a weight change vector for all the weights in the network, based on its pattern. After the sweep is complete, these weight change vectors are added together using a very fast MP-1 library routine called ***reduceAdd***. Then, the weight vectors on all the networks are updated based on the total weight change vector. This vector is sent to all the PEs of MP-1 using the XNET structure.

The use of the exact algorithm results in data parallelism, and most of the speed-up achieved is due to this type of parallelism. Thus, the two types of parallelism utilized by the SIMD-BP are as follows:

- **Architectural Parallelism:** This parallelism is simply due to the parallel nature of the architecture of layered feed-forward networks. The computations performed in the neurons of the same stage can be performed all at the same time. Since there are no connections between the neurons of the same stage, no communication overhead is necessary*.

- **Data Parallelism:** As discussed above, most of the speed-up is due to data parallelism. Since the weight changes do not occur until after the sweep is over, there is no more data dependency between the operations performed for different patterns in the sweep. Consequently, these computations can be done in parallel. Therefore, we can simulate more than one network at a time and train each one to learn a different input pattern simultaneously. These networks all have the same initial random weights and, ideally, only one input pattern to learn. Each network calculates weight changes for its weights based on the input pattern and the desired output pattern it is assigned to. This is done for all the networks at the same time. After this step, the weight changes are accumulated from all the networks and the weights of all the networks are updated simultaneously, based on the accumulated weight changes from all the networks.

* *One could assign a PE to every neuron in the network. However, this does not bring a higher degree of parallelism than the case when there is only as many PEs assigned to the network as the number of neurons in the largest layer. This is due to the serial nature of the stages and the communication overhead required for communication between two layers.*

To better describe the SIMD training algorithms, we discuss the algorithm with the example of the 10-class remote sensing Colorado problem. This problem was described in Section 2.1.2. It involves classifying each input pattern into one of ten possible classes. The data set consists of 1188 patterns of length seven for training and 831 patterns for testing. Figure 5.7 shows the PE array of MP-1 in

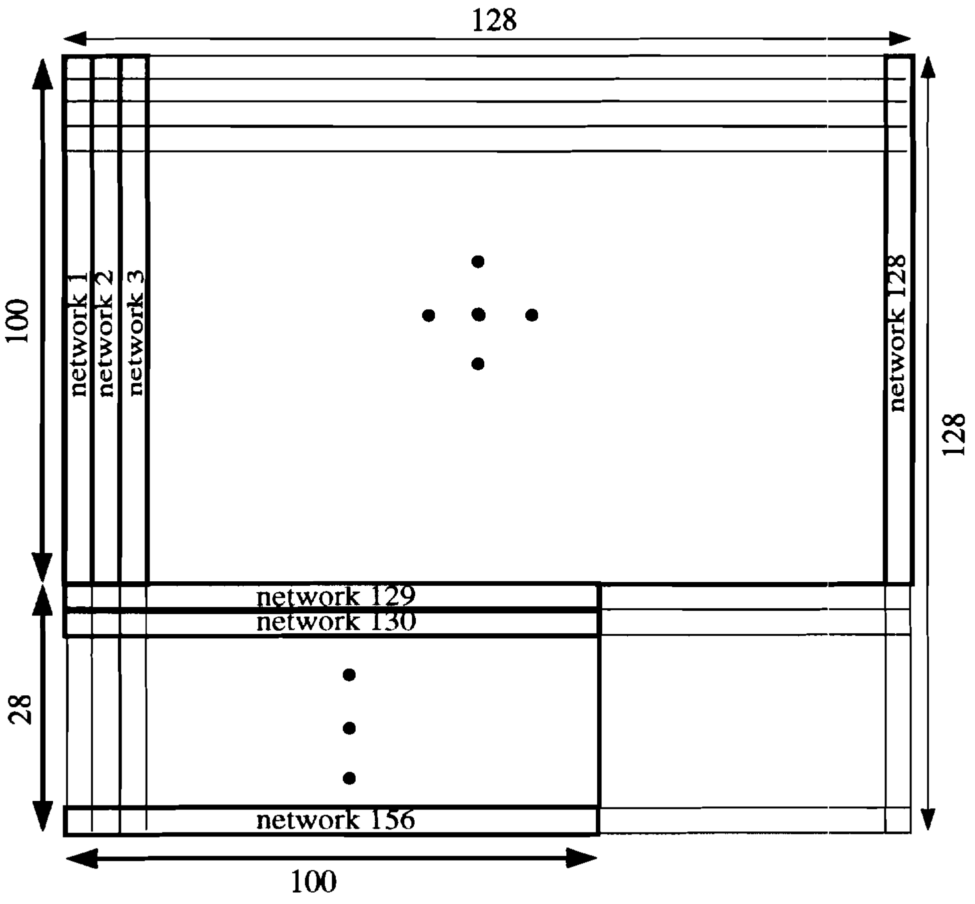


Figure 5.7 PE Array of MP-1 Partitioned for the Colorado Data Set for the 7-100-10 BP network. Each Network Learns only up to 8 Patterns of the Training Set.

a 128×128 grid array as it was arranged for this problem, using a 7-100-10 input-hidden-output neuron backpropagation network. Figure 5.8 shows the PE array of MP-1 arranged for the same problem for the 7-10 input-output delta rule network.

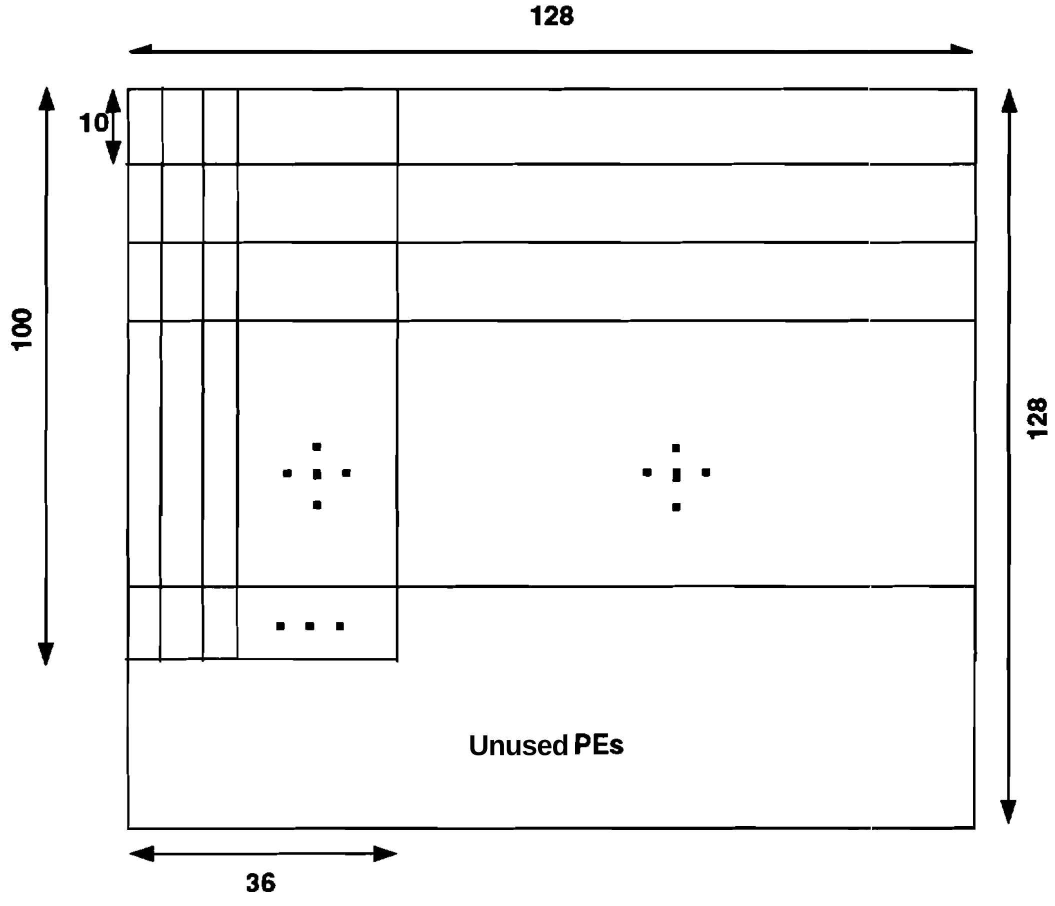


***Figure 5.8** PE Array of MP-1 Partitioned for the Colorado Data Set for the 7-10 Delta Rule network. Each Network Learns Only One Pattern of the Training Set.*

Figures 5.7 and 5.8 show the architectural parallelism for the Coloratlo data set. Each network is simulated by 100 PEs (10 in figure 5.8), which is the size of the hidden layer of the backpropagation network (size of the output layer of the delta rule network). These

100 (10) PEs first emulate the 100 (10) hidden (output) neurons of the network. In the case of the two-stage backpropagation network, once the calculations for the first stage art: performed, the output values of the 100 hidden neurons are communicated to the first 10 of the 100 PEs. Then, the remaining 90 are disabled and only the first 10 PE are active to emulate the output layer. Figures 5.7 and 5.8 also show the data parallelism for the Colorado data set. With the layout shown, the SIMD-BP and the SIMD-Δ learn 156 and 1188 patterns simultaneously, respectively.

It is important to keep in mind that the degree of parallelism achieved depends on the number of processors assigned to each network and the number of training patterns in the training set. For example, the 10-class Colorado problem has 1188 patterns in its training set and the number of PEs required for each backpropagation network is 100, where for the. delta rule network it is 10. Therefore, the maximum number of backpropagation networks running simultaneously is $\leq \frac{16384}{100} = 163$, where the maximum number of delta rule networks is $\leq \frac{16384}{10} = 1638$. For the simplicity of communication patterns, we chose to have only 156 backpropagation networks running simultaneously**.

For the SIMD-A there were only 1188 simultaneous networks, since: there were only 11138 patterns in the training set. Out of the 156 backpropagation networks, 94 were given 8 patterns and the remaining 62 were given 7 patterns ($7\text{x}62 + 8\text{x}94 = 1188$),

** *If we hod chosen 163 networks running simultaneously, loading the input patterns into the PEs correctly would become more difficult and the communication pattern among the PEs would have become irregular, which would have caused the PE-to-PE communication to be achieved in several serial steps rather than me parallel step.*

which gives a degree of virtualization of 7 (which is explained further below). The SIMD-Δ networks each received one pattern, making the degree of virtualization 0. Hence, at any given time, we are computing the weight changes for 156 different patterns in the SIMD-BP algorithm and 1188 in the SIMD-A. Figures 5.7 and 5.8 show the layout of the 156 backpropagation networks and the 1188 delta rule networks in the MasPar PE array.

