View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Purdue E-Pubs

Purdue University
Purdue e-Pubs

ECE Technical Reports | Electrical and Computer Engineering

6-1-1993

PNS MODULES FOR THE SYNTHESIS OF PARALLEL SELF-ORGANIZING HIERARCHICAL NEURAL NETWORKS

Faramarz Valafar
Purdue University School of Electrical Engineering

Okan K. Ersoy
Purdue University School of Electrical Engineering

Follow this and additional works at: http://docs.lib.purdue.edu/ecetr

Valafar, Faramarz and Ersoy, Okan K., "PNS MODULES FOR THE SYNTHESIS OF PARALLEL SELF-ORGANIZING HIERARCHICAL NEURAL NETWORKS" (1993). *ECE Technical Reports.* Paper 231.
http://docs.lib.purdue.edu/ecetr/231

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

PNS Modules for the Synthesis of Parallel Self-Organizing Hierarchical Neural Networks

Faramarz Valafar
Okan K. Ersoy

TR-EE 93-22
June 1993

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

PNS MODULES FOR THE SYNTHESIS OF PARALLEL SELF-ORGANIZING HIERARCHICAL NEURAL NETWORKS

Faramarz Valafar, Okan K. Ersoy

Purdue University
School of Electrical Engineering
W. Lafayette, IN 47907

ABSTRACT

The PNS module is discussed as the building block for the synthesis of parallel, self-organizing, hierarchical, neural networks (PSHNN). The PNS consists of a prerejector (P-unit), a neural network (N-unit) and a statistical analysis unit (S-unit). The last two units together are also referred to as the NS unit. The P- and NS-units are fractile in nature, meaning that each such unit may itself consist of a number of parallel PNS modules. Through a mechanism of statistical acceptance or rejection of input vectors for classification, the sample space is divided into a number of subspaces. The input vectors belonging to each subspace are classified by a dedicated set of PNS modules. This strategy results in considerably higher accuracy of classification and better generalization as compared to previous neural network models. If the delta rule network is used to generate the N-unit, each subspace approximates a linearly separable space. In this sense, the total system becomes similar to a piecewise linear model.

1. INTRODUCTION

Parallel, self-organizing, hierarchical neural networks (PSHNN's) with quantized outputs were introduced in [1] and [2]. PSHNN's with continuous outputs were discussed in [3] and [4]. The PSHNN involves a self-organizing number of stages, similar to a multilayer network. Each stage can be a particular neural network, to be referred to as the stage neural network (SNN). Unlike a multilayer network, each SNN is essentially independent of the other SNN's in the sense that each SNN does not receive its input directly from the previous SNN. At the output of each SNN, there is an error detection scheme. If an input vector is rejected, it goes through a nonlinear transformation before being fed to the next SNN.

The motivation for this architecture evolves from the consideration that most errors occur due to input signals to be classified which are linearly nonseparable or which are close to boundaries between classes. At the output of each stage, such signals are detected by a scheme and rejected. Then the rejected signals are passed through a nonlinear transformation so that they are convened into other vectors which are more easily classified by the succeeding stage.

Learning with the PSHNN is similar to learning with a multilayer network, except that error detection is carried out at the output of each SNN, and the procedure is stopped without further propagation into the succeeding SNN's if no errors are detected. Testing (recall) with the PSHNN can be done in parallel with all the SNN's simultaneously rather than each SNN waiting for data from the previous SNN.

In this paper, we propose and discuss the PNS module as the building block for the synthesis of PSHNN's. The PNS consists of a prerejector (P-unit), a neural network (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. In the PSHNN networks discussed in this paper, the P-units have replaced the input nonlinearities even though the input nonlinearities can still be used.

The concept of the PNS module has evolved as a result of analyzing the major reasons for errors in classification problems. The major reasons for errors can be considered to be the following:

1. Patterns which are very close to the class boundaries are usually difficult to differentiate.
2. The classification problem may be extremely nonlinear.
3. A particular class may be undersampled such that the number of training samples for that class are too few, as compared to the other classes. Figure 1 a) visualizes such a scenario with Class 1 as compared to Class 2.

4. A particular class may be geometrically small in a certain region of the sample space such that the number of samples belonging to that class from that region is too few. This is visualized in Figure 1 b).

The PSHNN consisting of a number of self-organizing PNS modules is synthesized to minimize misclassification errors due to the reasons outlined above.

The paper consists of five sections. Section 2 describes the creation of the PNS modules. to generate the PSHNN. Section 3 describes the statistical technique to generate the S-unit. Section 4 discusses the comparative results of simulations. Conclusions are given in Section 5.

2. THE CREATION OF THE PNS MODULES

The block diagram for a PNS module is shown in Figure 2. The N-unit can be any type of neural network, but is chosen as a delta rule network with output nonlinearity [5] in this paper.

The procedure for the creation of the PNS modules is shown in the flow charts of Figures 3 and 4. 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. After the N-unit converges, the S-unit is created. The S-unit is a parallel statistical classifier which performs bit-level three-class Bayesian analysis on the output bits of the N-unit. It is discussed in detail in Section 3. One result of this analysis is the generation of the probabilities P_1^k, $k=1,2,\ldots M$, M being the number of classes. P_1^k signifies the probability of detecting an input pattern belonging to class k correctly. If this probability is equal to or smaller than a small threshold 6, the input vectors belonging to that class are rejected before they are input to the N-unit. In other words, if $P_1^k \leq 6$, the corresponding class is either geometrically small or undersarnpled, or has highly nonlinear boundaries such that the present network can not learn it.

The rejection of such classes before they are fed to the N-unit is achieved by the creation of the P-unit. It 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 patterns belonging to the classes which are easier to classify.

If a P-unit is created, the N-unit is retrained with the remaining classes accepted by the P-unit. Afterwards, the process discussed above is repeated. The S-unit is also regenerated. It may again reject some classes. Then, another P-unit is created to reject these classes. This results in a recursive procedure.

If there *are* no more classes rejected by the S-unit, a PNS module is generated . 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. A particular example is shown in Figure 5, which shows the PNS modules generated for the 10-class Colorado problem discussed in detail in Section 4. 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.

In addition to deciding which classes should be rejected, the S-unit also generates certain other thresholds for the acceptance or the rejection of an input pattern, as discussed in Section 3. 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, in the Colorado problem discussed in Section 4, two stages were required, as seen in Figure 5.

The recursive nature of the algorithm becomes evident when a P-unit and 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 to generate the particular training data for its learning, as shown in Figure 4.

Figure 4 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 3. To do so, the program gathers all the rejected data from the first stage. If there are no more rejected data or 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. P- and the NS units may themselves create PNS modules. If the delta rule network is used to generate the N-units, the net result will be 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. This topic is further discussed in Section 4.

3. THE STATISTICAL TECHNIQUE FOR THE CREATION OF THE S-UNIT

The S-unit is schematically shown in Figure 6. It consists of bit classifiers for every output bit of the N-unit. These classifiers are three class Bayesian classifiers which classify the output bit into one, zero, or reject classes. In addition, the S-unit generates P_1^k's among other a priori probabilities, which are used to determine whether to create a P-unit, and if so, which classes are to be rejected by the P-unit.

In the strict rejection scheme, an input vector is rejected if any one of the output bits that is generated by the N-unit is rejected by the S-unit. This strategy could be relaxed by introducing a Vector Rejector (VR) network after the S-unit. The VR network would be trained to reject or to accept an input vector based on the certainty of the classification of all the bits. In the experiments reported in Section 4, we used the strict method of bitwise rejection without the VR network.

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

The statistical analysis technique for the creation of the S-unit is described below:

Bitwise rejection: Bitwise rejection is performed by the bitwise classifiers. Each bitwise classifier is a three class Maximum A Posteriori (MAP) Detector [6].

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

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

Figure 7 visualizes an example of the types of input vectors which results in bit rejection. The patterns near the class boundaries cause bit rejection and are rejected by the S-unit.

The following notation will be used:

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

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

$p_i^k = p^k(H_i)$ = a priori probability for bit k that hypothesis H_i is true.

$P^k(H_i|z)$ = probability of hypothesis H_i being true for bit k, given the output value z.

