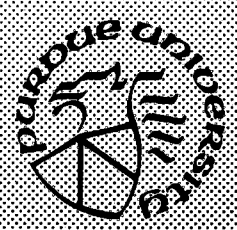
9-1-1989

Parallel, Self-Organizing, Hierarchical Neural Networks

O. K. Ersoy
Purdue University

D. Hong
Purdue University



Parallel, Self-Organizing, Hierarchical Neural Networks

O. K. Ersoy
D. Hong

TR-EE 89-56
September, 1989

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

PARALLEL, SELF-ORGANIZING HIERARCHICAL NEURAL NETWORKS

O. K. Ersoy and D. Hong

School of Electrical Engineering, Purdue University
West Lafayette, IN 47907 U.S.A.

ABSTRACT

A new neural network architecture called the parallel self-organizing hierarchical neural network (PSHNN) is discussed. The PSHNN involves a number of stages in which each stage can be a particular neural network (SNN). At the end of each SNN, error detection is carried out, and a number of input vectors are rejected. Between 2 SNN's there is a nonlinear transformation of those input vectors rejected by the first SNN. The PSHNN has many desirable properties such as optimized system complexity in the sense of minimized self-organizing number of stages, high classification accuracy, minimized learning and recall times, and truly parallel architectures in which all SNN's are operating simultaneously without waiting for data from each other during testing. The experiments performed in comparison to multilayered networks with backpropagation training indicated the superiority of the PSHNN.

1. INTRODUCTION

Some of the most important problems in artificial neural networks are network complexity, learning and recall times, robustness, fault-tolerance and quality of generalization. In this paper, we describe a new neural network (PSHNN) architecture which address these problems.

The PSHNN involves a 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. Between two SNN's, there is a transformation which transforms certain input vectors rejected by the first SNN nonlinearly. The transformed vectors are input to the later SNN. This is probably the most original property of the PSHNN, as distinct from other artificial neural networks.

The motivation for the PSHNN 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 SNN such signals are detected by a scheme and rejected. Then the rejected signals are passed through a nonlinear transformation so that they are converted into other vectors which are more easily classified by the succeeding SNN.

The paper is organized into 6 sections. Sec. 2 discusses signal representations, and how signals belonging to different classes but also linearly nonseparable or close to class boundaries can be transformed nonlinearly so that they become less similar and more easily classifiable. Sec. 3 introduces the PSHNN in detail. Sec. 4 describes the performance results obtained with the PSHNN in comparison to multilayer networks using backpropagation training, in classification experiments with aircraft and satellite remote-sensing data. Sec. 5 discusses the fault-tolerance and robustness properties of the PSHNN. Sec. 6 is conclusions.

2. SIGNAL REPRESENTATIONS AND THEIR NONLINEAR TRANSFORMATIONS

Number representation of signal values is very important in neural networks because it influences a number of factors such as network size, complexity, and quality of performance.

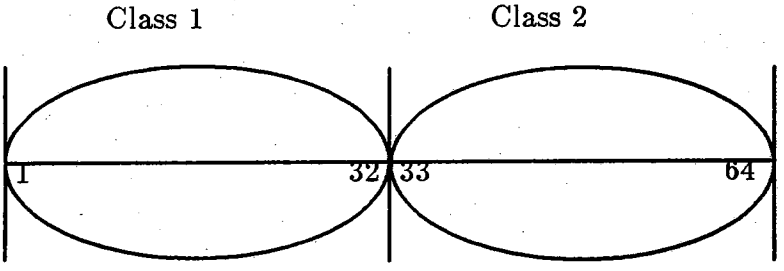


Figure 1. Two classes of integers.

For example, let us consider the problem of two integer classes to be classified. One class consists of the numbers from 1 to 32, the other from 33 to 64, as shown in Fig. 1. There are a number of ways to represent the integers with the binary numbers (1,0) or (1, -1), such as binary coding, temperature coding, and Gray coding [1].

Among these, the temperature coding is one of the most fault tolerant schemes, even though it is not economical in the number of bits. In this scheme, each integer is represented by 1's and 0's (or -1's), the 1's adding up to the integer value and increasing as in a thermometer as the integer gets larger. One important advantage of this scheme is that similar integers are represented similarly. In this section, we will use temperature coding. Another scheme which gives similar representations with similar integers as well as compact representation is the Gray code. It is often used in digital communications. An example of the Gray code is shown in Table 1 [2]. We will use the Gray code in Sec. 4.

The gray code representation can be derived from the binary code representation as follows: If $b_1\ b_2\ \ldots\ b_n$ is a code word in an n-digit binary code, the corresponding Gray code word $g_1\ g_2\ \ldots\ g_n$ is obtained by the rule

Table 1. 3-Bit binary and gray code representation of integers between 0 and 7.

Integer	Binary Code	Gray Code
0	0 0 0	0 0 0
1	0 0 1	0 0 1
2	0 1 0	0 1 1
3	0 1 1	0 1 0
4	1 0 0	1 1 0
5	1 0 1	1 1 1
6	1 1 0	1 0 1
7	1 1 1	1 0 0

$$g_1 = b_1$$
$$g_k = b_k \oplus b_{k-1} \quad k \geq 2$$

where, $\oplus$ is modulo-two addition.

In the 2-class problem discussed above, 64 bits are needed to represent one integer in the temperature coding. Consider only the boundary integers 32 and 33. The difference between these data is just one bit with the temperature coding, but they are in the opposite classes. Such data make classification difficult and causes errors in the presence of noise. However, a more important source of classification errors in one-stage networks is often the linear nonseparability of the classes. An example of linear nonseparability is shown in Fig. 2. The vectors close to the class boundaries, say within the dotted lines, are also difficult to classify. We will call the set of vectors which are difficult to classify or causing errors the "hard" vectors. We can reduce and possibly eliminate classification errors if we can find a transformation of the set of hard vectors into another set in R^n such that they are easier to classify. In so doing, it is important to separate the set of easily classifiable vectors from the set of hard vectors since we want to minimize the possibility of creating new sets of hard vectors.