In any parallel machine, the degree of parallelism is limited to the physical parallel resources of the machine. For example, in the MP-1 with 16K PEs, the maximum degree of parallelism achievable is 16384, since a maximum of 16384 operations can be run simultaneously at any given time. The real degree of parallelism for a given algorithm is normally much lower than the maximum degree possible. For example, in the Colorado problem, every backpropagation network required 100 PEs, thus allowing 156 parallel networks. In order to have one backpropagation network per training pattern, we ideally would have required 100x1188 = 118800 PEs. Since this many PEs were not available, we implemented a concept referred to as ***virtualization.***

The idea is similar to that of virtual memory, where one assumes that there is a much larger memory space than what the machine's physical resources offer. We assumed that 118800 PEs were arranged in a three dimensional PE grid array. The three-dimensional array is made of 8 layers ***(slices)*** of 128x128 PEs (Figure 5.9). Since: there is actually one: physical layer of PEs available, the PE array of MP-1 has to be programmed to emulate the layers of the 3-D grid serially. Thus we end up running 156 networks at a time, and at any given time the PE array is emulating a different layer of the virtualized

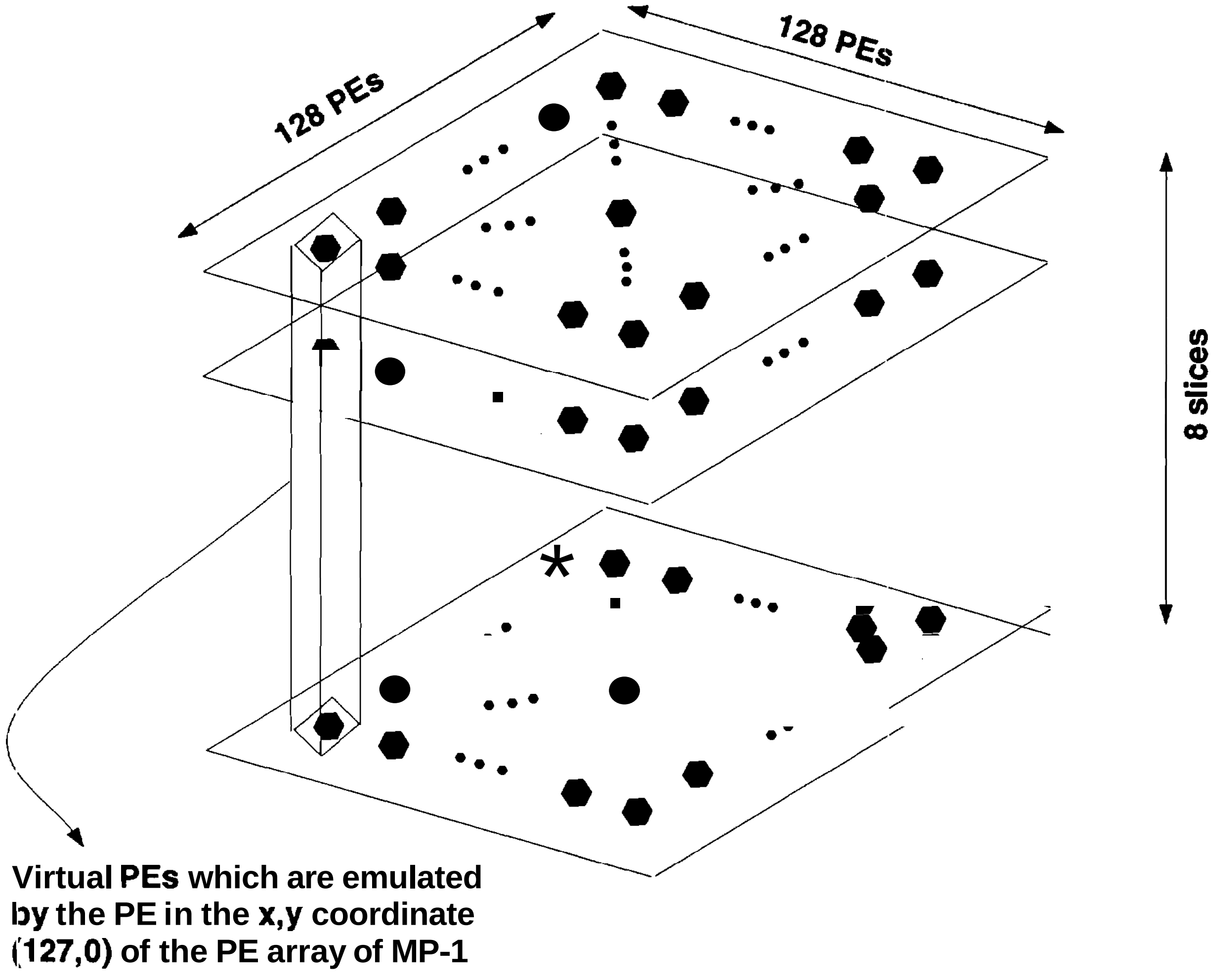

128 PEs
128 PEs
8 slices
Virtual PEs which are emulated by the PE in the x,y coordinate (127,0) of the PE array of MP-1

Figure 5.9 The *3-D* Virtual PE Array for the *10*-Class *Colorado* Data Set.

PE grid. Notice that the shift from one virtual array to another is done serially. In other words, the physical **PE** array has to process the first 156 networks before it can switch to the second batch. This serial portion of the algorithm is a "bottle neck" for the **throughput*** of the algorithm. This serial loop is eliminated in the SIMD-A case for the

* *By throughput we mean the part of the algorithm in which the output of the network for a given pattern is calculated.*

10-class Colorado problem because of the degree of virtualization of zero.

The data distribution among the PEs has to take the virtualization factor into account. Each PE receives the data for all the virtual PEs which it is assigned to emulate on all the virtual layers. Care must be taken in loading the data into the PEs, so that each PE receives only the data which the virtual PEs it is assigned to would have received. Also, the programmer must be careful about the fact that in the last slice there might not be enough data to require the services of the entire PE array. In this case, those PEs which have run out of data must be inactive for the computations of the last slice. Loading the data into the correct PEs was done using the ***PP-read*** and the ***xnetc*** constructs described later. These two parallel constructs are very efficient, making the cost of this preprocessing relatively small in relation to the actual cost of learning. Table 5.1 of Section 5.4 shows the average time required for loading and distributing training data in the case of the backpropagation networks with the virtualization degree of 7.

Another costly part of initiating the networks (backpropagation or delta. rule networks) is generating floating point random numbers for initial connection weights and distributing them among the PEs correctly. This procedure is so costly that storing some random values and loading them from a file should be considered. To generate the random numbers, we used a random vector generator routine from the MasPar mathematics library called ***fp_veyran***, which generates a Y-oriented random vector and stores its elements in the first column of the MP-1 PE array. To distribute the weights among all the networks, we again used the ***xnet*** constructs. Table 5.1 shows the average time required for this task for the SIMD-BP.

Figure 5.5 shows the block diagram of the serial and the SIMD version of backpropagation or delta rule algorithm. The SIMD-BP and the SIMD-Δ programs are designed to arrange the PE array to achieve the minimum degree of virtualization thereby achieving the maximum degree of parallelism. They are written in such a way that they detect and adjust to the size of any given problem automatically. For this purpose, the program considers two parameters: 1- The size of the largest layer of the network, 2- the number of training patterns. For example, for a classification problem with 500 training patterns and a network with 10-20-5 input-hidden-output neurons, the program requires no virtualization (virtualization degree of zero). Figure 5.10 shows the PE array arrangement for this problem. The remaining part of the SIMD-BP takes the degree of virtualization (slice) and a parameter called offset into account. The *offset* is the number of PEs in the last slice which still have data and which should be kept active for the calculations of that slice. The program then performs the operations of each slice separately. It first deactivates the PEs not required for that slice and then has the ACU decode the instructions and send them to the PEs, which in turn perform the operation if their enable flag is high. The SIMD-BP and SIMD-A programs are thereby written in such a way that they detect and adjust to the size of any given problem automatically.

Figure 5.7 shows how the backpropagation networks are organized in the MP-1 implementation for the Colorado problem. The first 128 networks were chosen in a vertical layout fashion and the remaining 28 in the horizontal layout fashion. This produces the simplest communication pattern. An inverse layout pattern (first 128 horizontal and the rest vertical), would result in additional communication overhead to

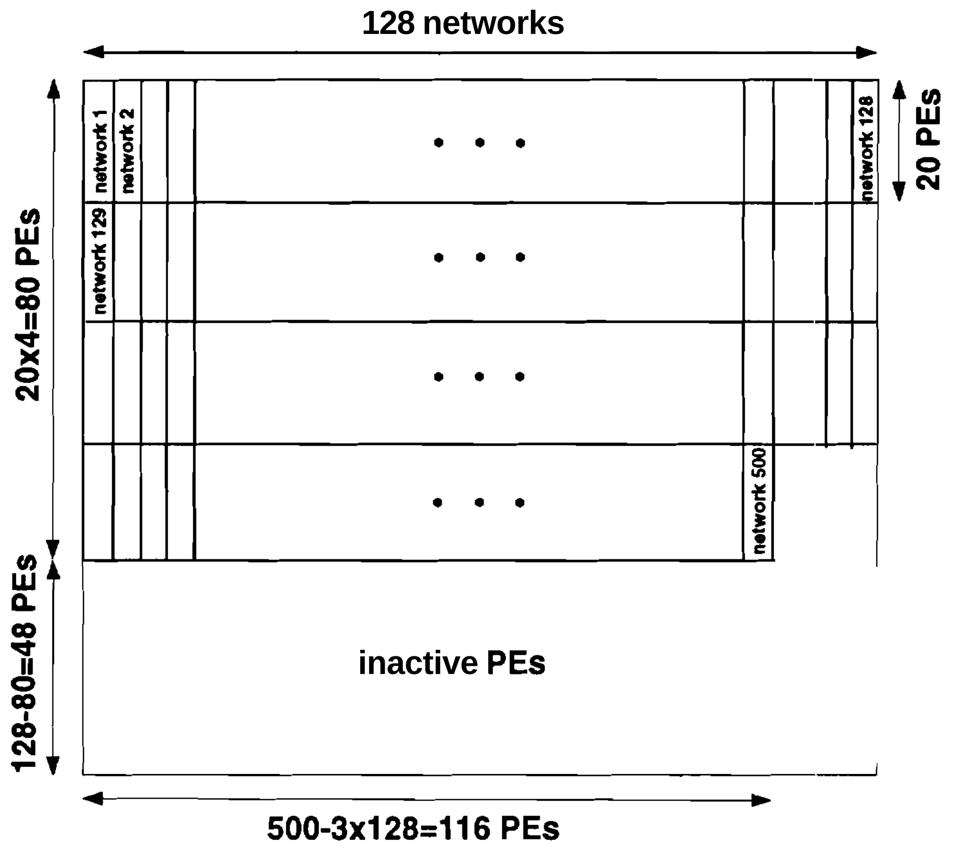


Figure 5.10 The PE Arrangement for a 2-Stage Backpropagation Network with the largest layer of size 20 for a Problem with 500 Trainiing Patterns.

distribute the input patterns to all the PEs in each network. Further speed-up can be achieved by assigning 10×10 square of PEs to each network instead of a 100×1 array of PEs. This results in communication paths with maximum length of 10, instead of 100. At the cost of a more complicated communication pattern, this could result in a slight speed-up.

The way the networks are organized is such that the first PE in all the networks can easily be enabled. The input patterns are loaded into the first PEs of the networks using the parallel read command [12]:

```
cc = p_read(d, buf, nbytes)
plural int cc;
int d;
plural char *buf;
int nbytes;
```

This command was used in the following format:

```
if (iyproc==0) || ((iyproc>=hn)&&(ixproc==0)))
   Fstatus=p_read(fd, &x[slice][0], invecbt);
```

The if statement enables the first PE of each network (Figure 5.7). *ixproc* and *iyproc* are the x and the y coordinates of each PE, respectively, in the 128×128 PE array. *hn* is the size of the hidden layer (in this case 100). *invecbt* is the size of the input vector in bytes, and *slice* is the degree of *virtualization*. Notice that the entire input vector is read into the first PE in one shot.

After the loading of input data, the first PEs proceed to communicate the data to the rest of the PEs in their networks. This communication uses the *xnetc* command [12]. The xnetc command was used as follows:

```
if (iyproc==0)
   xnetcS[hn-1].x[slice][i] = x[slice][i];
if( (ixproc==0) && (iyproc >= hn))
   xnetcE[hn-1].x[slice][i] = x[slice][i];
```

The if statements enable the first PEs of the networks. The letters "*S*" and "E' specify the direction in which data should be sent (South and East). *hn-1* is the step size, which means "send $100 - 1 = 99$ PEs to the south or east". Notice that since *xnetc* is used, a copy of the communicated data is left in each relaying PE memory at the right location.