The a posteriori probability, $P^k(H_i \mid z)$, can be computed from $f^k(z \mid H_i)$ using Bayes rule:

$$P^k(H_i \mid z) = \frac{f^k(z \mid H_i)\ p^k(H_i)}{f^k(z)} \tag{1}$$

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

$$R^k(H_i \mid z) = \sum_j C_{ij}^k\ P^k(H_i \mid z) \qquad i,\ j\ \varepsilon\ \{\ 0,\ 1,\ r\ \}. \tag{2}$$

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

$$R^k(H_0 \mid z) = C_{00}^k P^k(H_0 \mid z) + C_{01}^k P^k(H_1 \mid z) + C_{0r}^k P^k(H_r \mid z). \tag{3}$$

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

$$R^k(H_1 \mid z) = C_{10}^k P^k(H_0 \mid z) + C_{11}^k P^k(H_1 \mid z) + C_{1r}^k P^k(H_r \mid z), \tag{4}$$

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

$$R^k(H_r \mid z) = C_{r0}^k P^k(H_0 \mid z) + C_{r1}^k P^k(H_1 \mid z) + C_{rr}^k P^k(H_r \mid z) \tag{5}$$

An expected loss is called a risk, and $R^k(H_i \mid z)$ is known as the conditional risk. For a particular output z, we can minimize the expected loss by selecting the hypothesis that minimizes the conditional risk. This can be shown as follows:

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 $R^k(H_i \mid z)$ is the conditional risk associated with choosing H_i, and since the decision rule specifies the hypothesis chosen, the overall risk is given by

$$R = \int R\ (\zeta^k(z) \mid z)\ f^k(z)\ dz \tag{6}$$

where dz is signifies a d-space volume element, and the integral extends over the entire feature space. Clearly, if $\zeta^k(z)$ is chosen so that $R\ (\zeta^k(z) \mid z)$ 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, we compute the conditional risk

$$R^k(H_i \mid z) = \sum_j C_{ij}^k \; P^k(H_i \mid z) \qquad i, j \in \{ 0, 1, r \} \tag{7}$$

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

Thus, for the output value z at every bit k, we define the following decision rule which has minimum risk:

$$\zeta^k(z) = \begin{cases} \text{if } R^k(H_0 \mid z) < R^k(H_1 \mid z) \;\&\; R^k(H_0 \mid z) < R^k(H_r \mid z) & \text{choose } H_0 \\ \text{if } R^k(H_1 \mid z) < R^k(H_0 \mid z) \;\&\; R^k(H_1 \mid z) < R^k(H_r \mid z) & \textit{choose } H_1 \\ \text{otherwise} & \text{choose } H_r \end{cases} \tag{8}$$

This decision rule indicates that there are three tests to be performed as follows:

TEST 1:

The first test is between H_0 and H_1:

$$R^k(H_0 \mid z) \underset{H_0}{\overset{H_1}{\gtrless}} R^k(H_1 \mid z) \tag{9}$$

Using (3) and (4) and letting $C_{00}^k = C_{11}^k = C_{rr}^k = 0$, which is the common assumption in most classification problems, we get

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

$$C_{01}^k P^k(H_1 \mid z) - C_{10}^k P^k(H_0 \mid z) \underset{H_0}{\overset{H_1}{\gtrless}} (C_{1r}^k - C_{0r}^k) \; P^k(H_r \mid z). \tag{11}$$

Assuming $f^k(z) \neq 0$, we can multiply both sides by $f_z^k(z)$ and get

$$C_{01}^k P^k(H_1 \mid z) f^k(z) - C_{10}^k P^k(H_0 \mid z) f^k(z) \underset{H_0}{\overset{H_1}{\gtrless}} (C_{1r}^k - C_{0r}^k) P^k(H_r \mid z) f^k(z). \tag{12}$$

Using Bayes rule (1) and assuming $P_i^k \neq 0$, Eq. (12) becomes

$$C_{01}^k f^k(z \mid H_1) P^k(H_1) - C_{10}^k f^k(z \mid H_0) P^k(H_0) \underset{H_0}{\overset{H_1}{\gtrless}} (C_{1r}^k - C_{0r}^k) f^k(z \mid H_1) P^k(H_r) \quad (13).$$

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

$$\frac{f^k(z \mid H_1)}{f^k(z \mid H_0)} \underset{H_0}{\overset{H_1}{\gtrless}} \frac{C_{10}\ P^k(H_0)}{C_{01}\ P^k(H_1)}, \tag{14}$$

$$p_1^k\ f^k(z \mid H_1) \underset{H_0}{\overset{H_1}{\gtrless}} p_0^k\ f^k(z \mid H_0). \tag{15}$$

TEST 2:

The second test is between H_0 and H_r:

$$R^k(H_0 \mid z) \underset{H_0}{\overset{H_r}{\gtrless}} R^k(H_r \mid z) \tag{16}$$

Using (3) and (5) and letting $C_{00} = C_{rr} = 0$, we get

$$C_{01}^k P^k(H_1 \mid z) + C_{0r}^k P^k(H_r \mid z) \underset{H_0}{\overset{H_r}{\gtrless}} C_{r0}^k P^k(H_0 \mid z) + C_{r1}^k P^k(H_1 \mid z). \tag{17}$$

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

$$C_{01}^k P^k(H_1 \mid z) f^k(z) + C_{0r}^k P^k(H_r \mid z) f^k(z) \underset{H_0}{\overset{H_r}{\gtrless}} C_{r0}^k P^k(H_0 \mid z) f^k(z) + C_{r1}^k P^k(H_1 \mid z) f^k(z) \quad (18),$$

$$C_{0r}^k f^k(z \mid H_r) P^k(H_r) - C_{r0}^k f^k(z \mid H_0) P^k(H_0) \underset{H_0}{\overset{H_r}{\gtrless}} (C_{r1}^k - C_{01}^k) f^k(z \mid H_1) P^k(H_1) \quad (19),$$

$$C_{0r}^k f^k(z \mid H_r) p_r^k - C_{r0}^k f^k(z \mid H_0) p_0^k \underset{H_0}{\overset{H_r}{\gtrless}} (C_{r1}^k - C_{01}^k) f^k(z \mid H_1) p_1^k \quad (20)$$

TEST 3

The third test is between H_r and H_1:

$$R^k(H_1 \mid z) \underset{H_1}{\overset{H_r}{\gtrless}} R^k(H_r \mid z) \quad (21)$$

With the same assumptions as in the previous two tests and the same operations, test 3 results in

$$C_{1r}^k f^k(z \mid H_r) p_r^k - C_{r1}^k f^k(z \mid H_1) P_1^k \underset{H_1}{\overset{H_r}{\gtrless}} (C_{r0}^k - C_{10}^k) f^k(z \mid H_0) P_0^k \quad (22)$$

For simplicity, let us define the following three functions:

$$\Gamma_1^k(z) = p_1^k \; f^k(z \mid H_1) - p_0^k \; f^k(z \mid H_0) \quad (23)$$

$$\Gamma_2^k(z) = C_{0r}^k f^k(z \mid H_r) p_r^k - C_{r0}^k f^k(z \mid H_0) p_0^k + (C_{01}^k - C_{r1}^k) f^k(z \mid H_1) p_1^k \quad (24)$$

$$\Gamma_3^k(z) = C_{1r}^k f^k(z \mid H_r) p_r^k - C_{r1}^k f^k(z \mid H_1) P_1^k + (C_{10}^k - C_{r0}^k) f^k(z \mid H_0) P_0^k \quad (25)$$

Then, the decision rule (8) becomes

$$\zeta^k(z) = \begin{cases} \text{if } \Gamma_1^k(z) \ \& \ \Gamma_2^k(z) < 0 & \textit{choose } H_0 \\ \text{if } \Gamma_1^k(z) > 0 \ \& \ \Gamma_3^k(z) < 0 & \textit{choose } H_1 \\ \text{otherwise} & \text{choose } H_r \end{cases} \tag{26}$$

The decision rule (26) corresponds to determining decision thresholds as follows:

For test *1*, set $\Gamma_1^k(z) = 0$, and use (23) to find

$$z_{01}^k = \Gamma_1^{k^{-1}}(0). \tag{27}$$

Then, the interval $\iota = \left[0, 1\right[$ is divided into two subintervals, $\iota_1^{k0} = \left[0, z_{01}^k\right]$ in which H_0 is true, and $\iota_1^{k1} = \left[z_{01}^k, 1\right]$ in which H_1 is true.

In the same manner, we compute z_{0r}^k and z_{r1}^k, from test 2 and 3, using (24) and (25), respectively. 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 2 intervals by each test (ie. $\Gamma_1^k(z)$, $\Gamma_2^k(z)$, and $\Gamma_3^k(z)$ have only one root). Figure 8 shows a typical outcome of the three tests in which

$$0 < z_{0r}^k < z_{01}^k < z_{r1}^k < 1 \tag{28}$$

It is also possible that the order in (28) does not hold.

The decision strategy governed by (26) 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 8, if the output value of bit k falls in the interval $\left[0, z_{0r}^k\right]$, that bit is classified as zero. If the output value falls in $\left[z_{r1}^k, 1\right]$, then it is classified as one, and finally, if it is in $\left[z_{0r}^k, z_{r1}^k\right]$, that bit is rejected.

In order to evaluate Γ_0^k, Γ_1^k and Γ_r^k, we need to compute the conditional probability density functions $f^k(z \mid H_i)$ as well as all the a priori probabilities P_i^k, required in (23), (24), and (25).

Estimation of the Conditional Density Functions ($f^k(z \mid H_i)$): There are two general approaches to density estimation, parametric and *nonparametric*. If we can assume a density function which can be characterized by a set of parameters, we can design a classifier which uses estimates of these parameters to estimate the probability density function. Unfortunately, it is often difficult to assume a parametric form for the density function.