We will assume the input and output vectors to be binary with elements equal to ± 1 (1 and 0 also possible). The first method for the desired transformation will be achieved by using a fast transform followed by the bipolar thresholding (sign) function given by

$$S'(n) = sgn(S(n)) = \begin{cases} 1 & , S(n) \geq 0 \\ -1 & , otherwise \end{cases} \tag{1}$$

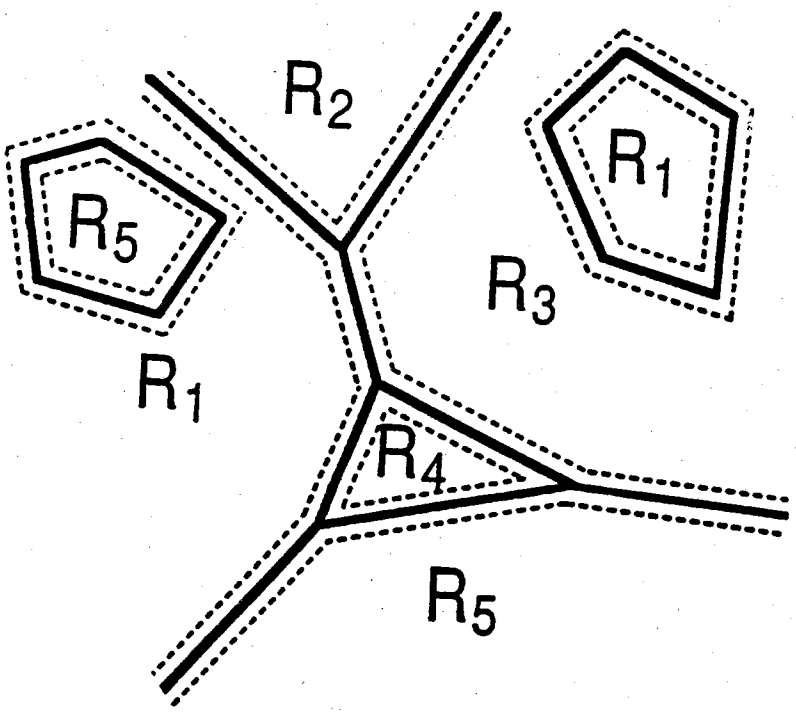


Figure 2. A Five-Class Linearly Nonseparable Problem.

There are a number of fast transforms which can be utilized. Because of its many excellent features, the real discrete Fourier Transform (RDFT) will be utilized in this paper [3]. The one-dimensional RDFT of a sequence x(n) of N data points is given by

$$Y(n) = \sum_{k=0}^{N-1} x(k) \cos\left(\frac{2\pi nk}{N} + \theta(n)\right) \tag{2}$$

where

$$\theta(n) = \begin{cases} 0 & ,0 \leq n \leq \frac{N}{2} \\ \frac{\pi}{2} & ,otherwise \end{cases} \tag{3}$$

The inverse RDFT (IRDFT) is

$$x(n) = \frac{2}{N} \sum_{k=0}^{N-1} Y(k)\, v(k) \cos\left(\frac{2\pi nk}{N} + \theta(n)\right) \tag{4}$$

where

$$v(k) = \begin{cases} \frac{1}{2} & , \; k = 0, \; \frac{N}{2} \\ 1 & , \; otherwise \end{cases} \tag{5}$$

The fast algorithms for the computation of the RDFT are known as the fast real discrete Fourier transform (FRFT) algorithms [4].

The nonlinear transformation using the RDFT is very sensitive to the Hamming distance between the binary vectors; the difference between two binary vectors is changed from 1 bit to many bits after using the nonlinear transformation.

Fig. 3 shows how the similar inputs (32 and 33) trained well after converting them with the nonlinear transformation. In this figure, the error is given by

$$E = \frac{1}{2} \sum_j (do_j - o(W)_j)^2 \tag{6}$$

where do_j is the desired output and $o(W)_j$ is the actual output as a function of the interconnection matrix W. In the first system (dashed line), the input vectors were coded with the temperature scheme. In the second system (solid line), the temperature coded vectors were converted into other binary vectors by the nonlinear transformation. The training was based on the delta rule [5]. It is apparent that the second system converges much faster.

Even though the nonlinear technique discussed above gives excellent performance, its implementation is not trivial. The main purpose of the fast transform followed by pointwise nonlinearities is to "shake up" the hard vectors so that they look different and hopefully become simpler to classify. This can be done by easier techniques. One way is to simplify the fast transform further. For example, the discrete Fourier preprocessing transforms (DFPT's) are obtained by replacing the basis function $cos(\frac{2\pi nk}{N} + \Theta(n))$ in Eq. (2) by a very simple function [6]. There are many DFPT's. The simplest one is Class-2, Type-5 DFPT [7]. Its basis function values are mostly zeros. The nonzero values are ± 1.

The simplest approach, however, to achieve the "shake-up" of the hard vectors is by simply complementing the input vector if it is represented in a binary code. Another simple approach, which can also be used together with complementing, is the scrambling of the binary components of the input vector.

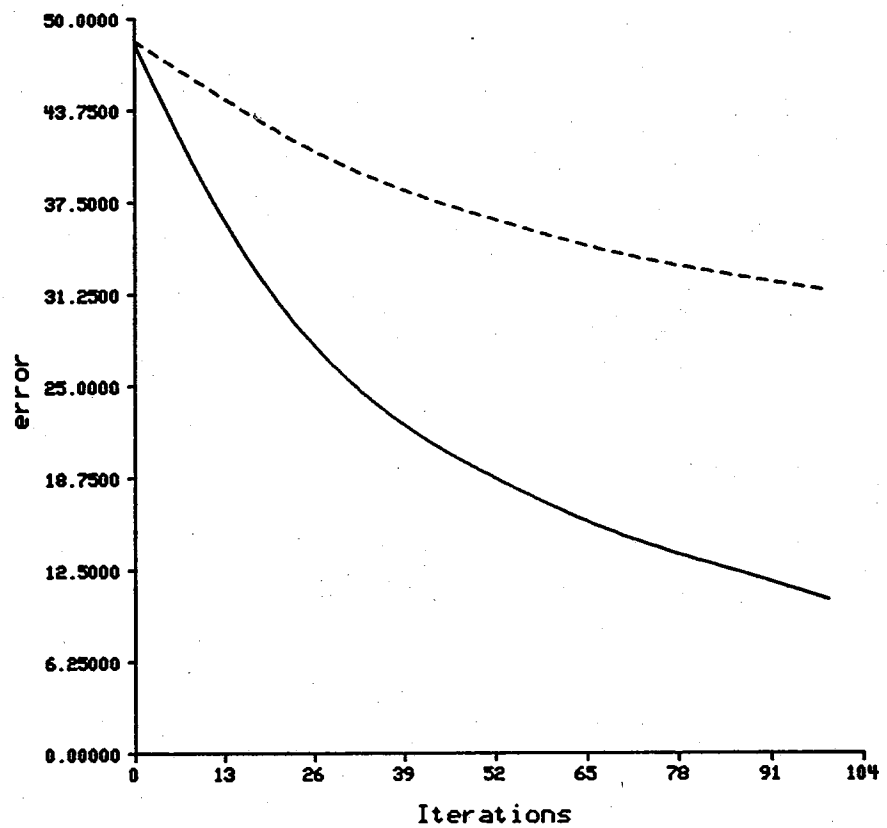


Figure 3. Training time versus error with two systems when the input vectors are close to class boundaries.

3. THE PARALLEL, SELF-ORGANIZING, HIERARCHICAL NEURAL NETWORK

The parallel, self-organizing hierarchical neural network (PSHNN) to be described is envisioned for the purposes of decreasing system complexity, self-organizing the number of stages needed in each application, increasing classification accuracy, avoiding local minima, reducing learning and recall times, obtaining a high degree of robustness and fault-tolerance, and achieving truly parallel architectures.