The forward calculation of data also requires some communication which uses *xnetp* and *xnetc*. To calculate the total AW (the change in the weight matrix), we used two library routines from MP-1's mathematics library MPML [14]. These two routines are:

```
void fp_matsumtovex ( ny, nx, B, nxB, yoffB, xoffB, VX )

int ny, nx, nxB, yoffB, xoffB;
plural float *B, *VX;
```

and

```
void fp_matsumtovey ( ny, nx, B, nxB, yoffB, xoffB, VY )

int ny, nx, nxB, yoffB, xoffB;
plural float *B, *VY;
```

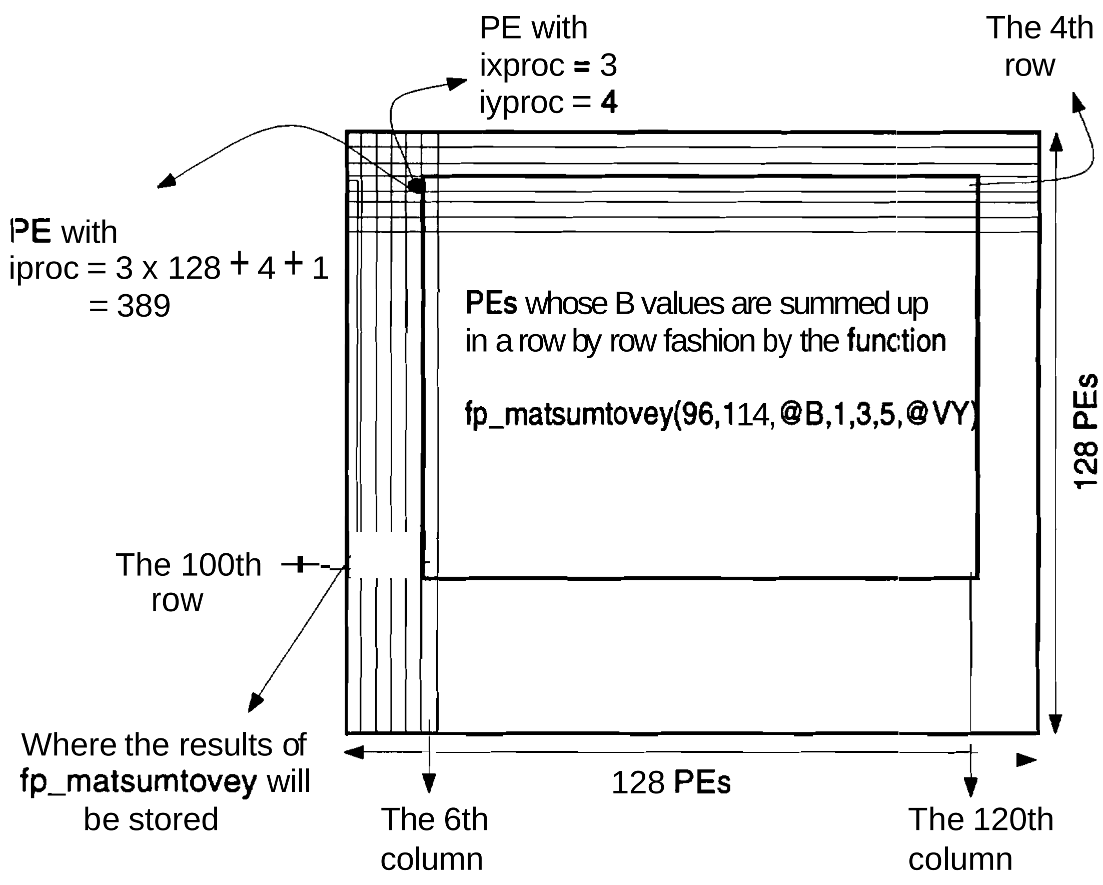
The first routine adds the columns of the matrix B starting from row *yoffB* and column *xoffB* for *ny* rows and *nx* columns and puts the results in the x-oriented vector *VX*. The second routine adds the rows of this submatrix and puts the results in the Y-oriented *VY* vector.

For example, one could use the **fp_matsumtovey** library routine to add the processor numbers *(iproc*)* assigned to each processor row by row from the 4th row to the 100th row, and from the 6th **PE** in each row through the 120th **PE** in that row, and put the sum values in a Y-oriented vector in the 0th column of the **PE** array. The steps to perform this operation are as follows:

```
1  plural float B, VY;
2  B = (plural float) iproc;
3  fp_matsumtovey( 96, 114, @B , 1 , 3 , 5 , @VY );
```

In statement 1, the variables B and VY are declared across all processors. In statement 2, the iproc value of each **PE** is assigned to the variable B of that **PE.** In statement 3, the **fp_matsumtovey** function is used to add the values of the B variables in each row from the 4th to the 100th row, and each row from the 6th element to the 120" element, and put the result of

* *In the PE array of MP-I each PE can be identified in two ways. First way is to identify the row number ixproc* **and the column number** ***iyproc*** **of the PE in the two dimensional PE grid array. The second way is to identify the processor number** ***iproc*** **of the PE (see Figure 5.11). Where** ***iproc=ixproc×nxproc+iyproc+1*** **and** ***nxproc*** **is the number of PEs in a row (in 16K machine, 128). Therefore the expressions** ***proc[3][4].B*** **and** ***proc[389].B*** **are equivalent and both point to the value of the variable B of the PE in the 4th row and the 5th column.**



PE with
ixproc = 3
iyproc = 4
The 4th
row
PE with
iproc = 3 x 128 + 4 + 1
= 389
PEs whose B values are summed up in a row by row fashion by the function
fp_matsumtovey(96,114,@B,1,3,5,@VY)
128 PEs
The 100th
row
Where the results of fp_matsumtovey will be stored
128 PEs
The 6th
column
The 120th
column

Figure 5.11 An Example of the Operation of the fp_matsumtovey Routine.

each row in the VY variable of the first PE of that row (see Figure 5.11).**

The backward propagation of error and updating the weights uses the same routines in the reverse direction of the network.

** The number of PEs in the Y direction ny=100–4=96
The number of PEs in the X direction nx=120–6=114
The starting row $yoffB$=4–1=3; the first PE in each row is the 0^{th} PE
The starting PE number in every row $xoffB$=6–1=5; the first PE in each row is the 0^{th} PE

5.3 Time Complexity Analysis

In this section, we will analyze the time complexity of PPSHNNl, PPSHNN2, and their respective parallel versions. Since training takes much longer than testing, we only concentrate on the time complexity of the respective training procedures.

5.3.1 The PPSHNN1 and The SIMD-PPSHNN1 Algorithms

The PPSHNN1 consists of several two-stage networks. A few examples of these networks are: the first N-unit created for the first module, the P-unit created for the first module (if necessary), the reduced N-unit for the first module (if a P-unit was created for that module), the N-unit network for the second module, the P-unit created for the second module (if necessary), the reduced N-unit for the second module (if a P-unit was created for that module), etc.

Over 90% of the training time of PPSHNNl is spent on training these networks. The time required for the statistical analysis of the S-units, and overhead operations required for self-organization is less than 10% of the total training time. It is also important to keep in mind that all these networks are equal to or smaller in size than the first N-unit created for the first module. Also, the number of patterns with which they are trained is less than that of the first N-unit created for the first module. Therefore the time required for their training is less than the training time of the first N-unit network created for the first module. For this reason we get

$$T_{PPSHNN1} = \theta\left[T_{BP}\right] \tag{108}$$

where $T_{PPSHNN1}$ is the time complexity of the PPSHNNl network and T_{BP} is the training time complexity of the first backpropagation network created. With the same argument,

$$T_{SIMD-PPSHNN1} = \theta\left[T_{SIMD-BP}\right] \quad (109)$$

For this reason, we first analyze the time complexity of the serial backpropagation BP and the parallel version SIMD-BP algorithms for a two-stage feed-forward network.

Since the time taken to perform floating point addition, multiplication, and exponentiation is a good indication of the time required by the training procedure, we estimate the number of such operations performed in each type of training procedure.

The Serial BP Algorithm:

Let us denote the number of input neurons to the network with n_i, the number of hidden neurons with n_h (assuming one hidden layer in the network), the number of output neurons with n_o, and the number of training patterns in the training set with P. Since, in the first stage, a backpropagation network has to perform one multiplication for every connection, we get $n_i \times n_h$ floating point multiplications for the first stage. To add the incoming signals to each neuron and subtract the result from a threshold [1], we need $n_h \times n_i$ floating point additions for the first stage. In the same way, we can find $n_h \times n_o$ floating point multiplications, and $n_o \times n_h$ floating point additions for the second stage. Therefore we get a total of $n_h \times \left[n_i + n_o\right]$ floating point multiplications, and $n_h \times \left[n_i + n_o\right]$ floating point additions. We also require a total of $n_h + n_o$ floating point exponentiation for the two stages.

Let us denote the time required for a floating point addition by a, the time needed for a floating point multiplication by β, and the time required for a floating point exponentiation by γ. Since the error backpropagation through the net and weight changes require the same order of floating point additions, multiplications, and exponentiation as forward propagation, and since this procedure is repeated P times, once for each pattern, the time complexity of the backpropagation network becomes

$$T_{BP} = O\left[P\, n_h \left[n_i + n_o\right] \left[\alpha + \beta\right] + P \left[n_h + n_o\right] \gamma\right] = O\left[P n_h \left[n_i + n_o\right]\right]. \tag{110}$$

Since n_i is $O\left[n_o\right]$ for the Colorado problem, we get

$$T_{BP} = O\left[P\, n_h\, n_o\right]. \tag{111}$$

The SIMD-BP algorithm:

To calculate the time complexity of the SIMD backpropagation algorithm, in addition to the time required for floating point additions and multiplication, we have to consider the communication overhead. Let us first consider the additions, the multiplications and the exponentiation. Since in SIMD-BP all the neurons of each stage operate in parallel, we only need n_i multiplications, n_i additions, and 1 exponentiation for the first stage and n_h multiplications, n_h additions, and 1 exponentiation for the second stage. Thus, the computation time needed to process one pattern is on the order of $\left[n_i + n_h\right] \times \left[\alpha + \beta\right] + 2 \times \gamma$. Since the communication overhead is on the order of the length of a side of the PE array which is 128, the communication overhead is on the order of $nyproc \times C$, where C is the time it takes to communicate a float value from one PE to its immediate neighbor, and ***nyproc*** is the length of the PE array in the y direction (*nyproc* =128).

Thus, we get

$$T_{SIMD-BP} = O\left(\left(\left(n_i + n_h\right)\left(\alpha + \beta\right) + 2\,\gamma + nyproc \times C\right) slice\right), \tag{112}$$

where $slice = \left\lceil \frac{P \times n_h}{N} \right\rceil$, is the degree of virtualiration and N is the number of PEs in the MP-1 PE array. Because both n_i and n_h are $O\left(nyproc\right)$, we can right

$$T_{SIMD-BP} = O\left(\frac{P\ n_h\ \left(\left(n_i + n_h\right) + nyproc\right)}{N}\right) = O\left(\frac{P\ n_h\ nyproc}{N}\right) \tag{113}$$

and since N $= \boldsymbol{nyproc}^{\mathbf{2}}$ we get

$$T_{SIMD-BP} = O\left(\frac{P\ n_h}{\sqrt{N}}\right). \tag{114}$$

Therefore, by using equations ***(108)*** and *(111)* we can write:

$$T_{PPSHNN1} = \theta\left(T_{BP}\right) = O\left(P\ n_h n_o\right), \tag{115}$$

and using ***(109)*** and ***(114)*** gives:

$$T_{SIMD-PPSHNN1} = \theta\left[T_{SIMD-BP}\right] = O\left[\frac{P\, n_h}{\sqrt{N}}\right]. \tag{116}$$

5.3.2 The PPSHNN2 and SIMD-PPSHNN2 algorithms

The PPSHNN2 which implements PNS modules, uses delta rule networks. This means removing the hidden layer(s) of the backpropagation network. Then, there are just the input and the output layers. The derivations of the Equations (92) through (98) still apply. The error function is defined as in (92) and the gradient descent algorithm results in the weight change of

$$\Delta w_{ij} = \rho \sum_{p=1}^{P} x_j^p o_i^p (1 - o_i^p)(d_i^p - o_i^p) \tag{117}$$

as before. Since there are no hidden layers, this weight change equation applies to all the weights in the network.