There are two popular **nonparametric** estimation techniques: the Parzen ***density*** estimation and the k-nearest neighbor ***density*** estimation **(kNN)** [7]. They are fundamentally 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 decided to use the Parzen approach since instead of a uniform kernel, a Gaussian dismbution function can be used, which gives smoother estimation. The Parzen approach uses the following equation to estimate the density function:

$$f(z) = \frac{1}{n}\sum_{i=1}^{n} \kappa(z - z_i). \tag{29}$$

Where $\kappa(z - z_i)$ is the kernel with the mean of z_i, z_i's are the data samples, and n is the total number of samples. The following normal kernel function was used in our experiments:

$$\kappa(z) = \frac{1}{\sqrt{2\pi}\,\sigma_z} e^{-\frac{z^2}{2\sigma_i^2}} \tag{30}$$

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

Consider the training set $\Omega = \{X_1, X_2, \ldots, X_N\}$ 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 bitk: $\Omega_0^k = \{X_1, X_2, \ldots, X_{M_0}\}$ 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 c 0.5): $\Omega_{00}^k = \{X \mid z_x^k < 0.5\} = \{X_1, X_2, \ldots, X_{r_0}\}$ 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: $\Xi_{00}^k = \{z \mid X_z \varepsilon\, \Omega_{00}^k\} = \{z_1, z_2, , \ldots, z_{r_0}\}$

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

$$\kappa(z - z_i) = \frac{\alpha_i}{\sqrt{2\pi}\,\sigma_i} e^{-\frac{(z-z_i)^2}{2\sigma_i^2}} \tag{31}$$

where α_i is a normalization constant given by

$$\alpha_i = \frac{\sqrt{2\pi}\,\sigma_i}{\int_0^1 e^{-\frac{(z-z_i)^2}{2\sigma_i^2}}\, dz} \tag{32}$$

The constant α_i compensates for the fact that z can only be between 0 and 1.

5. Use *(29)* to estimate $\hat{f^k}(z \mid H_0)$:

$$\hat{f^k}(z \mid H_0) = \frac{1}{r_0} \sum_{i=1}^{r_0} \kappa(z - z_i) = \frac{1}{r_0} \sum_{i=1}^{r_0} \frac{\alpha_i}{\sqrt{2\pi}\,\sigma_i} e^{-\frac{(z-z_i)^2}{2\sigma_i^2}} \tag{33}$$

The above procedure is the same for $\hat{f^k}(z \mid H_1)$ and $\hat{f^k}(z \mid H_r)$ except for steps 1 and 2. For $f^k(z \mid H_1)$, steps *1* and *2* are as follows:

1. Find the set Ω_0^k of data samples in Ω which have a desired output value of 1 for bit k $\Omega_0^k = \{X_1, X_2, \ldots, X_{M_1}\}$ 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 ($z^k > 0.5$):
$$\Omega_{11}^k = \{X \mid z_x^k > 0.5\} = \{X_1, X_2, \ldots, X_{r_1}\}$$

For $\hat{f^k}(z \mid H,)$, 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:

$$\Omega_{rr}^k = \{X \mid (X \varepsilon \Omega_0^k \;\&\; z_x^k > 0.5) \; or \; (X \varepsilon \Omega_1^k \;\&\; z_x^k < 0.5)\} = \{X_1, X_2, \ldots, X_{r_r}\} \tag{34}$$

r_0, r_1, r_r satisfy

$$r_0 + r_1 + r_r = N \tag{35}$$

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:

$$\hat{P}_0^k = \frac{r_0^k}{N} \quad \hat{P}_1^k = \frac{r_1^k}{N} \quad \hat{P}_r^k = \frac{r_r^k}{N} \tag{36}$$

Cost of error (C_{ij}^k)**:** Though it is possible to have different cost criteria for different bits, we decided to have one criterion for all bits. Then, C_{ij}^k, simplifies to C_{ij}. In addition, the following logical assumptions were used:

1. $C_{ii} = 0$ (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 as 0 or 1, are the same).
3. $C_{0r} = C_{1r}$ (The costs of choosing H_0 or H_1 when H_r should have been chosen are equal).
4. $C_{01} = C_{10}$ (The cost of choosing H_0 when H_1 was true and the cost of choosing H_1 when H_0 was true are equal).
5. $C_{r0} = C_{r1} < C_{0r} = C_{1r}$ (The consequence of classifying H_0 or H_1 as H_r is less severe than classifying H_r as H_0 or H_1).
6. $C_{01} = C_{10}$ *w* $C_{0r} = C_{1r} > C_{r0} = C_{r1}$ (The consequence of classifying H_0 as H_1 or reverse is much higher than any other error).

Section 4 describes some of the experimental values which were selected for these ***error penalties.***

Using the above a posteriori and a priori estimates in (23), (24), and (25), we can estimate $\Gamma_1^k(z)$, $\Gamma_2^k(z)$, and $\Gamma_3^k(z)$. Using these estimates in (26), we can decide on one of the three hypotheses H_0, H_1, or H_r. For example, Figure 9 shows the boundaries created between the three hypotheses by the first NS-unit in the 10-class Colorado problem.

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 that sample and send it to the next stage or accept it and send it to the classifier for classification.

Remarks:

It can be shown [2] 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. COMPUTER SIMULATIONS

PSHNN networks generated with PNS modules were tested with the 10-class Colorado remote sensing data, and the results were compared to those obtained with other networks. The Colorado data set was described in [1]. It contains 1188 training patterns and 831 testing patterns. Each pattern is a vector of seven components and belongs to one of ten possible classes. Table 1 shows the number of training and testing patterns in each class.

The PNS modules of the designed PSHNN are shown in Figure 5, as discussed before. In constructing the network, the error cost values were experimentally chosen as $\{ C_{r0} = C_{r1} = 2 , C_{0r} = C_{1r} = 8 , C_{01} = C_{10} = 10 \}$ or $\{ C_{r0} = C_{r1} = 1 , C_{0r} = C_{1r} = 2 , C_{01} = C_{10} = 5 \}$. The results of the two cases were similar. We observed that the first criterion made the reject region to grow slightly, and the zero and the one regions to shrink slightly. The system seemed to be very robust and insensitive with respect to the numeric values of the error penalties as long as the following order was applied:

$$C_{01} = C_{10} > C_{0r} = C_{1r} > C_{r0} = C_{r1} \tag{37}$$

The classification performance of the new network was compared to backpropagation networks [5] and PSCNN networks[8]. The sample results of 3-layer backpropagation networks with 90, 100, 110 hidden units are shown in Table 2. The best performance was observed with the network with 100 hidden neurons at an accuracy of 55.72%. The networks were trained until no improvement in accuracy was observed with further training. This was achieved after the number of iterations indicated in Table 2.

The sample results of PSCNN networks with 7 and 9 modules (SNN's) are shown in Table 3. These two networks were also trained until no further improvement in accuracy was observed. This happened for both networks after 200 training sweeps.

Sample runs with the same data set were also done by other independent researchers [I.]. In no case, correct classification percentage was 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.

The performance of the new network with the PNS modules is shown in Table 4. The 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 improvement of the classification accuracy for both the "easy" and the "hard" to learn classes.

The P-unit of the first stage (Figure 5) 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 60.00%, respectively.

By not having the other four classes with much larger training sample sets, the second stage to learn the remaining classes. The NS-unit of the second stage further breaks down the problem space into simpler subspaces 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 10 shows the division of classes among the PNS modules of the network. The P-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 the 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 class 9 data 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 3^{rd} class and rejects the rest and since its N-unit performed its task satisfactorily, the 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 belonging to class 8.

As discussed in Section 2, 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 8, this can be shown as follows:

In the previous section, we showed how to compute two rejection boundaries for every bit. In Figure 8, these rejection boundaries are marked as z^k_{0r} and z^k_{r1}. Assuming the N-unit is a single stage delta rule network with sigrnoidal output nonlinearity, the output value of the k^{th} neuron is computed by

$$y^k = \frac{1}{1 + e^{-\sum_{i=0}^{n_i} x_i \omega_{ki}}} \tag{38}$$

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 (38), the equation describing the boundary imposed by the S-unit at bit k between the zero and the reject regions is

$$\frac{1}{1 + e^{-\sum_{i=0}^{n_i} x_i \omega_{ki}}} = z_{0r}^k. \tag{39}$$

The above equation can be written as:

$$\sum_{i=0}^{n_i} x_i \omega_{ki} = \ln \left[\frac{z_{0r}^k}{1 - z_{0r}^k} \right]. \tag{40}$$

This is a linear equation and describes a hyperplane in the n_i-dimensional space. The same argument of linearity can be used for the boundary between the reject and the one regions.

From the above discussion, the following important result follows: The network 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 this paper (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.