The PSHNN involves a number of stages, which is self-organizing. Each stage is a particular neural network, to be referred to as the stage neural network (SNN). In the experiments to be described in Sec. 4, each SNN is chosen as a 1-layer network with delta rule learning [8].

Between two SNN's there is a transformation (the nonlinear RDFT or DFPT or complementing methods in the experiments to be discussed) which transforms those input vectors rejected by the first SNN nonlinearly, using the procedures described in the last section. The rejection and acceptance of input vectors is done according to a procedure described below.

3.1 Training

In order to speed up learning, the upper limit of iterations in each SNN during learning will be restricted to an integer k. If k is arbitrarily large, then the SNN usually reaches convergence after a certain number of iterations. Let us assume that the i-th SNN is denoted by SNN(i). Below we describe the training procedure:

TRAINING PROCEDURE

Assume : *The number of iterations is upper-bounded by k for each SNN.*

Initialize : $i = 1$

1. *Train SNN(i) by a maximum of k iterations.*
2. *Check the output for each input vector.*
   1) *If no error --> stop the training*
   2) *If errors --> get the reference values and go to step 3.*
3. *Select the data which are within the rejection bounds.*
   1) *If all the chosen data are in one class, then define the final reference value (FRV) as that class. Stop the training.*
   2) *If not, go to step 4.*
4. *Take the nonlinear transform(NLT) of the rejected data set. Increase i by 1. Go to step 1.*

The reference values referred to in step 2 above are the bounds (thresholds) discussed in the section below.

3.2 Detection of Potential Errors

How do we reject and accept input vectors at each SNN? The output neurons yield 1, 0 (or −1) as their final value. The decision of which binary value to choose involves thresholding. It is possible to come up with a number of decision strategies. Below we will describe a particular algorithm.

The value obtained after the weighted summation at the i-th output neuron is first passed through the sigmoid function defined by Eq. (7) to give a value y_i between 0 and 1:

$$f(x) = \text{sigmoid}\ (x) = \frac{1}{1 + e^{-x}} \tag{7}$$

The parameter x in the sigmoid function f(x) is actually the weighted summation plus a threshold term θ, which is trained by using an extra input neuron whose input is 1. The final output value z_i is obtained by the hard-limiter:

$$z_i = \begin{cases} 1, & \text{if} \quad y_i \geq 0.5 \\ 0, & \text{if} \quad y_i < 0.5 \end{cases} \tag{8}$$

In this process, it is assumed that the desired output of the system is represented by a binary number.

It is observed that there are 3 vectors involved: the input vector X, the vector Y with elements y_i, and the output vector Z with elements z_i. We can also show time-dependence by using subscript i in the form X^i, Y^i, Z^i.

After training the SNN by a maximum of k iterations, we compare the output vector Z with the desired output vector Z_d. If they are different from each other, the input vector is counted as an "error-causing" vector of the SNN.

The set of error-causing vectors are the input to the next SNN after being processed by the nonlinear transformation technique discussed in Sec. 3.

Now we need an algorithm to detect potential errors during testing. For this, we will define error bounds and no-error bounds. The following is the current procedure for estimating the error bounds during training:

ERROR BOUNDS

Assume : *number of data vectors* $= l$
length of vector $= n$
$y_j^i = j$*—th component of the i-th vector* Y^i

Initialize the reference values y_j^0*:*

$$\begin{cases} y_j^0(upper) = 0.5 \\ \qquad\qquad\qquad\qquad , \ where\ j = 1,2,\ldots,n \\ y_j^0(lower) = 0.5 \end{cases}$$

Initialize : $i = 1$

1. *Check whether the i-th data vector is an error-causing vector. If so,*
   1) *If* $y_j^i \geq 0.5$, *then* $y_j^i(upper) = MAX[y_j^{(i-1)}(upper), y_j^i]$
   2) *If* $y_j^i \leq 0.5$, *then* $y_j^i(lower) = MIN[y_j^{(i-1)}(lower), y_j^i]$
3. *If* $i = l$ *then*
   $r_j(upper) = y_j^0(upper)$
   $r_j(lower) = y_j^0(lower)$
   else $i = i + 1$ *and go to step 1.*
   End

r_j(upper) and r_j(lower) in step 3 above are the reference values (error bounds).

In pattern recognition, we often use basis vectors of R^n as the desired outputs to denote classes when the number of classes is n. In other words, the desired output of each class can be represented as

$$\begin{aligned} \text{class 1} &\longrightarrow (1, 0, 0, \ldots, 0)^T \\ \text{class 2} &\longrightarrow (0, 1, 0, \ldots, 0)^T \\ &\vdots \\ \text{class n} &\longrightarrow (0, 0, \ldots, 0, 1)^T \end{aligned}$$

In this case, an input vector is classified as an error-causing vector if any of the output bits is incorrect. The binary scheme described above can be replaced by any other suitable binary scheme. Each output neuron i can also be duplicated a number of times before finding z_i , and the final thresholding to find z_i can be done after summing them to increase accuracy.

The simplest rejection procedure during testing is to check whether the vector Y is within the error bounds or not. If it is, then the corresponding input data vector is rejected. During testing, some misclassified data may not be rejected since they are not within the error bounds. Simultaneously some correctly classified data may also be rejected since they are within the error bounds. These sources of error can be further reduced by simultaneously utilizing no-error bounds. The following is the current procedure for estimating

the no-error bounds:

NO-ERROR BOUNDS

Initialize the reference values y_j^0*:*

$$\begin{cases} y_j^0(upper) = 0.5 \\ \qquad\qquad\qquad\qquad , \text{ where } j = 1,2,\ldots,n \\ y_j^0(lower) = 0.5 \end{cases}$$

Initialize : $i = 1$

1. *Check whether the i-th data vector is an error-causing vector. If so,*
   then $i = i + 1$ *and go to step 1*
   else go to step 2.
2. *Get the reference values for* $j = 1,2,\ldots,n$.
   1) *If* $y_j^i > 0.5$, *then* $y_j^i(upper) = MIN[y_i^{(i-1)}(upper), y_j^i]$
   2) *If* $y_j^i < 0.5$, *then* $y_j^i(lower = MAX[y_j^{(i-1)}(lower), y_j^i]$
3. *If* $i = l$ *then*
   $r_j(upper) = y_j^0(upper)$
   $r_j(lower) = y_j^0(lower)$
   else $i = i + 1$ *and go to step 1.*
   end

With the no-error bounds, the rejection procedure can be to check whether the vector Y is not in the correct region determined by the no-error bounds. If it is not, then the corresponding input data vector is rejected. However, a more accurate procedure is to utilize both the error and the no-error bounds. In this case, we check whether the vector Y is within the error-bounds *and* not in the correct region determined by the no-error bounds. If so, the corresponding input data vector is rejected. With this procedure, better accuracy is achieved because correctly classified data vectors are not rejected even if they are within the error bounds. However, some error-causing data vectors can still be among those not rejected since they satisfy the above criterion.