Similar to the argument for the PPSHNN1, we can show that most of the time required for the training of a PPSHNN2 network is spent on training the neural. network modules which are chosen to be single stage delta rule networks. Hence, we can write

$$T_{PPSHNN2} = \theta\left[T_{\Delta}\right] \tag{118}$$

where $T_{PPSHNN2}$ is the training time complexity of the PPSHNN2 network, and T_{Δ} is the training time complexity of the first delta rule network created. With the same argument,

$$T_{SIMD-PPSHNN2} = \theta\left[T_{SIMD-\Delta}\right] \tag{119}$$

For this reason, we first analyze the time complexity of the serial delta rule algorithm which we denote with A, and its parallel version SIMD–A. Like before, we take the time needed to perform floating point addition, multiplication, exponentiation, and the communication overhead in the parallel case as a measure of the time required for the training procedure.

The Serial Delta Rule Algorithm:

Since there is no hidden layer in the two-layer network, the number of PEs assigned to each network on the MP-1 PE grid depends on the number of neurons in the output layer of the network. This is determined by the coding scheme used for output.

As before, we denote n_i to be the number of input neurons, n_o the number of output neurons, and P the number of training patterns in the training set. Since there are two layers of neurons, there is only one stage of connections between the layers. In this stage, the ***delta rule*** performs one multiplication for every connection (hence $n_i \times n_o$ floating point multiplications), $n_i \times n_o$ floating point additions to add the incoming signals to the output neurons and subtract them from a threshold, and n_o exponential operations.

If, as before, we denote the time required to perform a floating point addition, multiplication, and exponentiation by α, β, and γ, respectively, the time complexity of a serial delta rule network can be estimated as

$$T_\Delta = O\left[P\, n_i\, n_o\left[\alpha+\beta\right] + P\, n_0\, \gamma\right], \tag{120}$$

or.,

$$T_\Delta = O\left[P\, n_o\, n_i\right] \tag{121}$$

The SIMD Delta Rule Algorithm:

Similar to the case of networks with hidden layers, in addition to the: time required for floating point addition and multiplication, the communication overhead also has to be taken into account in the parallel algorithm. For this purpose, as before, the value C is introduced as the time required for a floating point value to be sent from a **PE** to its immediate neighbor.

Since the operations in the stage are performed in parallel, there are only n_i floating point multiplications, n_i floating point additions, and 1 floating point exponentiation. Thus, the total time required for the additions and multiplications and exponentiations needed for the computations of one pattern is $n_i \times \left[a+\beta\right]+\gamma$. Since the **PE** array is *nxproc* x nyproc, which is 128 x 128 in the 16K machine, the communication overhead is at most on the order of C x nyproc. Therefore, the time complexity can be estimated

$$T_{SIMD-\Delta} = O\left[slice\left[n_i\,[\,a+\beta\right] + C\; nyproc + \gamma\right]\right] \tag{122}$$

where $slice = \left\lceil \frac{P \times n_o}{N} \right\rceil$ is, as before, the degree of virtualization and N is the number of PEs in the MP-1 PE array. Also, because n_i is $O\left[nyproc \right]$ we can write

$$T_{SIMD-\Delta} = O\left[\frac{P\ n_o\ nyproc}{N} \right] = O\left[\frac{P\ n_o}{\sqrt{N}} \right] \tag{123}$$

The PPSHNN2 and SIMD-PPSHNN2 Algorithm: Again by using equations (118), (119), (121), and (123), we can estimate the time complexity of the A and the SZMD-A algorithms as follows:

$$T_{PPSHNN2} = \theta \left[T_{\Delta} \right] = O\left[P\ n_o n_i \right], \tag{124}$$

and

$$T_{SIMD-PPSHNN2} = \theta \left[T_{SIMD-\Delta} \right] = O\left[\frac{P\ n_o}{\sqrt{N}} \right]. \tag{125}$$

5.4 Speed-Up Analysis

In this section, we compare the order of theoretical speed up and the actual speed up achieved in our experiments for the PPSHNN1 network with two-stage backpropagation networks, and the PPSHNN2 network with single stage delta rule networks.

The actual speed up comparison is made between the run time of each algorithm on a Sun 3/60 station and its respective SIMD version on MasPar MP-1 with 16K PEs.

It is important to mention here that the actual speed-up factor achieved in experiments embodies both parallel speed-ups and hardware differences in the floating point units of the two systems. The floating point co-processor in the sun system is a full blown floating point unit, whereas the floating point units of the MP-1 have 4-bit ALUs and most of their operations are performed by table look-ups. In addition, in MP-1 the floating point units are shared among the PEs of a PE cluster. Therefore, not every PE has access to a floating point unit at all times. Despite all the hardware differences, our experiments show that the overall floating point capabilities of a MP-1 PE and of the Sun 3/60, for most applications, are comparable.

PPSHNN1: The order of estimated speed-up is to be measured by $\frac{T_{PPSHNN1}}{T_{SIMD-PPSHNN1}}$

Equations (111) and (112) give

$$\frac{T_{PPSHNN1}}{T_{SIMD-PPSHNN1}} = O\left[\frac{P\, n_h\, n_o}{\frac{P\, n_h}{\sqrt{N}}}\right] = O\left[n_o\, \sqrt{N}\right] \tag{126}$$

For example, in the 10-class Colorado remote sensing problem, we have: $n_i = 7$, $P = 1188$, $n_h = 100$, $n_o = 10$, $slice = 8$. For this problem run on the MP-1 with N = 16384 PEs, we get

$$n_o\, \sqrt{N} = 10 \times \sqrt{16384} = 1280.$$

In our experiments with backpropagation on a Sun 3/60 work station, each sweep of training for the 10-class problem takes an average of approximately 7 minutes and 30 seconds. On **MasPar,** on the other hand, every **100** sweeps takes an average of approximately **14** seconds. This results in a speed-up factor in this **particular** case equal to

$$\frac{\left[7\times 60+30\right]\times 100}{14}=3214.$$

Figure **5.12** shows the run times for different size hidden layers of the SIMD-BP.

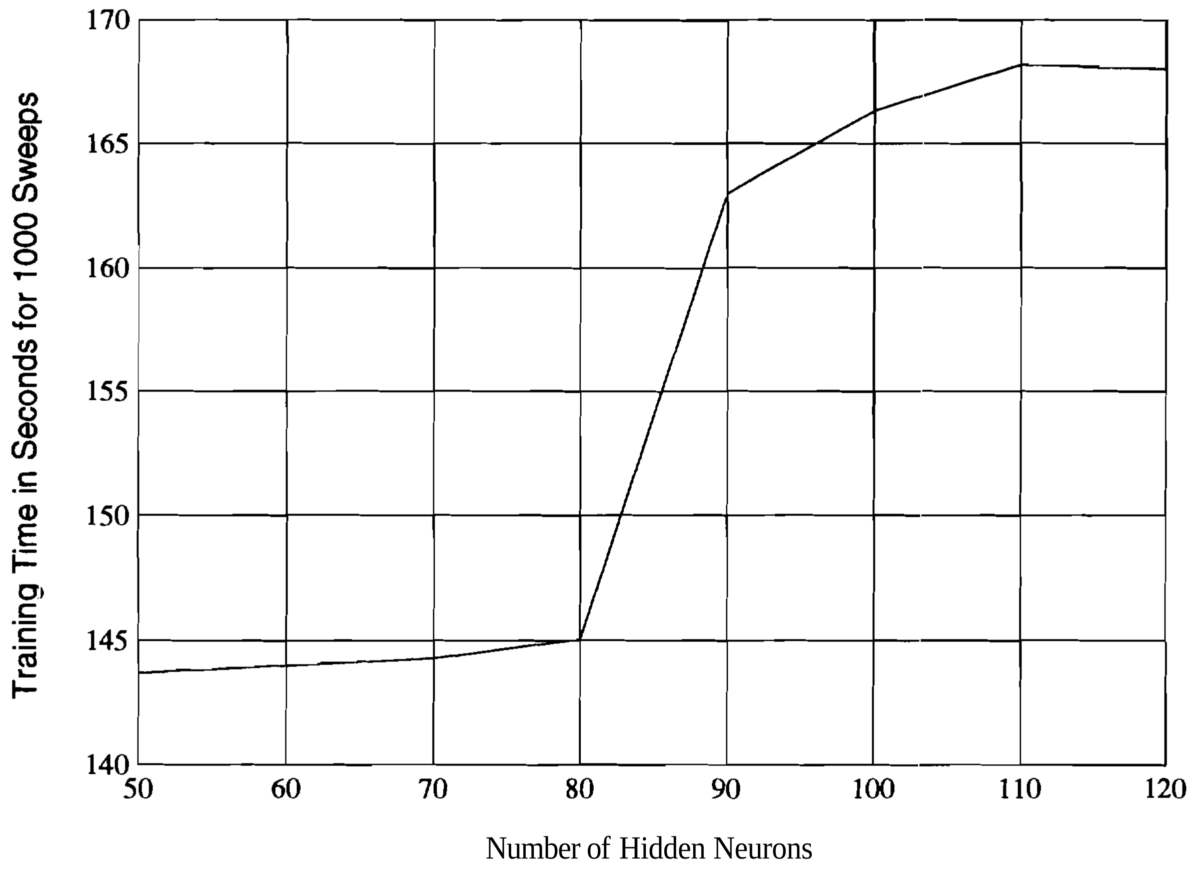


Figure **5.12** SIMD-BP Run Times for Networks with 7 Input Neurons and 10 Output Neurons for the Colorado Data Set **with** 1188 Training Patterns.

The: relatively big jump in the training time between the 80 and 90 hidden neuron networks is due to the addition of another slice to the virtual PE array, which increases the degree of **virtualization** by one.

Table 5.1 shows some time indexes for the 100 hidden neuron network, which performed the best classification for the

Table 5.1 Actual Time Indexes for Various Parts of the SIMD-BP Algorithm.

		first stage	second stage	[illegible]
throughput	best time	73.12 MCS 0.013001 sec. / sweep	19.26 MCS 0.062314 sec. / sweep	28.55 MCS 075315 sec. / sweep 0.
	worst time	73.07 MCS 0.0130064 sec. / sweep	19.25 MCS 0.0623242 sec. / sweep	28.54 MCS)753306 sec. / sweep 0.[illegible] sec. / sweep
weight update	best time	186.94 MCUPS 0.005084 sec. / sweep	39.47 MCUPS 0.030401 sec. / sweep	60.60 MCUPS 0.035485 sec. / sweep
	worst time	186.72 MCUPS 0.00509008 sec. I sweep	39.33 MCUPS 0.0305117 sec. / sweep	60.40 MCUPS 0.035611 sec. / sweep

loading and distributing training date	0.2364111	second for 1188 paterns
	5025.15	patterns / second
loading and distributing desired data	0.0410522	second for 1188 paterns
	28938.77	patterns / second
generating and distributing random weights	0.449686	seconds for 1810 connections
	4025.03	connections / second

10-'class problem. For this problem, the SIMD-BP algorithm reached a peak performance of 0.013001 seconds for calculating the throughput of the first stage (800 connections) for all the patterns in one sweep (1188 patterns). This is equivalent to 73.12 MCS (Million Connections per Second). The worst performance for the first stage was observed at 73.07 MCS, or 0.0130064 seconds for a sweep. Notice that the times mentioned for the first stage also include the floating point exponentiation required for the activation functions of the hidden neurons. The best performance of the second stage (1010 connections) was 19.26 MCS, or 0.062314 seconds for a sweep. The worst performance for this stage was observed at 0.0623242 seconds per sweep, or 19.25 MCS. The times for the second stage include the exponentiation required for the activation function of output neurons and the communication overhead to communicate the output of the hidden layer to the input of the output layer. For the weight update of the first stage we achieved a peak performance of 0.005084 seconds per sweep, or 186.94 MCUPS (Million Connection UPdates per Second), while the worst performance was 186.72 MCUPS, or 0.00509008 seconds per sweep. For the second stage, the peak performance was 39.47 MCUPS, or 0.030401 seconds per sweep, while the worst speed was 0.0305117 seconds per sweep or 39.33 MCUPS. The times for the second stage also include the communication overhead for the backpropagation of the partial errors to the first stage.