As another example of this process, Figure 11 shows the network created for the **XOR** problem and the boundaries generated. Figure 11 a) shows the network itself. It has two stages in which the second stage simply classifies everything passed on to it as class zero. In Figure 11 b) the boundaries which were created by the P-unit of the first stage are shown. Since this unit is built as a NS-unit, it created two boundaries between which a classification is rejected by its S-unit. If the pattern falls below these boundaries, the P-unit classifies it as reject and sends it to the next stage where it is classified as zero. If the pattern is above these boundaries, it is passed on to the NS-unit of the first stage which in turn creates its own set of boundaries. If the pattern falls below these boundaries, it is classified as one. If it falls between these boundaries, classification is rejected; and if it falls above the boundaries, it is classified as zero. This results in the division of the sample space as shown in Figure 11 b).

Notice that there are two types of rejected data by the P-unit of the first stage: data definitely rejected by the P-unit and data rejected by its S-unit. Data rejected by the P-unit are data which are sent to the next stage. Data rejected by the S-unit are data whose classifications are uncertain and it is not certain whether to send them to the next stage or to send them to the following N-unit of the same stage. Normally both types are sent to the next stage and that stage is trained to handle both types. In the **XOR** problem, however, the network did not encounter data of the latter type and since the first type should all be classified as zero, no PNS modules were built for the second stage.

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 in the **XOR** problem is shown in Figure 12. The only difference here is that, the input values are squared before inputting to the output neuron. The k^{th} output neuron has an output given by

$$y^k = \frac{1}{1 + e^{-\sum_{i=0}^{n_i} x_i^2 \omega_{ki}}} \tag{41}$$

The equation of the boundaries is given by

$$\sum_{i=0}^{n_i} x_i^2 \omega_{ki} = \ln\left[\frac{z_{0r}^k}{1 - z_{0r}^k}\right]. \tag{42}$$

This may result in a hyperbolic, or an elliptic boundary as shown in Figures 12 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 5 a).

We also have experimented with two stage backpropagation networks replacing the single stage networks. The resulting network created a piecewise nonlinear approximation of the true boundaries. The change in the model had very little effect in the overall accuracy 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 remaining relatively computationally inexpensive.

CONCLUSIONS

The PNS modules as basic building blocks for the synthesis of PSHNN's result in high classification accuracy, as compared to previous neural network models such as backpropagation and previous versions of PSHNN's

The effectiveness of the PNS module is due to the collaboration of the P- and S-units to reject input vectors as well as classes which are hard to classify, and allowing the N-units to achieve high performance of classification. The rejected vectors are handled by succeeding PNS modules.

The P- and NS-units are fractile in nature, meaning that each such unit may itself consist of a number of parallel PNS modules.

The S-unit has been designed as a statistical Bayesian analyzer. It can also be designed by using PNS modules to do same type of statistical analysis. Then, the total network consists of a number of parallel PNS modules. Initially, the network consists of a single N-unit. It grows to its final configuration by self-organization.

When each P- and N-unit is implemented by a delta rule network, the sample space is divided into regions which are linearly separable. This is similar to the approximation of a nonlinear system by a piecewise linear model.

It is clear that the basic strategy discussed in this paper can be further developed and modified for further improvement in performance.

REFERENCES

[1] 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.

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

[3] 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.

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

[5] D.E. Rumelhart, J.L. McClelland, Parallel Distributed Processing, The MIT press, Cambridge Massachusets, 1986.

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

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

[8] F. Valafar, O.K. Ersoy, "A Parallel Implementation of Backpropagation Neural Network on MasPar MP-1"; Technical Report, TR-EE-93-14, School of Electrical Engineering, Purdue University, W. Lafayette, IN 47907, March 1993, and submitted to *Int.* Journal of Parallel Programming.

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

[10] A. Papoulis, Probability, Random Variables, and Stochastic Processes, McGraw-

Hill Book Company, New York, 1965.

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

[12] H.L. Van Trees, Detection, Estimation, and Modulation Theory, Part *1*, John Wiley & sons, Inc. 1986.

[13] P.J. Werbos, "Backpropagation: Past and Future", Proceedings of ICNN 88, San Diego, Cal., pp.343-353, June 1988.

[14] R.G. Gallagher, Information Theory and Reliable Communication, John Wiley & suns, Inc., 1968.

[15] W.B., Jr., Davenport, W.L. Root, An Introduction to the Theory of Random Signals and Noise, McGraw-Hill Book Company, New York, 1958.

[16] L.J. Doob, Stochastic Processes, John Wiley and Sons, New York, 1953.

[17] W. Feller, An Introduction to Probability Theory and its Applications, John Wiley and Sons, New York, Vol. I, 1957, Vol. 11,1967.

[18] C.W. Helstrom, Statistical Theory of Signal Detection, 2^{nd} ed., Pergamon Press, New York, 1968.

Table 1. The Number of Training and Testing Patterns in the 10-Class Colorado Problem.

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

Table 2. The Results of BP Networks in the 10-Class Colorado Problem.

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
class 6	75	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
class 6	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)

Table 3 The Results of Two PSCNN Networks for the 10-Class Colorado Problem.

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)

Table 4 The Results of Two PSHNN 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

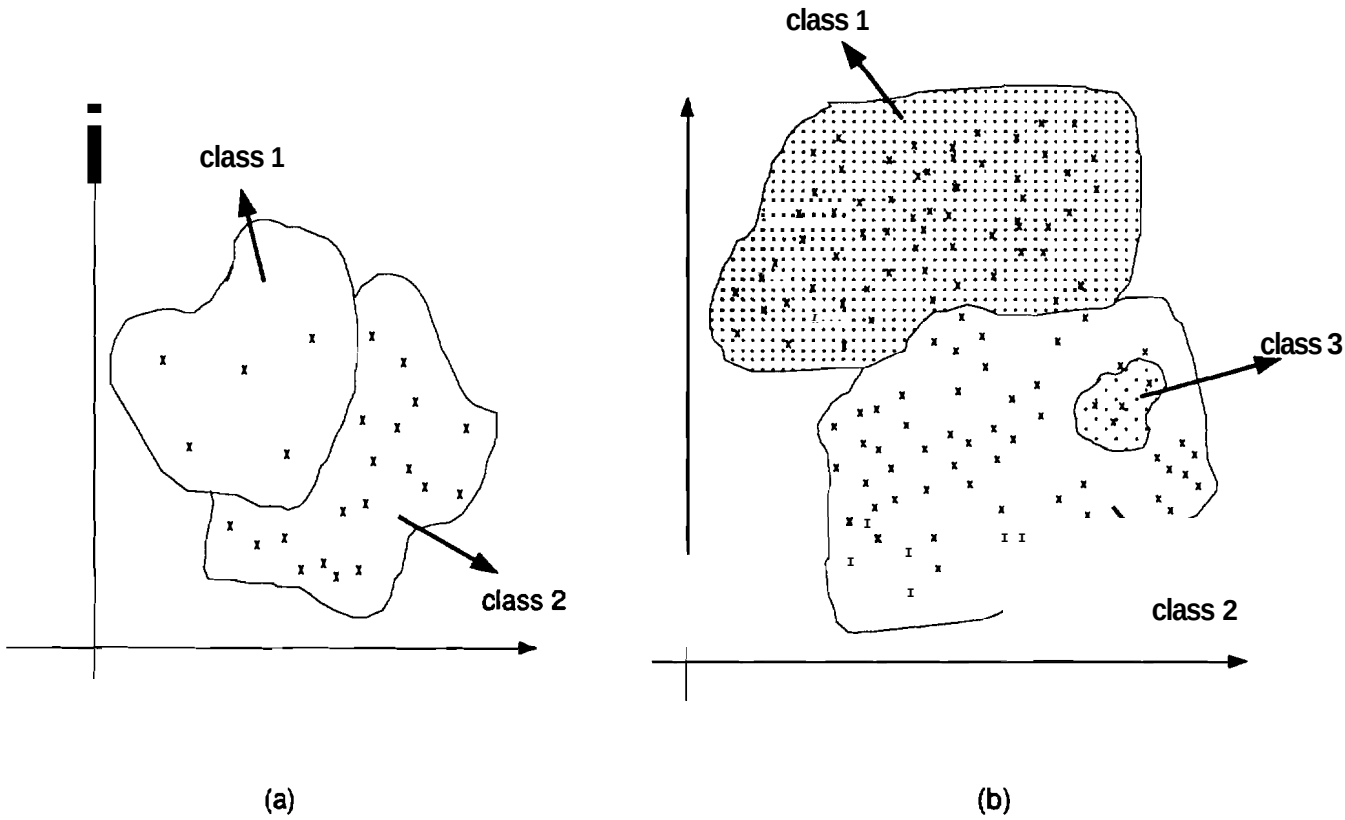


Figure 1. (a) An Example of an Undersampled Class (Class 1) (b) An Example of a Geometrically Small Class (Class 3).