3.3 Testing

Testing (recall) with the PSHNN is similar to testing 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 SNNs with vectors which are correctly classified. The following describes the testing procedure:

TESTING PROCEDURE

Initialize : $i = 1$

1. *Input the test vector to SNN(i).*
2. *Check whether the output indicates an error-causing input data vector. If so, then,*
   a) *if it is the last SNN, then classify with the FRV*
   b) *if it is not, nonlinearly transform the input test vector and go to step 1*

   else, classify the output vector.

An interesting observation is that the testing 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. This is shown in Fig. 4 in case of 3 SNNs. In this scheme, the test input is multiplexed into 3 inputs in terms of itself and the 2 additional vectors obtained by the nonlinear transformations NLT1 and NLT2. Then, these 3 inputs are fed into SNN1, SNN2 and SNN3, respectively. The logic unit chooses one of the 3 outputs from the SNNs as the correct output and passes it to the hard limiter.

The logic unit is shown in Fig. 5. The output of each Ref(i) is zero whenever error is detected. Each switch SW(i) allows its input through only if its control bit is 1. In this way, one of the 4 possible outputs (including FRV) is demultiplexed through the logic unit.

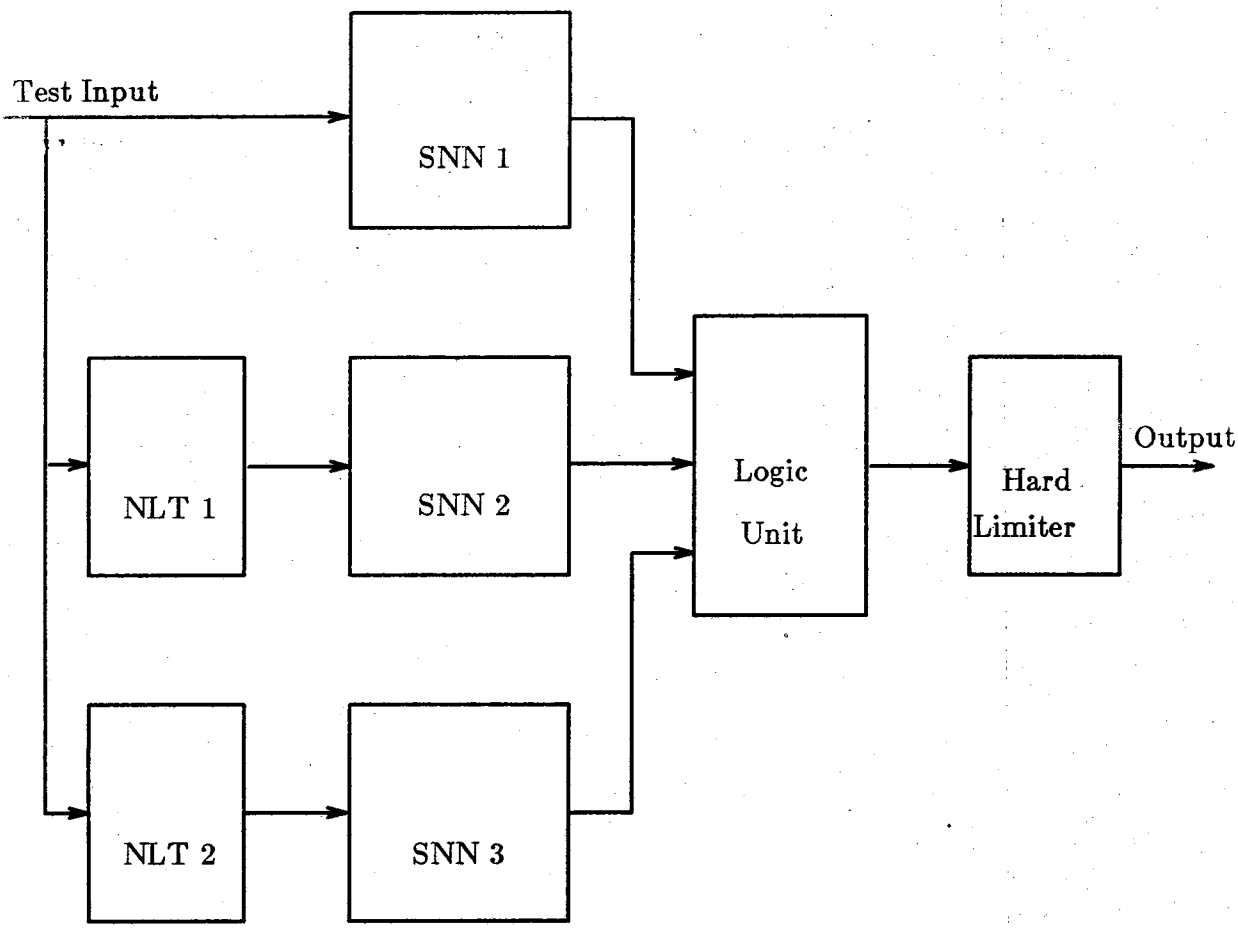


Figure 4. Block diagram of the parallel testing procedure with the PSHNN when the number of SNN's is 3.

4. EXPERIMENTAL RESULTS

In this section, we will describe experiments to study the performance of the PSHNN in comparison to 3-layer and 4-layer networks with backpropagation training [8]. Each SNN of the PSHNN was chosen as a 1-stage network with delta rule training. The initial weights were randomly chosen in the range from -0.5 to 0.5. The gain parameter was chosen as 0.05.