PPSHNN2: Similar to the PPSHNN1 case, the order of the theoretical speed-up of the parallel PPSHNN2 algorithm can be estimated by the ratio $\frac{T_{PPSHNN2}}{T_{SIMD-PPSHNN2}}$. Using equations (120) and (121) this ratio becomes

$$\frac{T_{PPSHNN2}}{T_{SIMD-PPSHNN2}} = O\left[\frac{P\,n_o\,n_i}{\frac{P\,n_o}{\sqrt{N}}}\right] = O\left[n_i\,\sqrt{N}\right]. \qquad (127)$$

For the example of the 10-class Colorado problem with n_i=7, n_o=10, P=1188, and $slice$=1, and a MP-1 array size of N=16384, we get

$$n_i\,\sqrt{N} = 7 \times \sqrt{16384} = 896\,.$$

The actual speed up in our experiments between the serial and the parallel versions of the PPSHNN2 algorithm run on Sun 3/60 and MP-1 respectively was measured as follows:

The serial algorithm takes approximately 19 seconds to complete one training sweep. The parallel algorithm running on MP-I takes an average of 1.75 seconds for every 100 training sweeps. This results in a speed up factor in this case equal to

$$\frac{19\times100}{1.75} = 1086.$$

Table 5.2 shows some time indexes for the PPSHNN2 network running on MP-1 for the 10-class Colorado problem.

For this problem, the $SIMD-\Delta$ algorithm reached a peak performance of 0.001625 seconds for calculating the throughput of the network (80 connections) for all the patterns in one sweep (1188 patterns). This is equivalent to 58.48 MCS. The worst performance for the first stage was observed at 58.46 MCS, or 0.0001.626 seconds for a sweep.

Table 5.2 Actual Time Indexes for Various Parts of the SIMD-A Algorithm.

		network
throughput	best time	58.48 MCS 0.001625 sec. / sweep
	worst time	58.46 MCS 0.001626 sec. / sweep
weight update	best time	149.55 MCUPS 0.0006355 sec. / sweep
	worst time	149.37 MCUPS 0.00063626 sec. / sweep

For the weight update of the network, we achieved a peak performance of 0.00063550 seconds per sweep, or **149.55** MCUPS, while the worst performance was **149.37** MCUPS, or **0.00063626** seconds per sweep.

As we see, while the first stage of the backpropagation network: achieves higher throughput and update rate than the delta rule network, as a whole, the backpropagation network performs slower than the delta rule network **(28.55** MCS versus **58.48** for throughput and **60.60** MCUPS versus **149.55** MCUPS for weight update). This is due to the much slower second stage of the backpropagation network. Much of this slow-down, compared to the first stage of the network, is due to the communication overhead required to communicate the output of the first stage to the PEs responsible for the output layer.

As we see, the weight update performances for both networks are about twice their respective throughput performances. This is unusual since updating the weights is much more computationally intensive than throughput. For weight update, one must find the gradient of the error function in order to find the steepest descent path. The evaluation of the following expression,

$$\frac{\partial o_i^p}{\partial w_{ij}} = \frac{x_j^p\, e^{-\left[\sum_{j=1}^{M} x_j^p w_{ij} + O_i\right]}}{\left[1 + e^{-\left[\sum_{j=1}^{M} x_j^p w_{ij} + O_i\right]}\right]^2}, \tag{128}$$

which is computationally more intensive than the computations involved with the throughput is necessary for the calculation of the steepest descent path. This expression, however, can be written as

$$\frac{\partial o_i^p}{\partial w_{ij}} = x_j^p o_i^p (1 - of'). \tag{129}$$

We see that all the components of this expression are either given or have been calculated during throughput. Thus, there is no need to recalculate these partial results. By using their values from the throughput stage, we can avoid floating point exponentiation as well as most other floating point operations. This produces the speed-up factor observed during weight update.

5.5 Parallel Testing

The procedure of parallel testing of the network is similar to that of training except that during the throughput the hierarchy of the modules can be ignored. Thus, all the P- and NS- units are implemented in parallel. All the P- and NS-units receive the incoming pattern, and based on their respective trainings, they perform classification. The result of this classification is interpreted differently from unit to unit. For example, the output of a P-unit is interpreted as accept or reject, whereas the output of a NS-unit is either classified into one of the classes which the unit was trained with, or it is classified as reject. If a P-unit and its S-unit classify a pattern as accept, the classification of the succeeding modules in the hierarchy are ignored. In this case only the classification of the NS-unit(s) corresponding to that P-unit matters. If a P-unit and/or S-unit classifies the pattern as reject, the classification of the module is disregarded, and the classification of the succeeding module is considered. Notice that, similar to training, depending on the size of the PE array and the number of PEs required to simulate the parallel network, several patterns are classified at the same time. Hence, the two types of data parallelism and architectural parallelism also exist in the testing procedure.

As an example, Figure 5.13 demonstrates the network developed for the 10-class Colorado problem. We have marked the P-unit of the first and second modules as PI and *PII*. The P-units within the NS-units are marked p1, p2, p3, etc. The: NS-units are also numbered in this manner. Figure 5.14 shows the division of the MP-1 PE array for the testing/recall of this network. As shown, the networks are simulated by columns of PEs. This arrangement results in the simplest communication pattern for distributing the

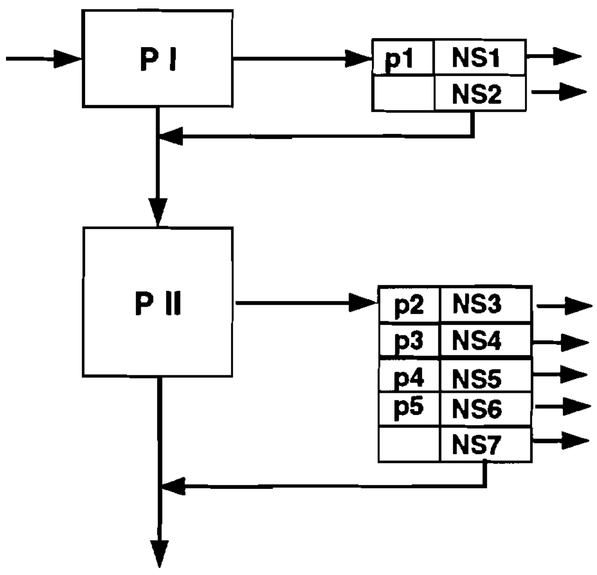


Figure 5.13 The PNS Block Diagram for the 10-Class Problem.

patterns. As we see, for each module first the P-unit is mapped and then the NS-unit. This way, if a P-unit accepts the current pattern, the classification of all the units after the corresponding NS-unit(s) are ignored. As we can also see from the figure, the network is repeated as many times as possible in the PE array. This allows data parallelism, which allows the classification of several patterns at a time.

For example, let us assume that the current pattern belongs to class 1. The PI unit will accept this pattern, rendering the disregard of the classification of all the units after NS3 and higher. Then the classification of p 1 is observed. If the vote is reject, the classification of *NS* 1 is also disregarded and the classification of *NS* 2 is regarded as the only relevant classification. This could result in either class 4 or 5 (see Figure 6.8), which would be a misclassification. If however, p 1 accepts the pattern, *NS* 1 is the relevant unit and classifies the pattern as either class 1 which would be correct, or 7 (see

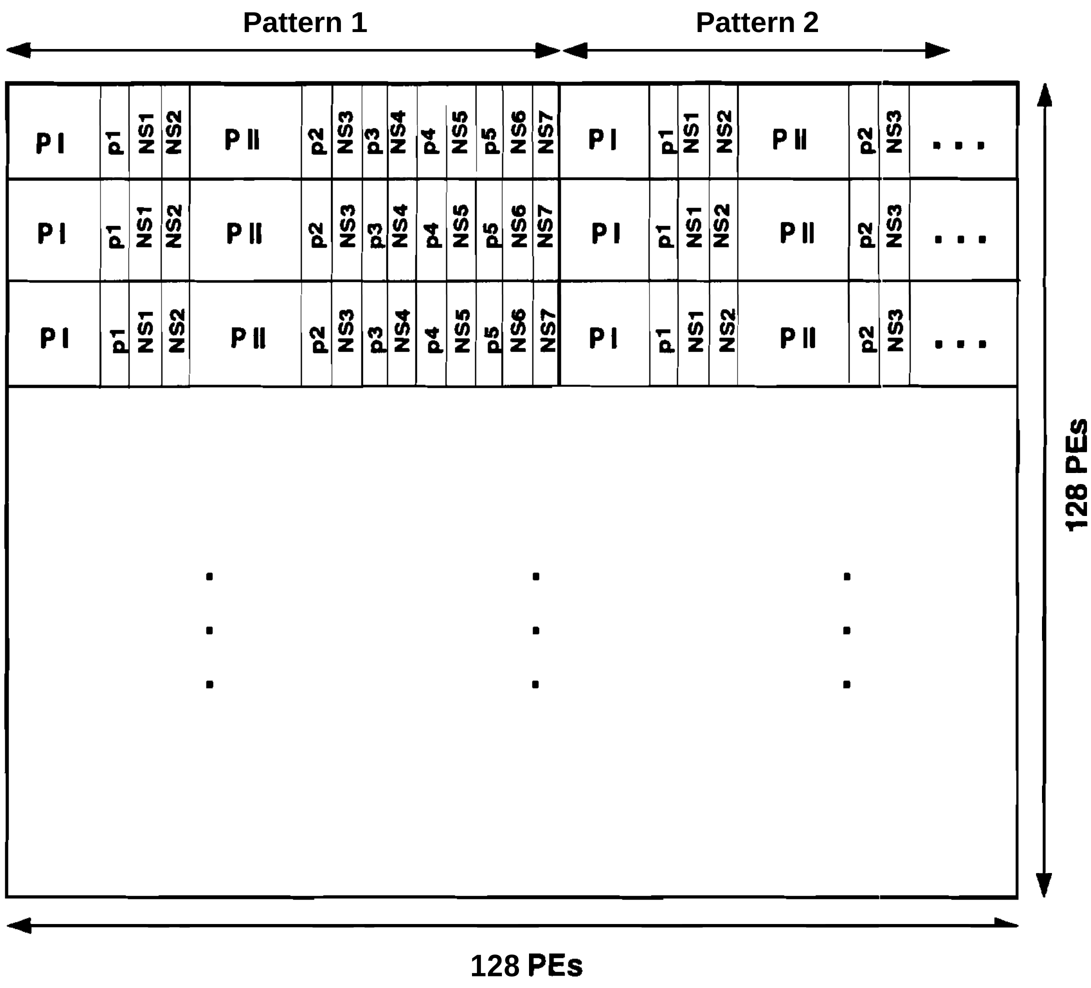


Figure 5.12 The Division of The PE Array for the Testing of the 10-Class Colorado Problem.

Figure 6.8) which would be incorrect.

CHAPTER 6

EXPERIMENTS AND RESULTS

The experimental results of two classification problems are discussed in this chapter. The first one is the speech synthesis problem and the second application is the 10-class Colorado data set.

The networks used are SIMD-PPSHNN1 and SIMD-PPSHNN2. The results of these networks are then compared to the results of the PSCNN and the backpropagation networks.

The backpropagation network used as a comparison was a two stage feed-forward fully-connected network. Various sizes of hidden layers were used to achieve the best performance. In all backpropagation network, the step size was kept at 0.7. In all SIMD-PPSHNN networks the step size was 0.01, and in PSCNN networks the step size was 0.05.

6.1 The Speech Synthesis Problem

There are two sets of data patterns for this application. One for training with 2319 patterns and another one for testing with 543 patterns. The characters "o", "u", "p", and "z" were intentionally under-represented in training. The FLAP class was the most represented class in the training set.