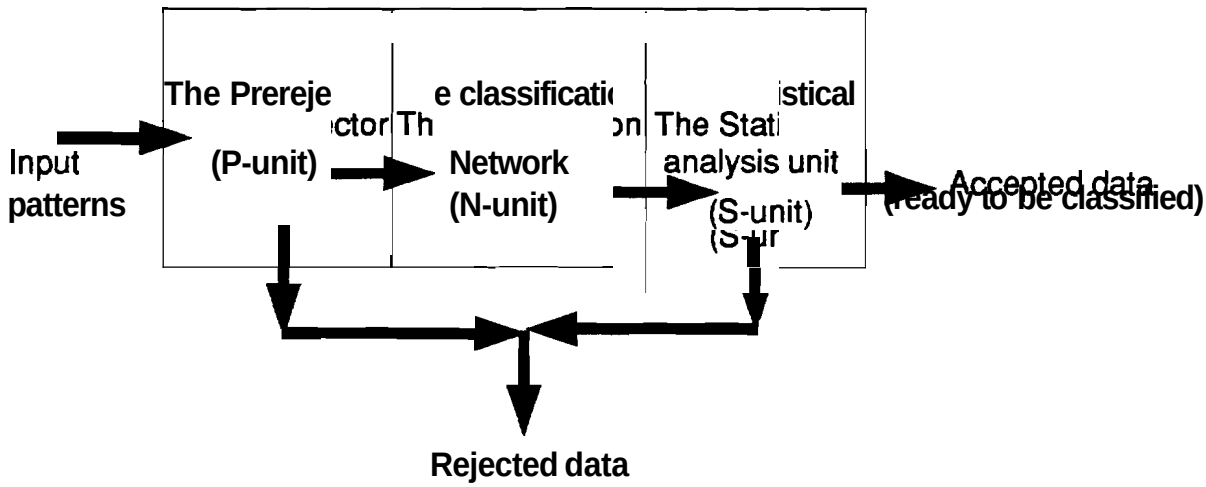


Figure 2. **The Block Diagram of a PNS Module.**

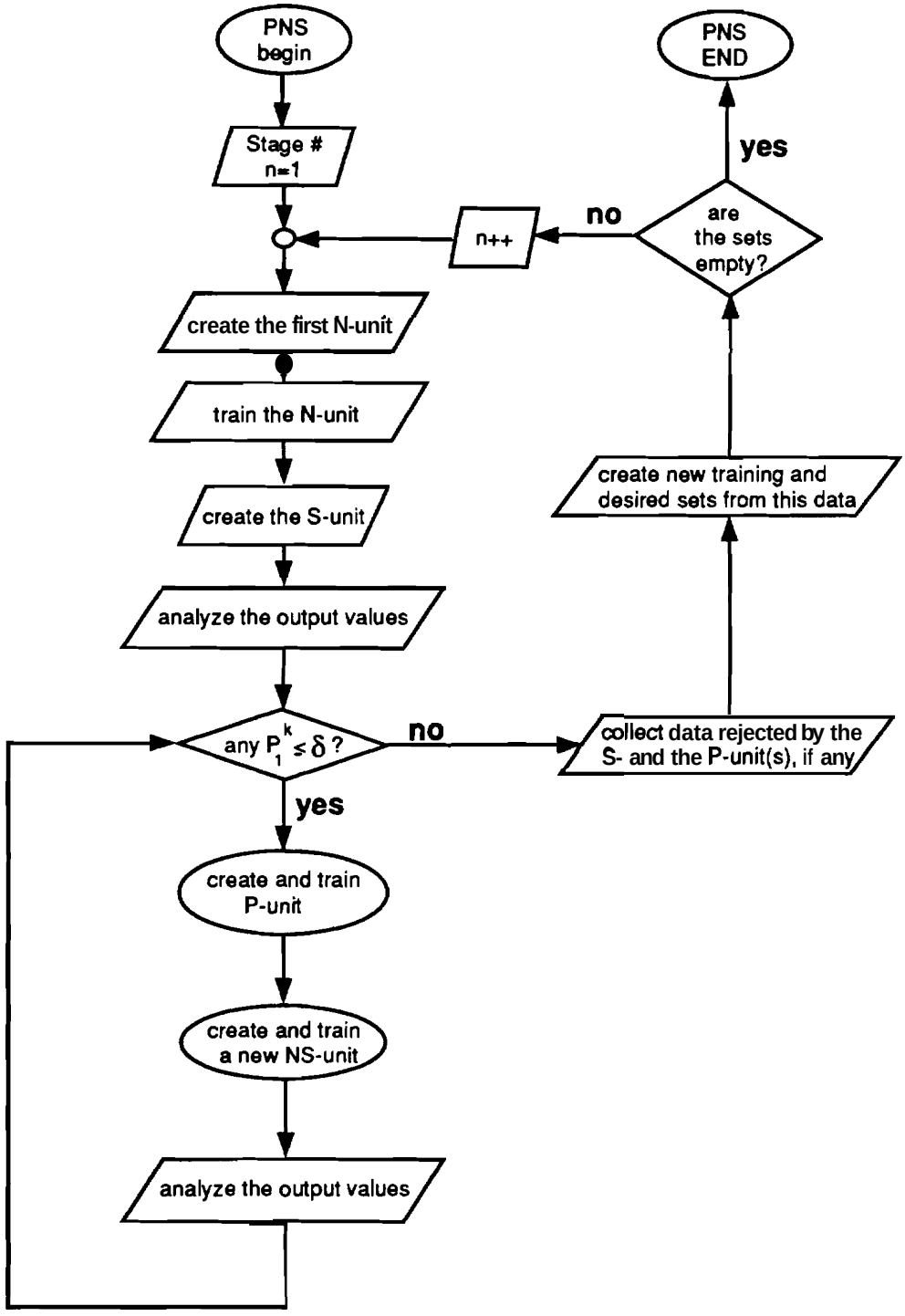


Figure.3. Flow Chart for Learning of a PNS Module.

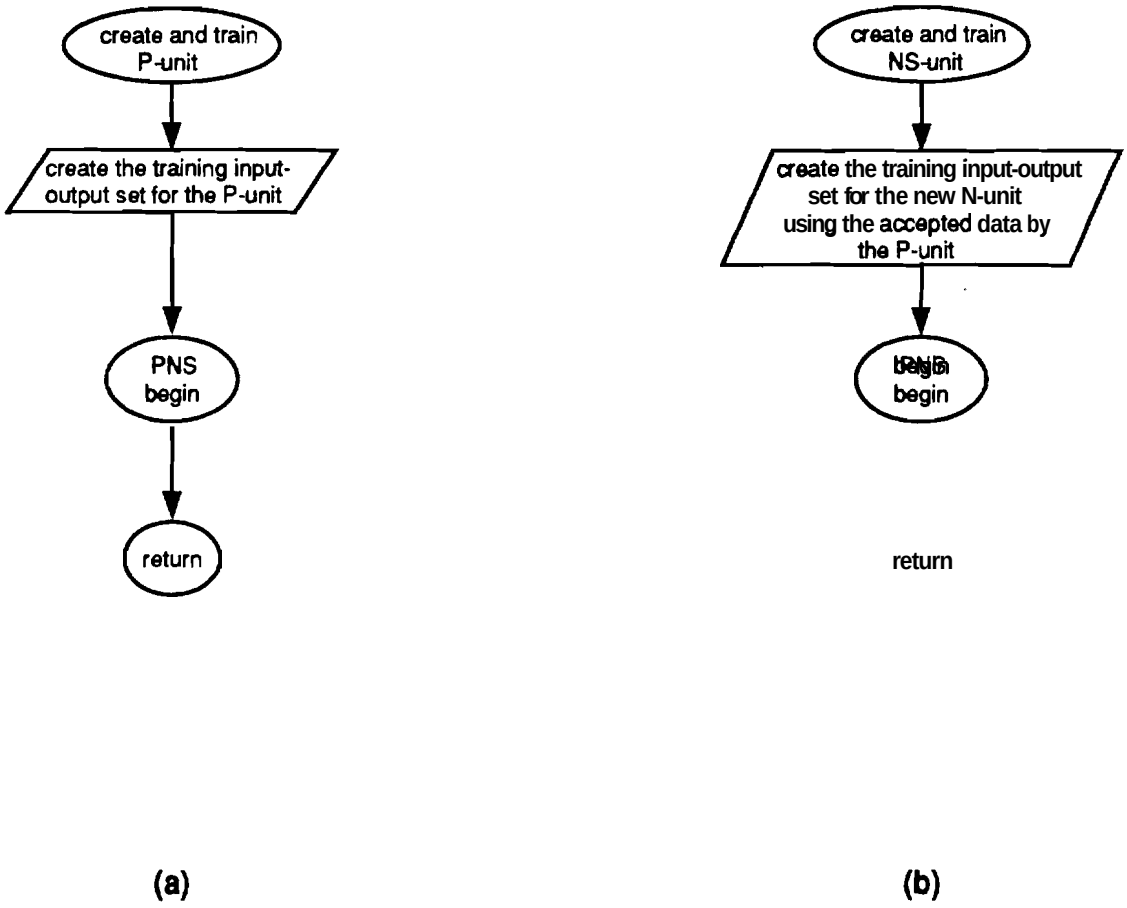


Figure 4. (a) The recursive procedure to create a N-unit.
(b) The recursive procedure to create a P-unit.