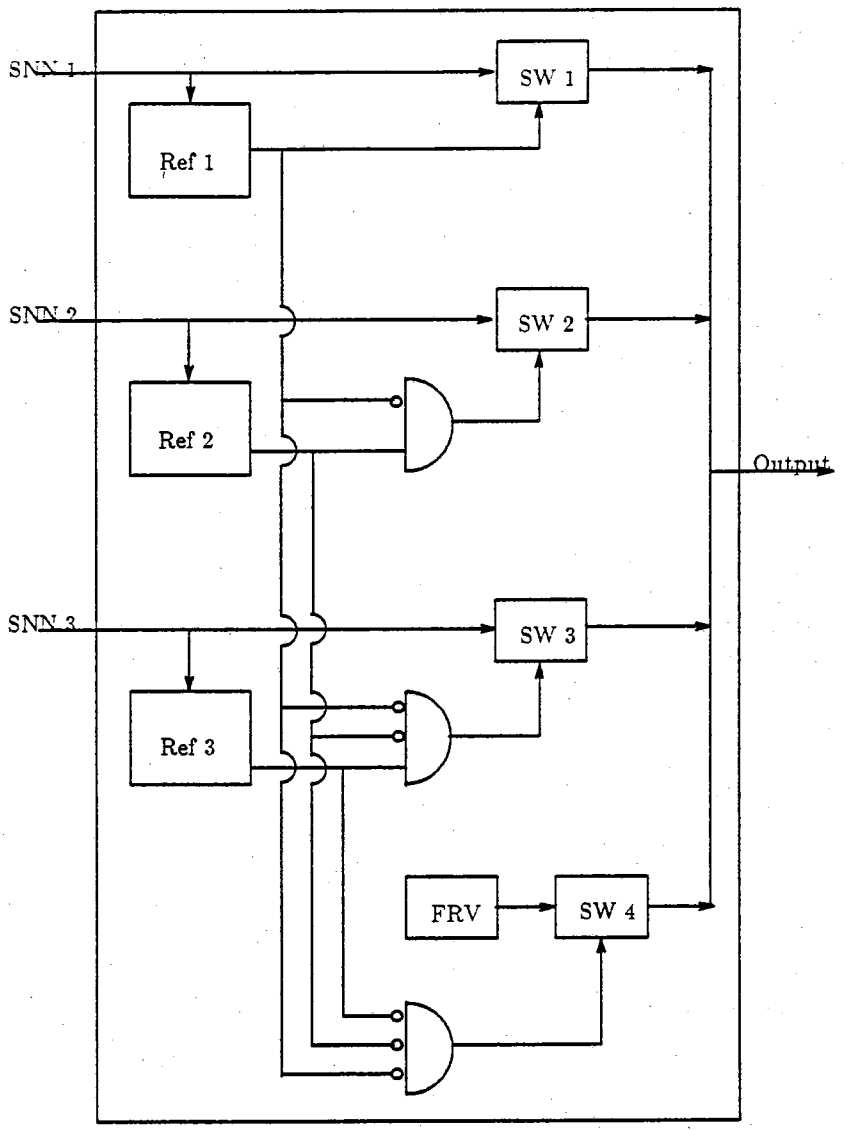


Figure 5. Logic unit of the PSHNN when the number of SNNs is 3 (parallel scheme).

Two sets of remotely sensed data sources were used in the experiments. The first set was aircraft remotely-sensed image data, called Flight Line C1, covering the southern part of Tippecanoe county, Indiana [9]. The second set was multispectral earth observation remotely sensed data covering a mountainous area in Colorado [10].

4.1 Experiments with the First Set of Data

The first set of data had 12 bands. 8 spectral bands out of 12 bands were chosen representing 8 classes of farm products (alfalfa, corn, oats, red clover, soybean, wheat, bare soil, rye). Each band signal value represents 256 gray levels, requiring a minimum of 8 bits for binary representation. Each signal value is generated by concatenating the corresponding band signal values together, thus requiring 64 bits for binary representation. The Gray code was used for representing the input signals.

In the experiments, we first tried to answer the question as to whether the nonlinear transformations are really useful in the construction of the PSHNN. When we removed the RDFT-based nonlinear transformations, 9 SNN's were required, as compared to 3 SNN's with the nonlinear transformations included in the PSHNN.

Table 2. Number of rejected data of PSHNN1 at each SNN when the allowed number of iterations is 70 in the 4-class problem (data set I).

Stage	No. rejected data				Total number
	class 1	class 2	class 3	class 4	
0	200	200	200	200	800
SNN 1	7	150	34	45	236
SNN 2	1	25	10	6	42
SNN 3	0	1	0	0	1

Training and testing with the networks were done with 200 signal samples per class and 375 other signal samples per class, respectively. Two sets of experiments were generated in terms of a 4-class problem and an 8-class problem. The first experiments to be described below with Tables 2 through 11 were carried out with the RDFT-based nonlinear transformations of the rejected vectors.

In the first set of experiments, 2 PSHNNs were constructed; the first one (PSHNN1) was based on the error bounds only, and the second one (PSHNN2) was based on both the error and the no-error bounds.

Table 2 shows the number of rejected data vectors at each SNN of the PSHNN1, when the allowed number of maximum iterations is 70. Table 3 shows the number of SNNs used and the classification accuracy of PSHNN1 as a function of the allowed number of iterations.

Table 4 shows the number of rejected data at each SNN of the PSHNN2, when the number of maximum iterations is 90. Table 5 shows the number of

Table 3. The classification accuracy of PSHNN1 and the total number of SNN's as a function of the number of iterations in the 4-class problem ($\eta = 0.05$) (data set I).

Number of Iterations	Number of SNNs	Classification Accuracy	
		Train (%)	Test (%)
5	6	99.98	82.60
10	5	99.98	86.33
20	4	99.98	87.87
30	3	99.98	87.60
40	3	99.98	87.87
50	3	99.98	89.07
60	4	99.98	88.33
70	3	99.98	89.27
80	4	99.98	88.07
90	4	99.98	87.73
100	4	99.98	87.87
150	3	99.98	87.93
200	3	99.98	87.00

Table 4. Number of rejected data of PSHNN2 at each SNN when the allowed number of iterations is 90 in the 4-class problem (data set I).

Stage	No. rejected data				Total number
	class 1	class 2	class 3	class 4	
0	200	200	200	200	800
SNN 1	7	10	25	6	48
SNN 2	0	0	0	0	0

SNNs used and the classification accuracy of PSHNN2 as a function of the allowed number of iterations.

The number of SNNs of the PSHNN2 is found to be always smaller than that of the PSHNN1. The classification accuracy during testing of the PSHNN2 is also better than that of PSHNN1. The best classification accuracy of the PSHNN1 is 89.27 % when the maximum number of iterations is 70 and the number of SNNs is 4. The corresponding values for PSHNN2 is 92.13 % accuracy with 90 iterations and 2 SNNs. The classification accuracy for training samples of PSHNN1 is always almost perfect, but during the recall

Table 5. The classification accuracy of the PSHNN2 and the total number of SNN's as a function of the number of iterations in the 4-class problem ($\eta = 0.05$) (data set I).

Number of Iterations	Number of SNNs	Classification Accuracy	
		Train (%)	Test (%)
5	4	95.38	89.93
10	3	96.12	89.73
20	3	96.50	89.67
30	2	97.50	90.40
40	2	97.87	90.73
50	2	97.88	91.20
60	2	98.63	91.87
70	2	98.75	91.93
80	2	98.75	92.00
90	2	98.75	92.13
100	2	98.50	92.03
200	2	99.00	90.80

Table 6. Number of rejected data of PSHNN2 at each SNN when the allowed number of iterations is 100 in the 8-class problem (data set I).

Stage	No. rejected data								Total
	c 1	c 2	c 3	c 4	c 5	c 6	c 7	c 8	
0	200	200	200	200	200	200	200	200	1600
SNN 1	5	28	34	8	3	28	0	4	110
SNN 2	2	10	5	0	0	2	0	1	20
SNN 3	0	0	0	0	0	0	0	0	0

procedure, its accuracy is not as good as that of PSHNN2. An interesting observation is that the accuracy of the network increases as the number of iterations increases until some point. After that, the accuracy starts to decrease.