6.1.1 Backpropagation Results

As mentioned before, the backpropagation networks were all two-stage networks. The size of the hidden layer was varied to achieve optimum classification accuracy. The hidden layers tried had 20, 30, 40, and 50 hidden neurons. Figure 6.1 shows the performance tables of these networks. The figure shows the best performance of the 20 hidden neuron network, which was after 50 sweeps. The 30 hidden neuron network had its peak performance at 320 sweeps. The 40 hidden neuron network had its best performance after 300 sweeps. Finally, the network with 50 hidden neurons reaches its best performance at 700 sweeps. We can also see from the graph that the network with 40 hidden neurons performs the best ($\frac{389}{543} = 71.64\%$) among the 4 networks. Any increase or decrease in the number of hidden neurons from 40 hidden neurons reduced the accuracy of the network.

6.1.2 PSCNN Results

Figure 6.2 shows the best results of four PSCNN networks. All four models were trained with 200 sweeps of the training set. The first network has only one module and its best performance is 60.59%. The second model has 3 modules and its best accuracy is at 72.74%. The third network has 5 modules and its classification accuracy is 74.77%. The last network and the best performing network has 9 modules and pel-forms at 75.14%. Any increase in the number of modules from here on reduced classification accuracy. Also the accuracy of the networks started to decrease after 200 sweeps.

6.1.3 SIMD-PPSHNN 1 Results

Two modules were created for this problem. Figure 6.3 shows the results of the two module PPSHNN. The first module required a P-unit. It was trained to reject /b/, /v/, /f/, /s/, /z/, /e/, /o/, /u/, and /i/ and to accept the rest. Figure 6.3 (a) shows the results of the P-unit. It performed at 92.82% accuracy. This submodule had most problems with /p/. This P-unit was trained to accept data belonging to this class, but it only accepted 23 of the 43 patterns belonging to this class and rejected the other 20. Among the rejected classes, the P-unit had the lowest accuracy with /e/. It was trained to reject all the /e/ patterns. It rejected 12 of 15 patterns and accepted 3 of them.

Figure 6.3 (b) shows the results of the performance of the NS-unit of module 1. The results shown in this figure do not include the rejected data by the P-unit. We see that module one correctly classified 84.3% of the data accepted by the P-unit, incorrectly

20 hidden neurons
after 50 sweeps

	correct classifications	incorrect classifications
|t|	32	24
|d|	21	11
|p|	0	43
|b|	5	0
|v|	5	0
|f|	4	0
|s|	4	0
|z|	1	17
flap	39	41
|a|	96	0
|e|	15	0
|o|	0	26
|u|	0	22
|i|	22	0
space	115	0
total	359=66.11%	184=33.89%

(a)

30 hidden neurons
after 320 sweeps

	correct classifications	incorrect classifications
|t|	29	27
|d|	31	1
|p|	0	43
|b|	5	0
|v|	5	0
|f|	4	0
|s|	4	0
|z|	0	18
flap	39	41
|a|	96	0
|e|	15	0
|o|	0	26
|u|	0	22
|i|	22	0
space	115	0
total	365=67.22%	178=32.78%

(b)

40 hidden neurons
after 300 sweeps

	correct classifications	incorrect classifications
|t|	39	17
Hl	32	0
|p|	0	43
|b|	5	0
|v|	5	0
|f|	4	0
|s|	4	0
|z|	0	18
flap	52	28
|a|	96	0
|e|	15	0
|o|	0	26
|u|	0	22
|i|	22	0
space	115	0
total	389=71.64%	154=28.36%

(c)

50 hidden neurons
after 700 sweeps

	correct classifications	incorrect classifications
	[illegible]	[illegible]
|p|	1	42
|b|	5	0
|v|	5	0
|f|	4	0
|s|	4	0
|z|	0	18
flap	39	41
|a|	96	0
|e|	15	0
|o|	0	26
|u|	0	22
|i|	22	0
space	115	0
total	375=69.06%	168=30.94%

(d)

Figure **6.1** Results of BP for Speech Synthesis.

classified **4.3%** of them, and rejected **11.39%** of them.

Module two did not build a P-unit. Figure 6.3.(c) shows the results of the second

1-module PSCNN
after 200 sweeps

	correct classifications	incorrect classifications
ltl	28	24
ldl	16	16
lpl	0	43
lbl	5	0
lvl	5	0
lfl	4	0
lsl	4	0
lzl	0	18
flap	24	56
lal	95	1
lel	15	0
lol	0	26
lul	0	22
lil	18	4
space	115	0
total	329=60.59%	214=39.41%

(a)

3-module PSCNN
after 200 sweeps

	correct classifications	incorrect classifications
ltl	40	16
ldl	32	0
lpl	2	41
lbl	5	0
lvl	5	0
lfl	4	0
lsl	4	0
lzl	0	18
flap	54	26
lal	96	0
lel	15	0
lol	0	26
lul	1	21
lil	22	0
space	115	0
total	395=72.74%	148=27.26%

(b)

5-module PSCNN
after 200 sweeps

	correct classifications	incorrect classifications
ltl	42	14
ldl	32	0
lpl	0	43
lbl	5	0
lvl	5	0
lfl	4	0
lsl	4	0
lzl	3	15
flap	61	19
lal	96	0
lel	15	0
lol	1	25
lul	1	21
lil	22	0
space	115	0
total	406=74.77%	137=25.23%

(c)

9-module PSCNN
after 200 modules

	correct classifications	incorrect classifications
ltl	42	14
ldl	32	0
lpl	1	42
lbl	5	0
lvl	5	0
lfl	4	0
lsl	4	0
lzl	2	16
flap	63	17
lal	96	0
lel	15	0
lol	1	25
lul	1	21
lil	22	0
space	115	0
total	408=75.14%	135=24.86%

(d)

Figure 6.2 Results of PSCNN for Speech Synthesis.

module's NS-unit. It correctly classified 47.87% of the patterns passed to this module. 36.7% of patterns passed to this module were misclassified, and 10.64% of them were rejected.

Pre-Rejector of first module after 1200 sweeps

	correct lassifications	incorrect classifications
|t|	51	1
|d|	32	0
[illegible]	23	20
|b|	-	-
|v|	5	0
|f|	4	0
|s|	3	1
|z|	17	1
flap	72	8
|a|	96	0
|e|	12 .	3
|o|	25	1
|u|	22	0
|i|	22	0
space	115	0
total	505= 92.82%	39= 7.1%

(a)

module I performance

	correct classifications	incorrect classifications	rejected
|t|	39	2	10
|d|	32	0	0
|p|	5	1	17
|b|	0	0	0
|v|	0	0	0
|f|	0	0	0
|s|	0	1	0
|z|	0	1	0
flap	48	10	14
|a|	93	1	2
|e|	1	1	1
|o|	0	0	1
|u|	0	0	0
|i|	0	0	0
space	115	0	0
total	333=84.3%	17= 4.3%	45=11.39%

(b)

module 2 performance

	correct classifications	incorrect classifications	rejected
|t|	7	3	1
|d|	0	0	0
|p|	6	22	9
|b|	5	0	0
|v|	4	1	0
|f|	4	0	0
|s|	3	0	0
|z|	4	10	3
flap	15	7	0
|a|	1	1	0
|e|	12	0	1
|o|	4	16	5
|u|	3	18	1
|i|	22	0	0
space	0	0	0
total	90= 47.87%	69= 36.70%	20=10.64%

(c)

PPSHNN overall performance

	correct classifications	incorrect classifications	rejected
|t|	46	5	1
|d|	32	0	0
|p|	11	23	9
|b|	5	0	0
|v|	4	1	0
|f|	4	0	0
|s|	3	1	0
|z|	4	11	3
flap	63	17	0
|a|	94	2	0
|e|	13	1	1
|o|	4	16	5
|u|	3	18	1
total	423=77.9%	95= 17.5%	20=3.68%

(d)

Figure 6.3 Results of PPSHNN1 for Speech Synthesis.

The overall performance of the two-module PPSHNN1 is shown in Figure 6.3 (d). The best classification accuracy was 77.9%. As we can see, it outperformed the backpropagation and the PSCNN networks not only in overall classification accuracy but also in classification of patterns belonging to under-represented classes such as /p/ and /z/. Also, it is worth mentioning that 3.68% of the data was still rejected after two modules. A third module could increase the accuracy by a slight margin.

6.2 The 10-class Remote Sensing Problem

This data set contains a set of 1188 vectors for training and a set of 831 vectors for testing. The breakdown among the classes is shown in Figure 2.2. Each vector is of length seven and any component of the vector can have a value between 0 and 250. As seen in Figure 2.2, all 10 classes are present in both the training and the testing set.

6.2.1 Backpropagation Results

As for the speech synthesis problem, different size backpropagation networks (all with one hidden layer) were tried. Figure 6.4 shows the results of the three best performing network. Figure 6.4.(a) shows the best result among all backpropagation networks with 55.72% accuracy. This network has 100 hidden neurons. In figures (b) and (c) the results of two other networks are shown with 110 and 90 hidden neurons respectively.

6.2.2 PSCNN Results

Figure 6.5 shows the results of two PSCNN networks, one with 9 and one with 7 modules. The results are slightly better than the backpropagation networks, but still quite poor in the under-represented classes. Best performance was achieved with the 9 module network at about 56.68%.

Sample runs with the same data set were also done by other independent researchers [2]. In none of the cases was correct classification percentage above 60%. It is also important to mention here that none of the networks learned any of the classes 2, 3, 8, 9, and 10.

6.2.3 SIMD-PPSHNN1 Results

The P-unit used for this experiment is shown in Figure 3.10 and its performance statistics is shown in Figure 6.6.(a). The performance of the NS-unit of module one is shown in Figure 6.6.(b). Similar to the speech case, the results shown in the figure do not include the rejected data by the P-unit. The performance of the NS-unit of module 2 is shown in Figure 6.6.(c) and the overall performance of the network is shown in Figure 6.6.(d).

The P-unit was trained to reject classes 2, 3, 4, 8, 9, and 10 and to accept the remaining classes. Its performance was about 95.5%. Overall, the PPSHNN1 performed better than the other networks on the under-represented classes.

The result shown in Figure 6.6 are for the 100 hidden neuron network as the N-unit of the first module. Other hidden layer sizes were tested, but the best results were revealed

100 hidden neurons
after 500 sweeps

	correct classifications	incorrect classifications
class 1	190	5
class 2	0	24
class 3	0	42
class 4	29	36
class 5	96	43
class 6	82	106
class 7	65	5
class 8	0	44
class 9	0	25
class 10	1	38
total	463=55.72%	368=44.28%

(a)

110 hidden neurons
after 1000 sweeps

	correct classifications	incorrect classifications
class 1	189	6
class 2	1	23
class 3	0	42
class 4	30	35
class 5	95	44
class6	[illegible]	113
class 7	66	4
class 8	0	44
class 9	0	25
class 10	0	38
total	456=54.87%	374=45.01%

(b)

90 hidden neurons
after 700 sweeps

	correct classifications	incorrect classifications
class 1	171	24
class 2	0	24
class 3	0	42
class 4	30	35
class 5	96	43
class6	74	114
class 7	67	3
class 8	0	44
class 9	0	25
class 10	0	38
total	438=52.71%	392=47.17%

(c)

Figure 6.4 Results of BP for 10-Class Problem.

when we had 100 hidden neurons. Figure 6.7 shows the error curves of different SIMD-BP networks run for the N-unit. The smooth exponentially decaying error function is a

9 module PSCNN
after 200 sweeps

	correct classifications	incorrect classifications
class 1	192	3
class 2	0	24
class 3	0	42
class 4	29	36
class 5	96	43
class 6	86	100
class 7	65	5
class 8	0	44
class 9	0	25
class 10	1	38
total	471=56.68%	43.32%

(a)

7 module PSCNN
after 200 sweeps

	correct classifications	incorrect classifications
class 1	188	7
class 2	2	22
class 3	0	42
class 4	30	35
class 5	95	44
class 6	84	104
class 7	66	4
class 8	0	44
class 9	0	25
class 10	0	38
total	465=55.96%	366=44.04%

(b)

Figure 6.5 Results of PSCNN for 10-Class Problem.

characteristic of the exact algorithm. The error curves of the stochastic method are only piecewise smooth.