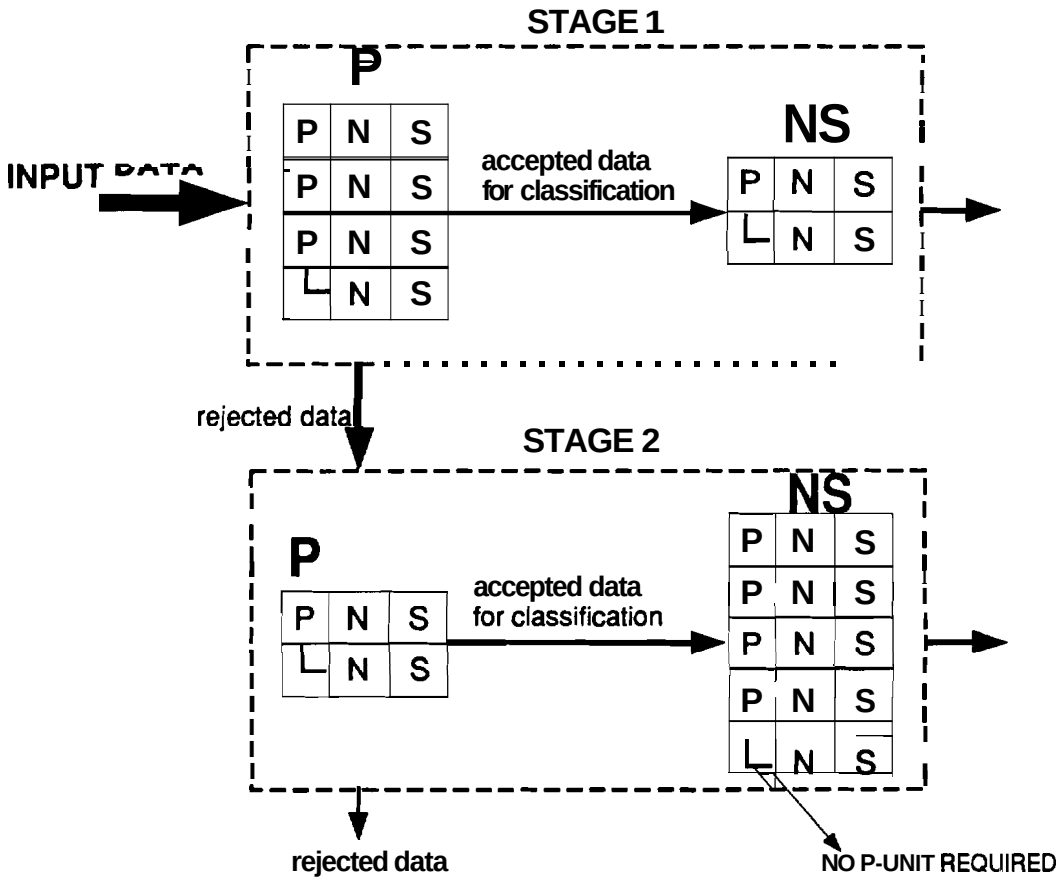


Figure 5. The PNS Modules in the PSHNN Designed for the 10-Class Colorado Problem.

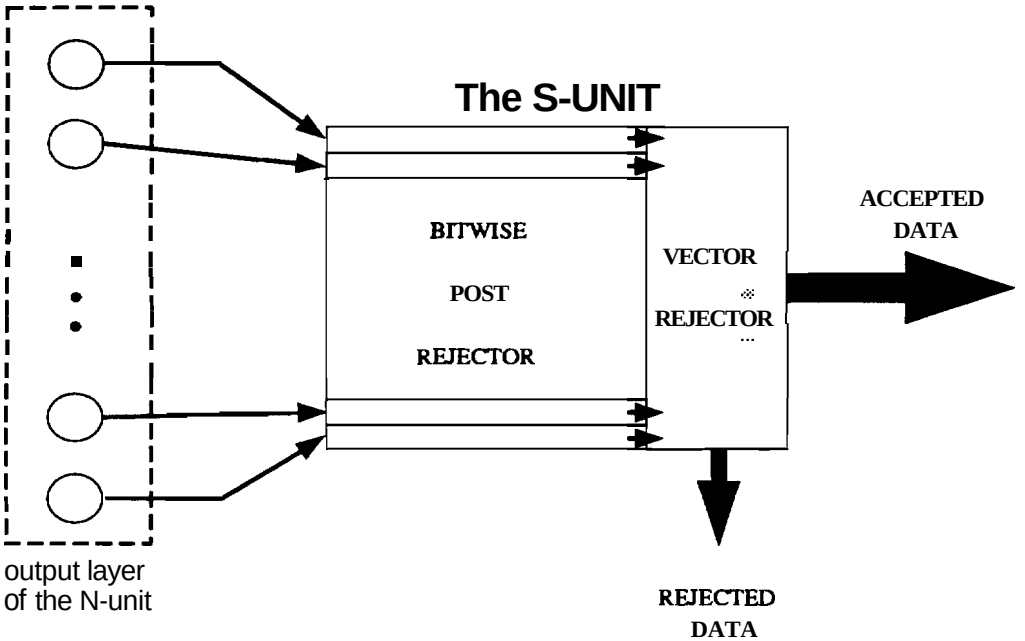


Figure 6. **The S-unit of a PNS Module.**

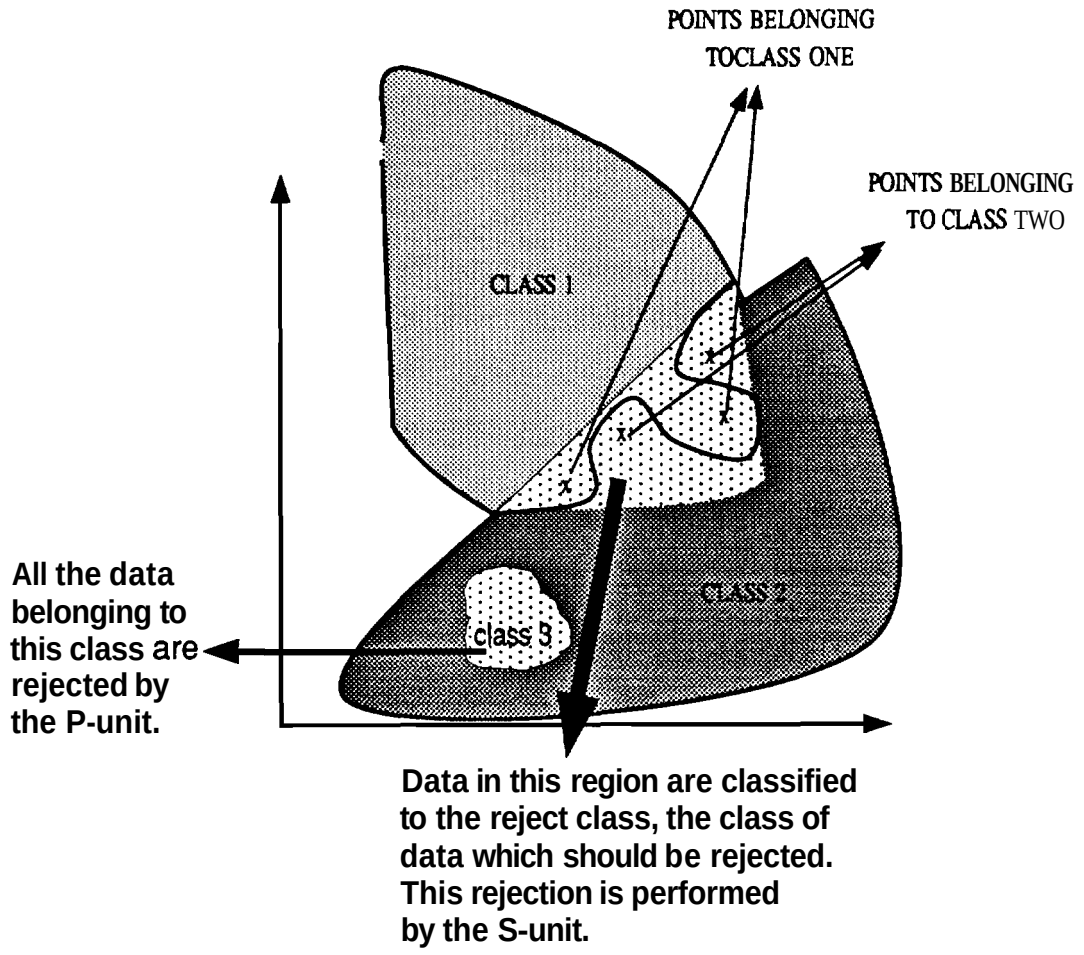


Figure 7. An Example of Data Types which The P- and S-units Reject.