The second set of experiments in terms of 8 classes resulted in the same type of results as the 4-class case. Thus, PSHNN's appear to scale nicely as the number of classes increase. Table 6 shows the number of rejected data vectors at each SNN of the PSHNN2, when the allowed number of maximum

Table 7. The classification accuracy of PSHNN2 and the total number of SNN's as a function of the number of iterations in the 8-class problem ($\eta = 0.05$) (data set I).

Number of Iterations	Number of SNNs	Classification Accuracy	
		Train (%)	Test (%)
5	6	94.19	85.06
10	4	95.25	85.63
20	4	96.25	86.46
30	3	96.81	86.67
40	3	97.87	87.33
50	3	97.25	88.01
60	3	97.37	88.47
70	3	97.38	88.87
80	3	97.38	88.90
90	3	97.44	89.13
100	3	97.75	89.30
200	3	97.63	88.67

iterations is 100. Table 7 shows the number of SNNs used and the classification accuracy of PSHNN2 as a function of allowed number of iterations. The best classification accuracy 89.30 % is obtained at 100 iterations with 3 SNN's.

The performance of PSHNN's were compared to the performance of multilayer neural networks using backpropagation algorithm with 3 layers (3NN) and 4 layers (4NN). In both cases, the length of each layer was fixed as 64. The initial weights were randomly chosen between -0.5 and 0.5. Again, 2 sets of experiments were performed in terms of 4 classes and 8 classes, respectively. The scaling parameter η for updating weights was chosen as 0.01 and 0.02 in the first and the second set of experiments, respectively.

The results of the first set of experiments are shown in Tables 8 and 9, in terms of the classification accuracy of the 3NN and the 4NN as a function of allowed number of iterations, respectively.

It is observed that the performance of the 4NN is always better than that of the 3NN. The best classification accuracy 92.80 % during testing is obtained with about 800 iterations for the 4NN. After that, the classification accuracy during testing is decreasing even though the classification accuracy of the training samples is increasing. For the 3NN, the best accuracy rate 89.07

Table 8. The classification accuracy of 3NN as a function of the number of iterations in the 4-class problem ($\eta = 0.01$) (data set I).

Number of Iterations	Classification Accuracy	
	Train (%)	Test (%)
100	80.75	73.60
500	91.25	86.07
1000	92.62	87.87
1500	93.37	88.26
2000	93.88	88.27
2500	93.88	88.60
3000	93.50	88.93
3500	93.37	89.07
4000	93.50	89.07
5000	93.74	88.93

Table 9. The classification accuracy of the 4NN as a function of the number of iterations in the 4-class problem ($\eta = 0.01$) (data set I).

Number of Iterations	Classification Accuracy	
	Train (%)	Test (%)
100	87.75	84.74
500	99.63	92.47
800	99.88	92.80
900	99.88	92.80
1000	99.88	92.60

% is obtained around 3500 - 4000 iterations.

The classification accuracy of the PSHNN2 is almost the same as that of 4NN and better than that of 3NN. If we assume the computation time of one pass in a one-stage network is 1, the training time for the PSHNN2 is 800 x 90 for SNN1 and 48 x 90 for SNN2, totaling 76,320. For the 4NN, to get the best accuracy, the computation time is 800 x 800 x 3 = 1,920,000, which is about 25 times longer than the time for the PSHNN2. In this estimation, the computation time for the nonlinear transformation of the rejected vectors in the PSHNN2 is not included since it involves fast, constant and simple operations.

Table 10. The classification accuracy of 3NN as a function of the number of iterations in the 8-class problem ($\eta = 0.002$) (data set I).

Number of Iterations	Classification Accuracy	
	Train (%)	Test (%)
100	51.00	44.27
500	78.19	71.47
1000	81.62	75.60
1500	83.63	77.23
2000	84.37	78.23
2500	84.87	78.97
3000	85.19	79.33
3500	85.44	79.63
4000	85.68	79.70

Table 11. The classification accuracy of 4NN as a function of the number of iterations in the 8-class problem ($\eta = 0.002$) (data set I).

Number of Iterations	Classification Accuracy	
	Train (%)	Test (%)
100	86.44	82.22
500	94.00	88.70
600	95.00	88.76
700	96.00	89.17
800	97.00	89.50
900	97.93	89.50
1000	98.31	89.47
1500	98.93	89.49

The results of the second set of experiments with 8 classes are shown in Tables 10 and 11, in terms of the classification accuracy of the 3NN and the 4NN as a function of allowed number of iterations, respectively. The best accuracy rate of 79.7% with the 3NN is obtained around 4000 iterations. The best accuracy rate with the 4NN is 89.5% at around 900 iterations.

Again the classification accuracy of the PSHNN2 is basically the same as that of 4NN and much better than that of 3NN. If we assume the computation time of one pass in a one-stage network is 1, the training time for the PSHNN2 is 1600 x 100 for SNN1, 110 x 100 for SNN2, and 20 x 100 for SNN3,

totaling 173,000. For the 4NN, to get the best accuracy, the computation time is 1600 x 900 x 3 = 4,320,000, which is about 25 times longer than the time for the PSHNN2.

In Sec. 2, we discussed the simplication of the nonlinear transformations of the rejected vectors in order to achieve easier hardware and software implementations. Table 12 shows the classification accuracy of the PSHNN2 and the total number of SNN's as a function of the number of iterations in the 4-class problem with the Class-2, Case 5 DFPT-based nonlinear transformation. Table 13 shows the same type of results with the complementing of rejected vectors for their nonlinear transformations. Tables 12 and 13 can be compared to Table 5. The RDFT-based nonlinear transformation gives only 1 % better accuracy in testing than the DFPT-based nonlinear transformation. Complementing the rejected vectors, the simplest approach, is almost as good as the RDFT-based approach with this data set.

Table 12. The classification accuracy of the PSHNN2 and the total number of SNNs as a function of the number of iterations in the 4-class problem ($\eta = 0.05$) (data set I). Class-2 Case-5 DFPT-based nonlinear transformation of rejected vectors was used.

Number of Iterations	Number of SNNs	Classification Accuracy	
		Train (%)	Test (%)
10	3	96.12	90.26
20	2	96.50	91.06
30	2	97.50	90.73
40	3	97.87	90.53
50	3	97.88	91.20
60	2	98.63	90.93
70	3	98.75	90.86
80	3	98.75	90.93
90	3	98.75	91.13
100	3	98.50	90.08
200	2	99.00	90.46

Table 13. The classification accuracy of the PSHNN2 and the total number of SNNs as a function of the number of iterations in the 4-class problem ($\eta = 0.05$). Complementing of rejected vectors was used for nonlinear transformation of rejected vectors (data set I).