Pre-rejector of first module after 1500 sweeps

	correct classifications	incorrect classifications
class 1	195	0
class 2	21	3
class 3	37	[illegible]
class 4	64	1
class 7		- -
class 8	30	14
class 9	19	6
class 10	30	8
total	794=95.55%	37= 4.45%

(a)

module 1 performance

	correct classifications	reject	incorrect classifications
class 1	194	1	0
class 2	1	0	2
class 3	2	2	1
class 4	0	0	1
class 5	90	19	30
class 6	80	40	68
class 7	65	3	2
class 8	0	9	5
class 9	1	0	5
class 10	0	0	8
total	433=68.84%	74= 11.76%	122=19.40%

(b)

module 2 performance

	correct classifications	reject	incorrect classifications
class 1	0	0	1
class 2	15	2	4
class 3	21	1	17
class 4	43	5	16
class 5	3	5	11
class 6	8	10	22
class 7	0	0	3
class 8	5	12	22
class 9	2	8	9
class 10	3	15	12
total	100=35.97%	58= 20.86%	117=42.09%

(c)

PPSHNN overall performance

	correct classifications	reject	incorrect classifications
class 1	194	0	1
class 2	16	2	6
class 3	23	1	18
class 4	43	5	17
class 5	93	5	41
class 6	88	10	90
class 7	65	0	5
class 8	5	12	27
class 9	3	8	14
class 10	3	15	20
total	533=64.14%	58= 6.98%	239=28.76%

(d)

Figure 6.6 Results of SIMD-PPSHNN 1 for 10-Class Problem.

6.2.4 SIMD-PPSHNN2 Results

The performance of the SIMD-PPSHNN2 with PNS modules is shown in Table 6.1. The

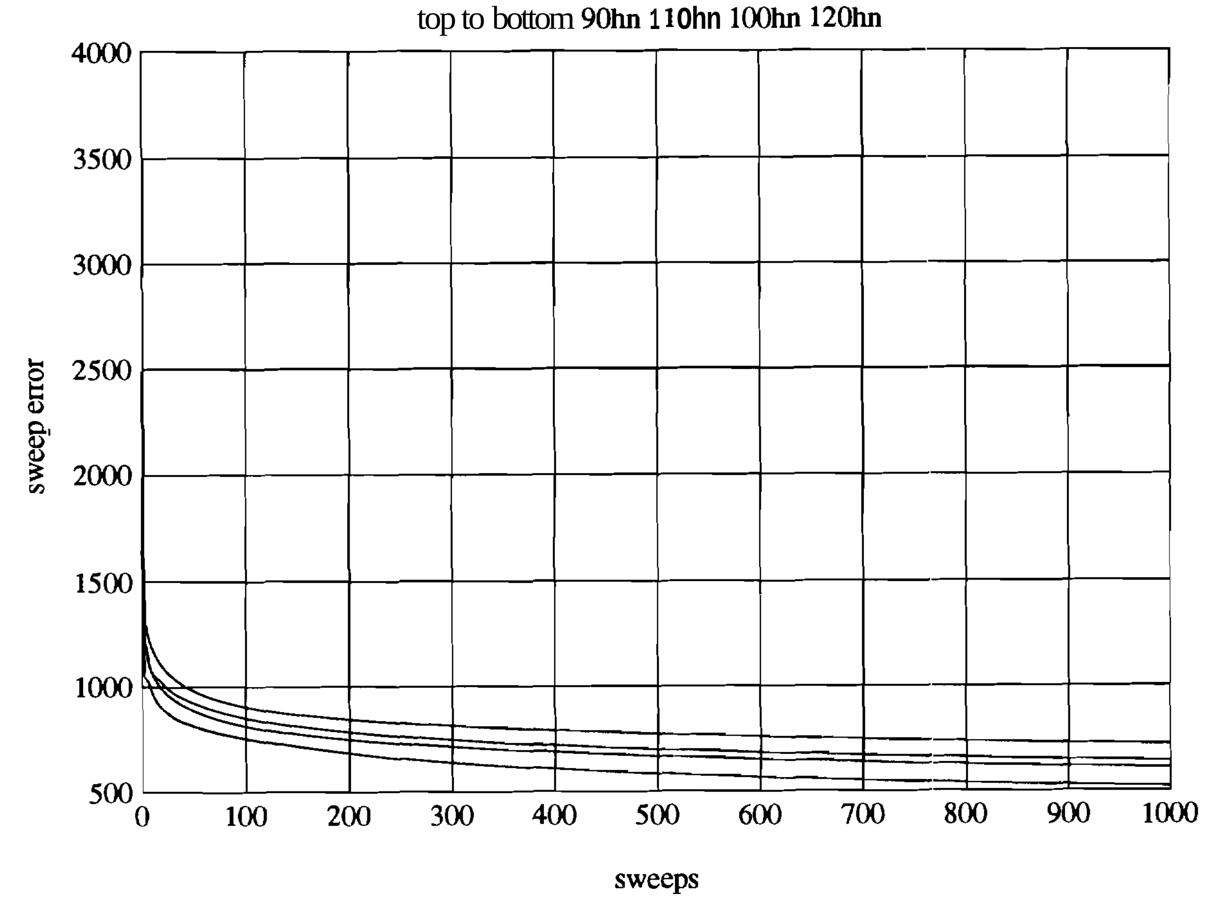


Figure 6.7. Error Curves of SIMD-BP.

correct classification performance was 73.16%. This performance improvement is mainly due to the separation of hard to learn classes (classes 2, 3, 8, 9, 10) from the rest of the classes in the first stage. This separation causes the simplification of the problem space and results in the improvement of the classification accuracy for both the "easy" as well as the "hard" to learn classes.

The P-unit of the first stage (Figure 6.8) allows classes 1, 4, 5, and 7 to be learned by the NS-unit of the first stage, separately from the other classes. These classes are relatively easy to learn, resulting in testing classification accuracy of 98.97%, 73.85%, 82.01%, and

Table **6.1 The Results of the SIMD-PPSHNN2 using PNS Modules for the 10-Class Colorado Problem.**

	correct	wrong	rejected	% correct	% wrong	% rejected
class 1	193	2	0	98.97	1.03	0
class 2	15	7 + 2	0	62.50	37.40	0
class 3	31	11	0	73.81	26.19	0
class 4	48	17	0	73.85	26.15	0
class 5	114	25	0	82.01	17.99	0
class 6	126	62	0	67.02	32.98	0
class 7	42	28	0	60.00	40.00	0
class 8	20	18 + 4	2	45.45	50.00	4.55
class 9	0	6	19	0	24.00	76.00
class 10	19	9 + 11	0	48.72	51.28	0
over all accuracy	608	202	21	73.16	24.31	2.53

60.00%, respectively.

By not including the other four classes with much larger training sample sets in the training set of the second stage, this stage can learn the remaining classes easier. The **NS-unit** of the second stage further breaks down the problem space into simpler polygons in terms of **PNS** modules. The testing performance of the second stage on classes **2, 3, 6, 8,** 9, and **10** are 62.5%, 73.81%, 67.02%, 45.45%, 0.00%, 48.72%, and 73.16%, which improves the overall performance of the network considerably.

Figure **6.8** shows the division of classes among the **PNS** modules of the network. The **P-**

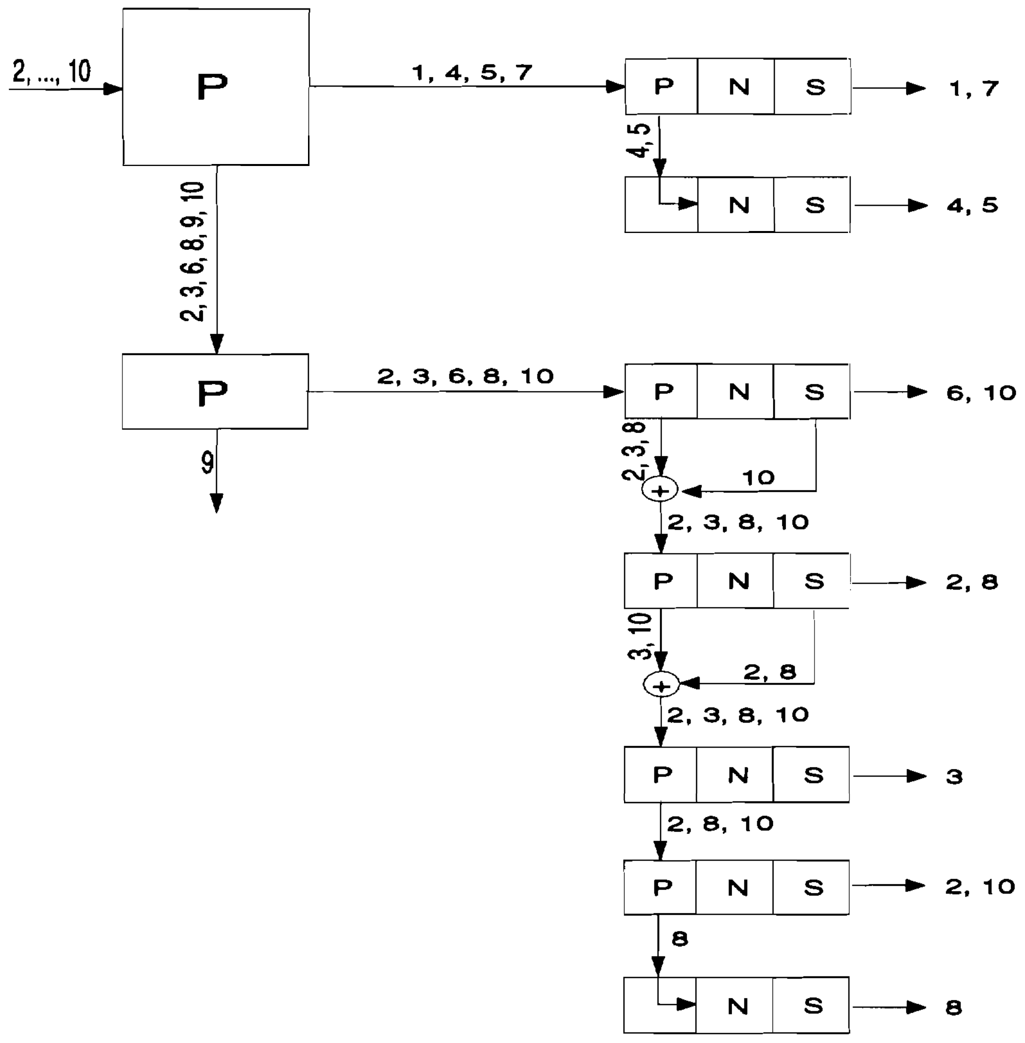


Figure 6.8. The Class Divisions Generated during Training of SIMD-PPSHNN2 for the 10-Class Colorado Problem.

unit of the first stage rejects classes 2, 3, 6, 8, 9, and 10, and accepts data belonging to classes 1, 4, 5, and 7. Data belonging to classes 1, 4, 5, and 7 are sent to the N-unit of the first stage for classification. There are two modules in this unit, one PNS module and one NS module. The P-unit of the PNS module rejects classes 4 and 5. The other two (classes 1 and 7) are sent to t he N-unit for classification. Hence, the NS module is responsible for the classification of classes 4 and 5, and with a correct classification performance of 73.81% and 82.01%, respectively, it was considered satisfactory and no P-unit was necessary.