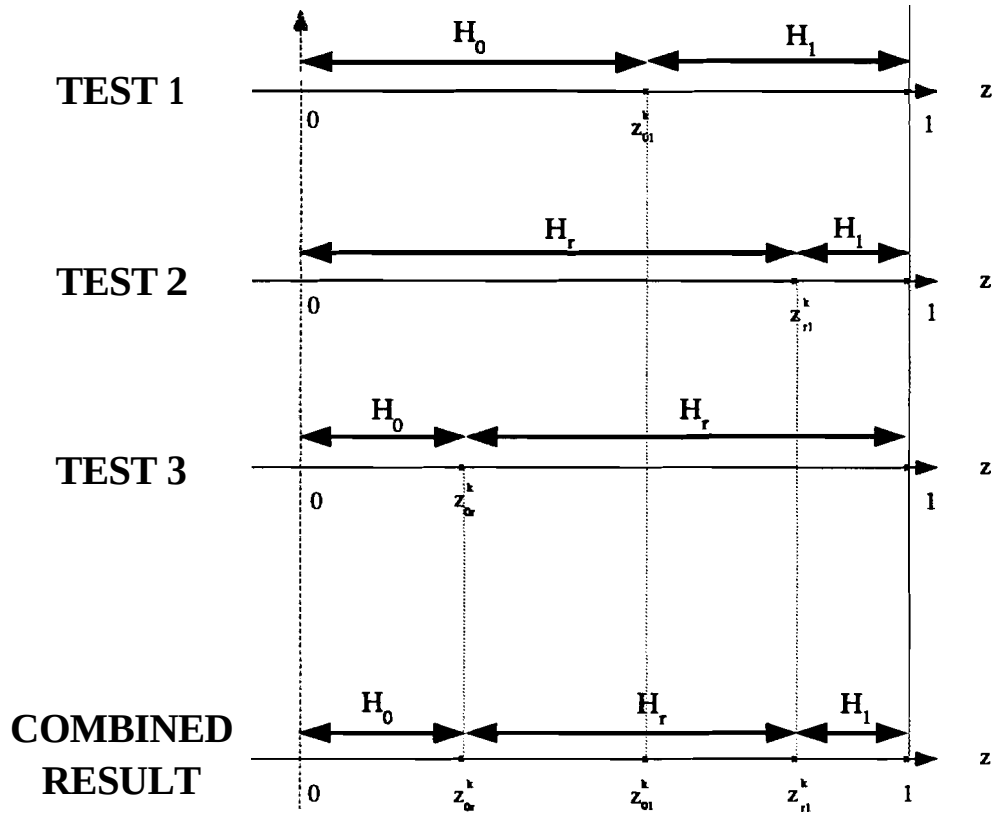


Figure 8. Sample Rejection Boundaries.

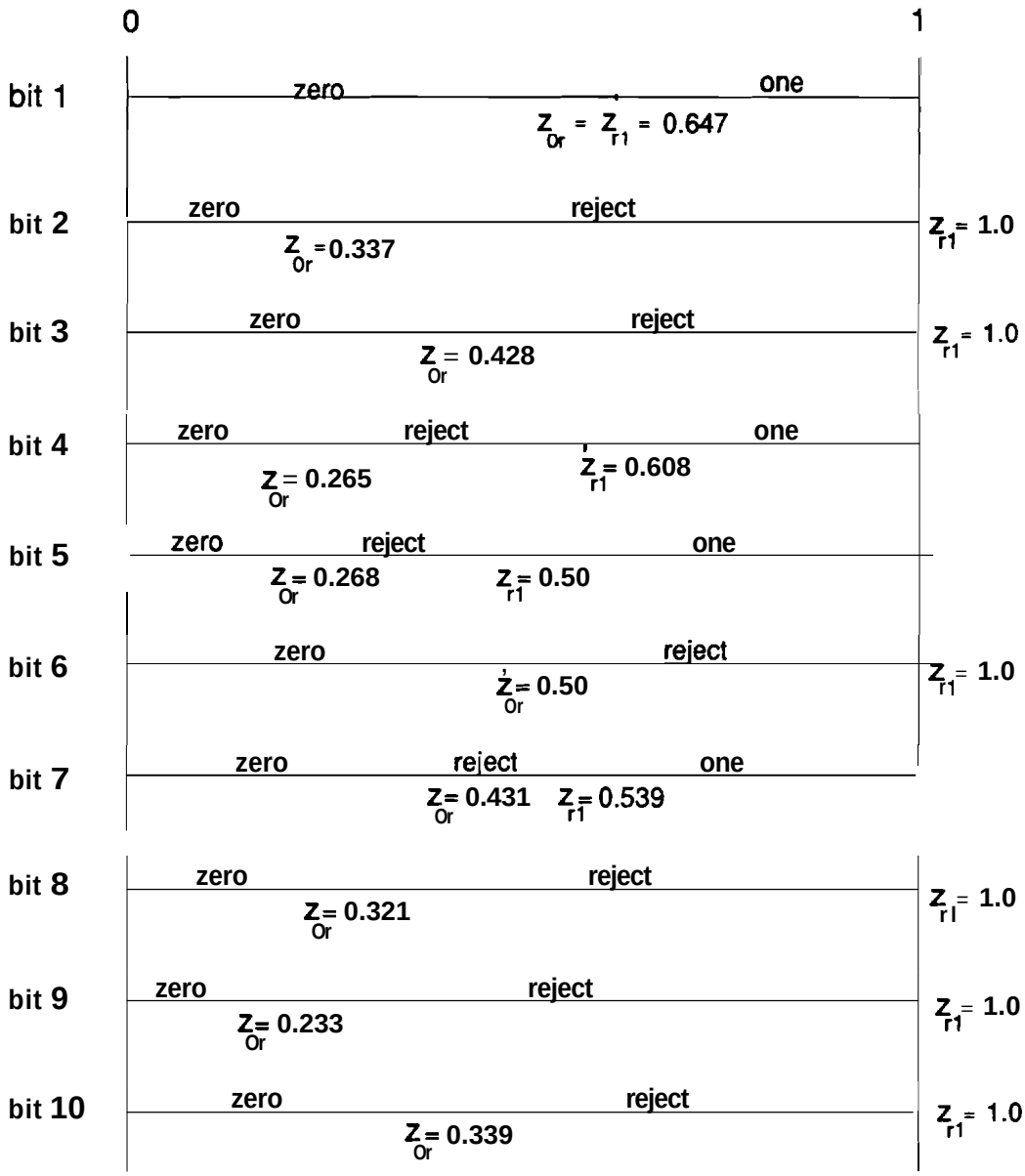


Figure 9. **The Rejection Boundaries for the First NS-unit created for the 10-Class Colorado Problem.**