Number of Iterations	Number of SNNs	Classification Accuracy	
		Train (%)	Test (%)
10	5	96.37	91.34
20	4	96.62	90.46
30	3	97.00	90.80
40	3	97.00	91.00
50	4	98.12	91.00
60	2	98.25	91.40
70	3	98.25	91.60
80	3	98.99	91.06
90	3	98.50	92.06
100	3	98.50	91.60
200	2	98.87	91.87

4.2 Experiments with the Second Set of Data

The second set of data consists of the following 4 data sources:

1) Landsat MSS data (4 data channels)
2) Elevation data (in 10 m contour intervals, 1 data channel)
3) Slope data (0-90 degrees in 1 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 are in Colorado. It has 10 ground cover classes which are listed in Table 14. Each channel 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 [10]. By this method, 2019 ground reference points (11.4 % of the area) were selected. Ground reference consisted 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

Table 14. 10 dominant classes of the multispectral image data of Colorado area.

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

shown in Table 15.

Table 15. The number of the samples of each class of the multispectral image data of Colorado area.

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

The neural network input data vector is constructed with 7 binary vectors, one from each band. Therefore, each input data vector has 56 bits.

Table 16. The classification accuracy of the 4NN as a function of the number of iterations in the 10-class problem ($\eta = 0.01$) (data set II).

Number of Iterations	Classification Accuracy	
	Train (%)	Test (%)
100	71.04	46.32
500	89.06	51.87
1000	92.42	53.30
1500	93.68	54.03
2000	93.86	54.27
2500	94.11	53.79

Table 17. The classification accuracy of the PSHNN2 and the total number of SNNs as a function of the number of iterations in the 10-class problem ($\eta = 0.05$). Complementing was used for nonlinear transformation of rejected vectors (data set II).

Number of Iterations	Number of SNNs	Classification Accuracy	
		Train (%)	Test (%)
10	8	95.37	55.23
20	10	95.79	56.19
30	5	95.53	55.00
40	5	95.45	54.75
50	4	95.71	56.43
60	4	95.20	53.06
70	5	95.20	56.20
80	5	94.45	55.23
90	4	94.45	55.35
100	5	94.45	56.31
130	4	94.45	57.03
200	4	95.20	56.50

In the case of the multilayer neural network using backpropagation algorithm with 4 layers (4NN), the length of the input layer and the hidden layers were fixed as 56, and the length of the output layer was 10. The initial

Table 18. The classification accuracy of the PSHNN2 and the total number of SNNs as a function of the number of iterations in the 10-class problem ($\eta = 0.05$) when the rejected vectors are not classified at the last stage (data set II).

Number of Iterations	Number of SNNs	Classification Accuracy Test (%)	Number of Rejected vectors
10	8	55.23	9
20	10	56.26	1
30	5	56.00	15
40	5	54.75	0
50	4	56.43	0
60	4	55.61	38
70	5	57.02	12
80	5	56.52	19
90	4	58.52	45
100	5	56.31	0
130	4	57.03	0
150	4	59.58	45
200	4	57.95	20

weights were randomly set between - 0.5 and 0.5. Table 16 shows the accuracy of the 4NN ($\eta = 0.01$) as a function of allowed number of iterations. The best classification accuracy 54.03 % was obtained with 1500 iterations.

In case of the PSHNN, the length of the input vector was 56 and the length of the output vector was 10. Rejected input vectors of an SNN were nonlinearly transformed with the complementing method and fed into the next SNN. The initial weights were randomly set in the range -0.5 to 0.5. The gain parameter for delta rule training was chosen as 0.05. Table 17 shows the number of SNNs used and the classification accuracy of PSHNN as a function of allowed number of iterations. The best classification accuracy of the PSHNN was 57.03 % when the maximum number of iterations selected was 130 and the number of SNNs was 4.

The number of SNN's is determined during training by the criterion that no input vectors belonging to more than one class are rejected with the last SNN. However, the error and the no-error bounds are still determined in the last SNN. These bounds can be used during testing in order not to classify

those vectors which indicate error in the last stage according to the bounds criterion discussed in Sec. 3.2. Table 18 shows the results when this is done. The best result is achieved with 59.58 % accuracy, an improvement of 2.55 % over the best result in Table 17. However, this is obtained at the cost of not classifying 45 input vectors.

Classification with the second set of data is a very difficult problem. During learning, there is only one training field available for each information class. Hence, the accuracy obtained is considered respectable [10].

5. PROPERTIES OF FAULT-TOLERANCE AND ROBUSTNESS

One of the most important considerations in the evaluation of artificial neural networks is their degree of fault tolerance and robustness. There are two major mechanisms by which the network connections may have weights different from the optimal ones. The first mechanism is due to the limitations of the technology of implementation. For example, in analog electronic and electro-optical implementations, it is difficult to achieve dynamic range greater than several bits. In all technologies, larger number of bits means more complex circuitry.

The second mechanism occurs when the weights are computed by a training set of input-output vectors, but the signal statistics change, and the weights need to be updated. In a large network, this is by no means trivial to achieve in real time.

We studied the robustness and the fault-tolerance properties of the PSHNN's in comparison to multilayer networks with backpropagation training by adding noise to the calculated values of the weights. Both additive and multiplicative noise were utilized.

The PSHNN was adapted to the corruption of the weight values by modifying the four bounds per output neuron at each stage. The procedure is as follows

A. The weights of a stage are learned.

B. The weights are corrupted.

C. The bounds are learned.

D. The procedure A to C is repeated for the next stage.

In the case of the corruption of the weight values by additive Gaussian noise, the amount of noise was measured in terms of the signal-to noise ratio(SNR). It is defined by

$$SNR = 10.0 \log_{10}\left(\sqrt{\frac{P_w}{P_n}}\right) \tag{9}$$

where, P_n is the average power of the noise signal, and P_w is the average power of the stage weight values. P_w is defined by

$$P_w = \frac{1}{N}\sqrt{\sum_{i}^{n}\sum_{j}^{m} w_{i,j}^{2}} \tag{10}$$

where, $w_{i,j}$ is the (i,j)-th weight value, and N is the total number of weights in the stage.

The performance of the PSHNN in comparison to the multilayered network with backpropagation training was studied through experiments with the first set of aircraft multispectral image data of the Purdue area.

Table 19 shows the classification accuracy of the 4-layer neural network with the backpropagation algorithm (4NN) when the number of iterations was 1000. As the SNR is decreased, the classification accuracy of the 4NN is also observed to decrease rapidly. Below -3.01 db, the system is almost useless. Even at -3.01 db, the classification accuracy of the 4NN for the testing samples is 85.33 %, considerably less than that of the 4NN with no errors (92.60 %).

Table 20. shows that the classification accuracy of the PSHNN when the maximum allowed number of iterations was 90. The RDFT based nonlinear transformation of rejected vectors was utilized. The multispectral image data of Purdue area is used for this experiment.

With the PSHNN, the classification accuracy decreases much more slowly than with the 4NN as the SNR decreases. 87.60 % accuracy is achieved with the testing samples when the SNR is -9.54 db, at which the power of the noise

Table 19. The classification accuracy of the 4NN as a function of the number of iterations in the 4-class problem ($\eta = 0.01$). Weights are corrupted by additive GAussian noise. The number of iterations is 1000 (data set I).