In the second stage, the P-unit rejects data from class 9 and accepts the rest. Classes 2, 3, 6, 8, and 10 are sent to the NS-unit of this stage for classification. The NS-unit consists of four PNS modules and one NS module. The first PNS is responsible for classes 6 and 10. The P-unit of this module rejects classes 2, 3, and 8. The S-unit of the same module also rejects some data belonging to class 10 due to the uncertainty of classification. Therefore, the data set sent to the second module contains classes 2, 3, 8, and 10. The second PNS is responsible for classes 2 and 8, and rejects classes 3 and 10 using its P-unit. The S-unit of this module also rejects some data belonging to both classes 2 and 8, thus resulting in a data set for the third PNS which contains all four classes 2, 3, 8, and 10. The third PNS is only responsible for the class 3 and rejects the rest. Because, the N-unit of this PNS performed its task satisfactorily, its S-unit did not reject any patterns to the next PNS. Classes 2, 8, and 10 are sent to the fourth module which in turn is responsible for data belonging to classes 2 and 10, and rejects data belonging to class 8. The last PNS (NS module) classifies the remaining data to class 8 or rejects them.

Overall, both PPSHNN modules outperformed the backpropagation and PSCNN networks in all our experiments. Choosing PPSHNN2 with PNS module has the additional advantage that it is relatively inexpensive to run. This is due to its simple single stage units.

CHAPTER 7

FUTURE RESEARCH AND CONCLUSIONS

7.1 Future Research

Future research will involve further development of SIMD-PPSHNN in terms of accuracy, speed, and architecture. These studies should be carried out in relation to complex classification problems, pattern recognition and signal processing. The outline of the major issues of future research is as follows:

After the experiments with the SIMD-PPSHNN1 were completed, it was clear that most of the effort should be directed towards the automation of the process of finding the optimal network size for the N- and P-units. Up to that point, most of the training time was spent to find the optimal N- and P-unit size rather than training them. The result of this research was the PNS module which replaces the nonlinear boundaries introduced by the backpropagation networks with piecewise linear boundaries. At this point, a logical next step would be to experiment with other types of networks and learning algorithms, such as competitive learning.

2. A study should be done to see if there are situations in which certain networks with certain learning rules perform better than others. If so, the network should employ certain types of networks in certain types of classification problems. Hence, PPSHNN would become an assembly of different types of networks and learning algorithms organized into a hierarchy. In such a case, a unit must be added to each module to detect a known situation and thereby use the optimal type of network. This task could be performed by the pre-processor.

3. It can be shown [5] that the output of the delta rule network can be interpreted as the probability of a class given the input vector. Using this knowledge, one can design a neural network module to estimate the required probability density functions, hence replacing the Parzen density estimation by a neural network module. Future research should consider this topic and the accuracy of the neural network unit in comparison the Parzen estimator.

4. Another important issue is to design an effective pre-processor. 'This research will look into techniques introduced in information theory and error control coding to devise a pre-processor which transforms the problem space into yet another easier space for classification. Another option is an adaptive pre-processor. This pre-processor learns a nonlinear transformation and performs it on the incoming data. The nonlinear transformation itself is learned from the training data.

Future research could also involve replacing the hierarchical nature of the algorithm with a consensual nature similar to that of PSCNN. Thus, gaining more parallelism in training and taking more advantage of machines such as MasPar becomes possible. Some recent work has been done by Professor Hank Dietz and his students at Purdue University in using MasPar in an MIMD fashion. The consensual nature can go hand in hand nicely with the MP-1 running in a semi-MIMD fashion.

In such a case, one must develop a decision mechanism to choose between the votes of different modules. When the hierarchy is not present, more than one P-unit could accept the input pattern. A decision must be made as to which module's classification result should be accepted. A voting mechanism such as the one from PSCNN could also be used. Once the hierarchy of the PPSHNN algorithm has been eliminated, the biggest source of serialism in the algorithm will also have been eliminated, and hence all the modules can be trained at the same time and with the entire training set (assuming enough hardware resources). This would perhaps increase the classification accuracy as well.

6. Future work also could involve further developing the postrejector and its statistical analysis of the output of the N-unit.

7. As mentioned before, we are currently implementing the simplest possible cost criterion. Further research is required to find the optimal cost criterion for

estimation of the reject boundaries. One suggestion is that it might be possible to learn the cost values during training. The effect of various cost criterion in classification accuracy can be studied.

8. In Chapter 3, we talked about the rejection boundaries z^k_{0r}, z^k_{01}, and z^k_{r1} and the order they held in our experiments, namely

$$0 < z^k_{0r} < z^k_{01} < z^k_{r1} < 1.$$

It is proposed that neurons whose outputs carry little information do not follow the above order. Future research is aimed at finding topologies in which there is a pattern for such behavior. If so, the knowledge gained can be used in designing a more efficient algorithm which can be used to detect the unneeded neurons early in training and to eliminate them.

7.2 Conclusions

In this thesis, a new neural network architecture called the Parallel Probabilistic Self-organizing Hierarchical Neural Network (PPSHNN) was introduced. The PPSHNN is a cornbination of statistical analysis techniques and adaptive neural networks. This cornbination is shaped into a new architecture which is designed to divide the problem space into subregions and make classification easier in these subregions. This division of space, performed by the P-unit, is completely data (application) dependent and is not a pre-set procedure.

The PPSHNN addresses problems that rise in complex classification applications such as under- or unproportionally represented classes in the training set. It also addresses the training time issues and is to a high degree parallelizable. Training times of over 3000 times shorter than serial backpropagation implemented on Sun 3/60 have been achieved by implementation on MasPar MP-1 with 16K PEs.

The experiments performed in comparison to a standard backpropagation network and the PSCNN indicate superior accuracy and speed. Further detailed study, analysis, and development of the PPSHNN is necessary to understand its potential in many classification applications.

The variation of the PPSHNN module called the PNS module offers several advantages. The PNS module is relatively inexpensive and at the same time accurate in classification. Because the architecture is fractal in nature and all the modules are simple and similar in architecture, the building of networks which use this module is inexpensive and strait-forward. It divides the problem space using simple linear boundaries and therefore, it. self-organization to adapt to the problem space is easier to understand.

Implementing neural network algorithms in massively parallel machines is very promising in reducing the training time from hours to minutes. This kind of speed-up is impossible to achieve even with a fast neural network algorithm implemented on the fastest serial machine.

The backpropagation algorithm can offer architectural parallelism as well as data pal-allelism if implemented in the way it was discussed in this thesis. While architectural parallelism is limited by the size of the largest layer of the network, the data parallelism is only limited by the number of PEs available and the number of training patterns, which is often far more than the number of neurons in a layer.

Massively parallel implementations of neural networks allow larger problems to be investigated in a short amount of time. Since the properties of neural networks often arise due to the collective behavior of the neurons, such implementations also have the potential of helping in the understanding of artificial and biological mechanisms of intelligence.

REFERENCES

[1] D.E. Rumelhart, J.L. McClelland, ***Parallel Distributed Processing,*** The MIT press, Cambridge Massachusetts, 1986.

[2] H. Valafar, O.K. Ersoy, "Parallel Self-Organizing, Consensual Neural Networks"; Report No. TR-EE 90-56, School of Electrical Engineering, Purdue University, October 1990.

[3] J.A., Benediktsson, P.H. Swain and O.K. Ersoy, "Neural Network Approaches versus Statistical Methods in Classification of Multisource Remote Sensing Data," ***IEEE Int. Geoscience and Remote Sensing Symposium,*** Vancouver, Canada, July 1989, and submitted to ***IEEE Tran. Geoscience and Remote Sensing.***

[4] D.G. Luenberger, ***Linear and Nonlinear Programming,*** Second Edition, Addison-Wesley Publishing Company, Massachusatts, 1984.

[5] O.K.Ersoy, D. Hong, "Parallel, Self-Organizing, Hierarchical Neural Networks"; ***IEEE Tran. on Neural Network,*** Vol. 1, No. 2, pp. 167-178, June 1990.

[6] O.K.Ersoy, D. Hong, "Parallel, Self-Organizing, Hierarchical Neural Networks II"; *IEEE* Tran. on Industrial Electronics, Special Issue on Neural Networks, April 1993.

[7] O.K.Ersoy, S-W. Deng, "Parallel, Self-Organizing, Hierarchical Neural Networks with Continuous Inputs and Outputs"; Proc. Hawaii Int. *Conf.* System Sciences, HICCS-24, pp.486-492, Kauai, January, 1991, and to appear in IEEE Tran. Neural Networks.

[8] O.K.Ersoy, S-W. Deng, "Parallel, Self-Organizing, Hierarchical Neural Networks with Forward-Backward Training"; Circuits Systems Signal Processing, Vol. 12, No. 2, pp. 223-246, 1993.

[9] Richard O. Duda, Peter E. Hart, Pattern classification and *scene* analysis, Wiley-interscience publication.

[10] K. Fukunaga, Introduction to Statistical Pattern Recognition, Academic Press, New York, 1972.

[11] K. Hwang, F. Briggs, Computer Architecture and Parallel Processing, McGraw-Hill Computer Science Series, New York, 1984.

[12] G. S. Almasi, A. Gottlieb, Highly Parallel Computer, The Benjamin/Cummings Publishing Company, Inc. 1990.

[13] MasPar MP-1 Reference Manuals, MasPar Computer Corporation, Sunnyvale, CA

[14] Kenneth E. Batcher, "Design of a Massively Parallel Processor", *ZEEE* Transaction on Computers, Vol. C-29, pp. 836-840, Sept. 1980.

[15] Peter Christy, "Software To Support Massively Parallel Computing on the MasPar MP-1"; Proceedings of the *ZEEE* Compcon Spring 1990, Feb. 1990.

[16] D.F. Specht, "Probabilistic Neural Networks and the Polynomial adaline as Complementary Techniques for Classification"; *ZEEE* Trans. Neural Networks, Volume 1, Number 1, pp. 111-121, March 1990.

[17] T.J. Sejnowski, "NETtalk: A Parallel Network that learns to read Aloud," Report No. TR-JHU/EECS-86/01, Electrical Engineering and Computer Science, Johns Hopkins University, 1986.

[18] O.K. Ersoy, D. Hong, "A Hierarchical Neural Network involving Nonlinear Spectral Processing", Proceedings of *ICNN* 89, Washington, D.C., June 1989.

[19] P.J.B. Hancock, "Data representation in neural nets: an empirical study", Proceedings of the 1988 Connectionist Models summer School, Morgan Kaufmann Publishers Inc., pp. 11-20, 1988.

[20] J.C. Pemberton, J.J. Vidal, "When is the Generalized Delta Rule a Learning Rule? A Physical Analogy", Proceedings of ICNN 88, San Diego, Cal., pp. 309-315, June 1988.

[21] M. Takeda, J. W. Goodman, "neural networks for computation: number representations and programming complexity", Applied Optics, Vol.. 25, No. 18, pp. 3033-3046, Sep. 1986.

[22] H.L. Van Trees, detection, Estimation, and Modulation Theory, Part I, John W'iley & sons, Inc. 1986.

[23] P.J. Werbos, "Backpropagation: past and future", *Proceedings* of ICNN 88, San Diego, Cal., pp.343-353, June 1988.

[24] B. Widrow, "Adaptive sampled-data systems-A statistical theory of adaptation," in 1959 WESCON Conv. Rec., pt. 4, pp. 74-85, 1959.

[25] R.G. Gallager, Information Theory and Reliable *Communication*, John Wiley & suns, Inc. 1968.

[26] R.E. Blahut, Principles and Practice of Information *Theory*, Addison-Wesley publishing company, 1987.

[27] N. Abrarnson, Information Theory and coding, McGraw-Hill Book Company, Inc. 1963.

[28] R. Ash, Information Theory, John Wiley & Sons, Inc. 1965.

[29] D. A. Pomerleau, G. L. Gusciora, D. S. Touretzky, and H. T. Kung, " Neural Network Simulation at Warp Speed: How We Got 17 Million Connections per Second", *Proc.* IEEE International Con. on Neural Networks, Voll. **II**, San Diago, CA, pp. 143-150, Jul. 1988.

[30] S. Borkar et al., "iWarp: An Integrated Solution to High Speed Parallel Computing", Proc. Supercomputing *'88,* IEEE Computer Society, Orlando, FL, pp. 330-339, 1988.

[31] J. R. Millan, P. Bofill, "Learning by Back-propagation: A Systolic Algorithm and its Transputer Implementation", Neural Networks 3, pp. 119-137, Jul. 1989.