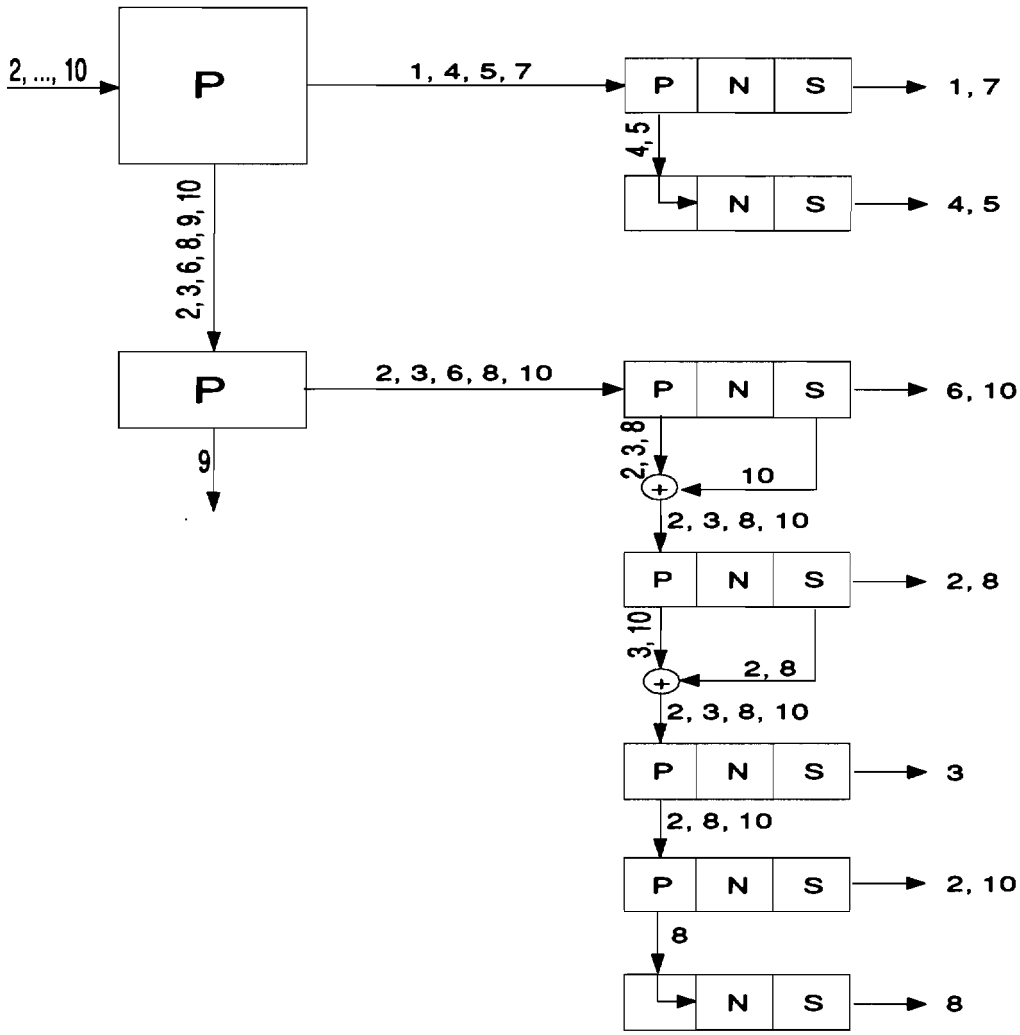
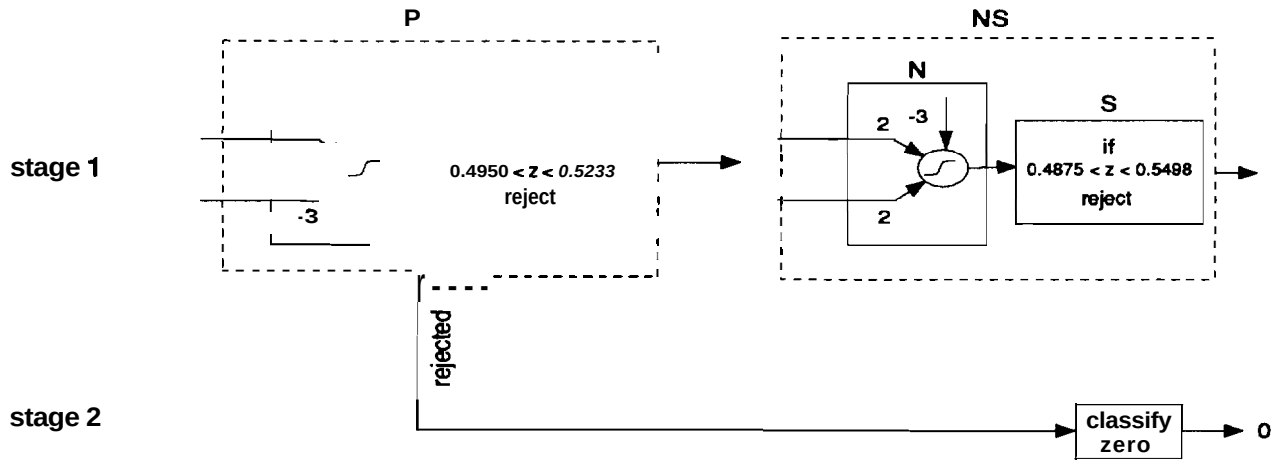
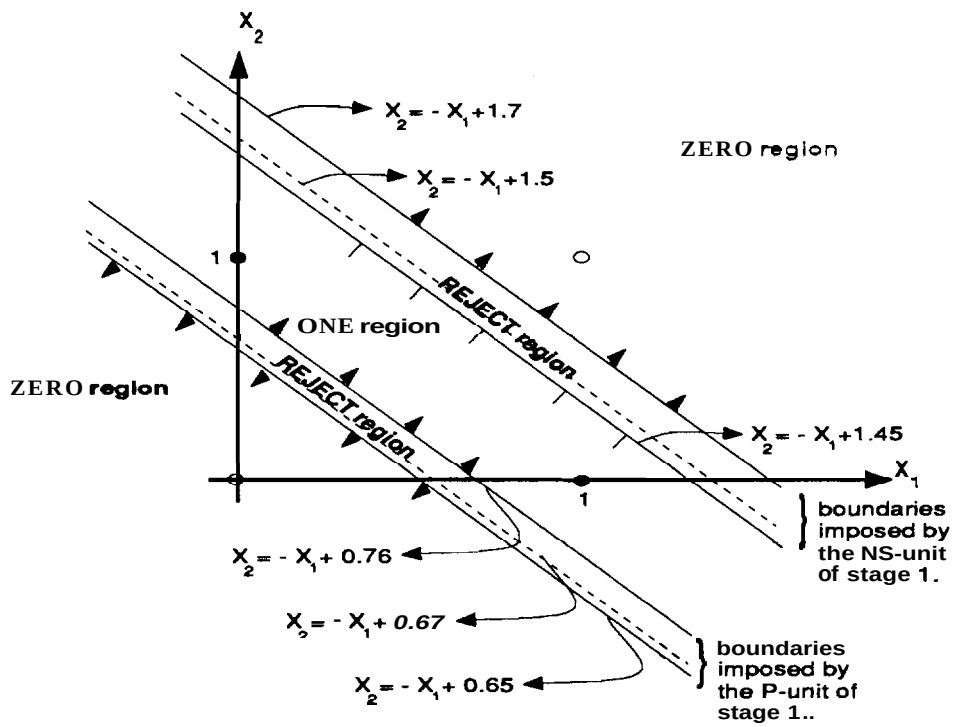


Figure 10. The Class Divisions Generated during Training for the 10-Class Colorado Problem.



(a)



(b)

Figure 11. a) The Network for the XOR Problem,
b) The Accept and the Reject Regions Showing how the System Operates as a Piecewise Linear Model.

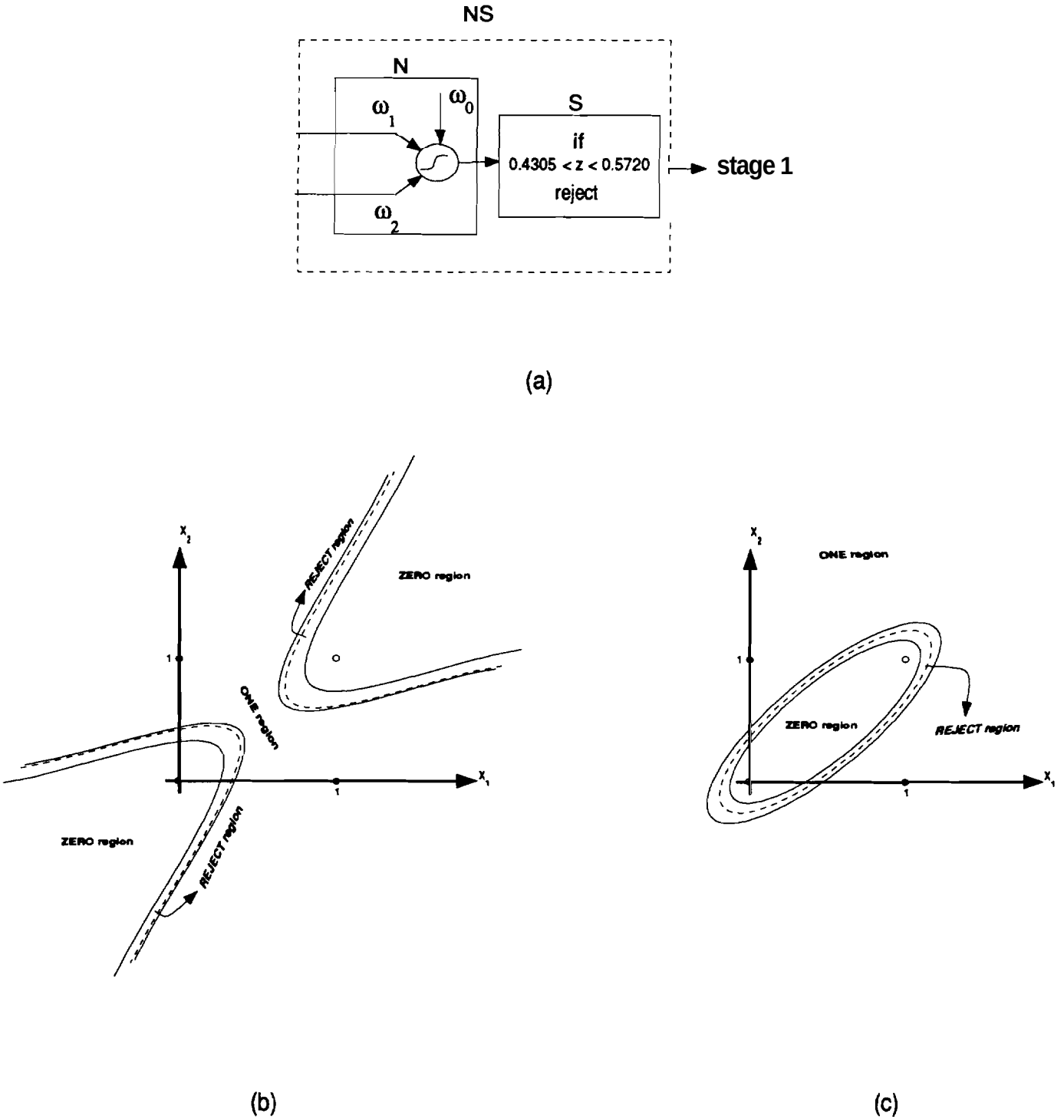


Figure 12. a) A Second Order Polynomial Network for the XOR Problem, b) and c) Possible Accept and Reject Regions.