SNR (db)	Classification Accuracy	
	Train (%)	Test (%)
no errors	99.88	92.60
20.0	99.87	92.53
10.0	99.50	92.33
3.01	97.75	91.00
0.00	96.62	88.60
-3.01	92.74	85.33
-4.77	75.37	76.06
-6.02	58.62	70.73
-6.98	70.87	68.13
-7.78	46.87	28.93
-8.45	53.62	43.86
-9.03	36.25	42.80
-9.54	38.24	50.34
-10.0	40.75	16.73

signal is 9 times bigger than that of the weight values. However, the number of stages also grow as the SNR is further decreased.

In the case of the corruption of the weight values by multiplicative noise, the corrupted weight values are obtained according to

$$\tilde{w}_{ij} = (1 + \alpha)\, w_{ij} \tag{11}$$

where, w_{ij} is the trained weight value, $\tilde{w}_{ij}$ is the corrupted weight value, and α is a random number in the range [-k, k].

The comparative performance of the PSHNN and the 4NN was studied as in the case of additive Gaussian noise. The results with the 4NN are shown in Table 21. If k is less than .5, the performance of the noisy system is similar to the performance of the error-free system. When k is greater than 1, the performance of the noisy system rapidly deteriorates. Table 22 shows the

Table 20. The classification accuracy of the PSHNN2 and the total number of SNNs as a function of the number of iterations in the 4-class problem ($\eta = 0.05$). Weights are corrupted by additive Gaussian noise. The maximum number of iterations allowed is 90 (data set I).

SNR (db)	Number of SNNs	Classification Accuracy	
		Train (%)	Test (%)
no errors	2	98.75	92.13
20.0	2	98.87	91.80
10.0	2	98.50	92.06
3.01	2	98.37	92.33
0.00	2	97.25	90.23
-3.01	2	97.87	90.20
-4.77	3	97.75	91.00
-6.02	4	97.87	89.06
-6.99	3	97.62	88.00
-7.78	3	98.00	88.67
-8.45	3	97.87	90.04
-9.03	4	98.00	88.20
-9.54	4	96.75	87.60
-10.0	5	95.00	84.80
-20.0	11	81.25	62.80

performance of the PSHNN. If k is less than or equal to 2, the performance of the noisy system is almost the same as the performance of the error-free system. When k is greater than 2, the system still performs quite well, with accuracy in the range of 80 %. The deterioration of the system performance is very slow with increasing k. Even when k increases towards 10, the testing accuracy remains around 60 - 70 %. However, as k increases, so does the number of stages.

The results with both additive and multiplicative noise for the corruption of the weight values indicate that the PSHNN is extremely fault-tolerant to errors in the weight values. This is due to the adaptation of the bounds at the end of each stage.

The results should also be valid when the input-output signal statistics change in time such that the weights are less optimal than before. The system

Table 21. The classification accuracy of the 4NN as a function of the number of iterations in the 4-class problem ($\eta = 0.01$). The number of iterations is 1000. Weights are corrupted by multiplicative noise (data set I).

Error k	Classification Accuracy	
	Train (%)	Test (%)
no errors	99.88	92.60
0.1	99.88	92.60
0.25	99.38	92.53
0.5	97.87	90.46
1.0	93.00	84.93
2.0	60.37	60.00
3.0	36.50	40.20

Table 22. The classification accuracy of the PSHNN2 and the total number of SNNs as a function of the number of iterations in the 4-class problem ($\eta = 0.05$). The maximum number of iterations allowed is 90. Weights are corrupted by multiplicative noise (data set I).

Error k	Number of SNNs	Classification Accuracy	
		Train (%)	Test (%)
no errors	2	98.75	92.13
0.25	2	98.25	91.40
0.5	2	98.75	91.73
1.0	3	98.25	89.93
2.0	3	97.37	91.13
3.0	4	96.50	88.13
4.0	5	91.50	85.73
5.0	7	91.00	81.13
6.0	7	86.50	71.13
10.0	10	72.87	61.53

performance can be kept high by real-time adaptation of the bounds. The adaptation of the weights can be done with a longer time constant, if necessary.

6. CONCLUSIONS

The PSHNN has many attractive properties. Its most unique property is error detection at the end of each SNN. This makes possible the avoidance of backpropagation of errors from stage to stage to learn the weights, the avoidance of the requirement for differentiable and invertible nonlinearities, faster learning time since fewer training vectors are utilized in later stages, parallel operation of SNN's during testing, real time adaptation to nonoptimal connection weights by adjusting the error detection bounds, and thereby achieving very high fault-tolerance and robustness. In this paper, each SNN was implemented with the delta rule. It can also be implemented by any other learning rule. This property increases the flexibility of the PSHNN architecture.

The PSHNN is self-organizing in the sense of learning the number of SNN's needed during training. Easy problems often require 2 stages. With more difficult problems, the number of stages increase. This is a very useful property since it allows adaptation to any type of problem by the network itself, without interference from other sources.

Better algorithms for the detection of errors should improve the accuracy performance of the PSHNN further.

ACKNOWLEDGMENT

We thank Jon A. Benediktsson and Prof. Phil H. Swain of the Laboratory for Applications of Remote Sensing, Purdue University for providing us with the remote sensing data used in experiments.

REFERENCES

1. M. Takeda and J.W. Goodman, "Neural Networks for Computation : Number representations and Programming Complexity," *Applied Optics,* Vol. 25, No. 18, September, 1986.

2. L.W. Couch II, *Digital and Analog Communication Systems,* Mac Millan Pub. Co., 1983.

3. O.K. Ersoy, "Real Discrete Fourier Transform", *IEEE Tran. ASSP, ASSP-33,* No. 4, p.880-882, Aug, 1985.

4. O.K. Ersoy, N.C. Hu, "Fast Algorithms for the Real Discrete Fourier Transform", *Proceedings of ICASSP 88,* p. 1902-1905, April, 1988.

5. G. Widrow, M.E. Hoff, "Adaptive Switching Circuits", *Inst. Radio Engineers Western Electronic Show and Convention Record,* Part 4, p. 96-104.

6. O.K. Ersoy, "A New Family of Discrete Orthogonal Transforms and Their Application," *Tenth Symposium on Signal Processing and Application,* Nice, France, May 1985.

7. O.K. Ersoy, N-C Hu, "The Discrete Fourier Processing Transforms and their Fats Algorithms," submitted to *IEEE Tran. Acoustics, Speech, Signal Processing and ICASSP 1990.*

8. D. E. Rumelhart, J. L. McClelland, PDP Research Group, *Parallel Distributed Processing,* The MIT Press, Cambridge Massachusetts, 1988.

9. H. Ghassemian, D. Landgrebe, "On-Line Object Feature Extraction for Multispectral Scene Representation," *Technical Report No. TR-EE 88-34,* Purdue University, August 1988.

10. 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.*