#### Purdue University Purdue e-Pubs

**ECE** Technical Reports

**Electrical and Computer Engineering** 

1-1-1992

### Hierarchical Neural Networks with Forward-Backward Training

S-W. Deng Purdue University School of Electrical Engineering

O. K. Ersoy Purdue University School of Electrical Engineering

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

Deng, S-W. and Ersoy, O. K., "Hierarchical Neural Networks with Forward-Backward Training" (1992). *ECE Technical Reports*. Paper 279. http://docs.lib.purdue.edu/ecetr/279

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.



# Parallel, Self-Organizing, Hierarchical Neural Networks with Forward-Backward Training

S-W. Deng O. K. Ersoy

TR-EE 92-3 January 1992

### PARALLEL, SELF-ORGANIZING, HIERARCHICAL NEURAL NETWORKS WITH FORWARD-BACKWARD TRAINING

S-W Deng, O. K. Ersoy

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

#### ABSTRACT

A forward-backward training algorithm for parallel, self-organizing hierachical neural networks (**PSHNN's**) is described. Using linear algebra, it is shown that the forward-backward training of an n-stage PSHNN until convergence is equivalent to the pseudo-inverse solution for a single, total network designed in the least-squares sense with the total input vector consisting of the actual input vector and its additional nonlinear transformations. These results are **also** valid when a single long input vector is partitioned into smaller length vectors. A number of advantages achieved are small modules for easy and fast learning, parallel implementation of small modules during testing, faster convergence rate, better numerical error-reduction, and suitability for learning input nonlinear transformations by other neural networks. The backpropagation (BP) algorithm is proposed for learning input nonlinearities. Better performance in terms of deeper minimum of the error function and faster convergence rate is achieved when a single BP network is replaced by a PSHNN of equal complexity in which each stage is a BP network of smaller complexity than the single BP network.

#### **1. INTRODUCTION**

Parallel, self-organizing, hierarchical neural networks (PSHNN's) are **multistage** networks in which stages operate in parallel rather than in series during testing [1], [2]. The PSHNN is self-organizing in the sense of number of stages. Each stage is a particular neural network, referred to as the stage neural **network(SNN)**. The PSHNN's **as** discussed in [1] and [2] assume quantized or continuous-valued inputs and quantized, say, binary outputs. At the output of each SNN, there is an error detection scheme which allows acceptance or rejection of input vectors. If an input vector is rejected, it goes through a nonlinear **transformation (NLT) befor** being inputted to the next stage. Only those input **vectors** which are rejected by the present stage are fed into the next stage after the nonlinear transformation.

In a recent paper, we discussed the generalization of parallel, self-organizing, hierarchical neural networks (PSHNN's) to continuous inputs as well **as** continuous outputs [**3**]. The block diagram for such a 3-stage PSHNN is shown in Fig. 1. It was shown that stages are generated by nonlinearly transforming input vectors, and each new stage attempts to correct the errors of the previous stage. It was also discussed that further error reduction in an n-stage network is possible by circularly transmitting the remaining error through the stages a number of times until convergence. Running through all the stages once can be called one sweep. At each successive sweep, the desired output of each stage is modified **as** the previous output of the stage plus the remaining error from the previous stage. The first stage receives the error from the last stage. Both in Ref. [**3**] and in this paper, the output nodes are assumed to be linear.

In this paper, forward-backward training of n-stage **PSHNN's** are introduced and discussed on a rigorous mathematical basis, in addition to providing experimental results. The results are actually valid for all linear **least-squares** problems if we consider the input vector and vectors generated from it by nonlinear transformations as the decomposition of a single, long vector. In this sense, the techniques discussed represent the decomposition of a large problem into smaller problems which are related through wrrors and forward-backward training. Generation of additional nodes at the input is common to a number of techniques such **as** generalized discriminant functions [4], higher order networks [5], and function-link networks [6]. After this is done, a single total network can be trained by the delta rule [7]. At convergence, the result is approximately the same **as** the **pseudo-inverse** solution, disregarding any possible numerical problems [8]. The PSHNN's are different because the single total network are rplaced by a number of subnetworks.

The main result in this paper is that forward-backward training of an **n**-stage network until convergence is equivalent to the **pseudo-inverse** solution for a

single total network with the total number of input nodes if each stage is optimized in the sense of least-squares. There are a number of advantages in achieving the **pseudo-inverse** solution in this fashion. The most obvious advantage is that-each stage is much easier to implement as a module to be trained than the whole network. In addition, all stages can be processed in parallel during testing. If the complexity of implementation without parallel stages is denoted by f(N) where N is the length of input vectors, the parallel complexity of the forward-backward training algorithm during testing is f(K) where K equals N/M with M equal to the number of stages.

The paper consists of eight sections. In Sec. 2, the forward-backward training algorithm is described in detail. In Sec. 3, the asymptotic properties with a two-stage network are discussed. These properties are extended to n-stage networks in Sec. 4. The suboptimal asymptotic properties due to the use of the delta rule during training are proved in Sec. 5. Experimental results are provided in Sec. 6. Another advantage of PSHNN is that input nonlinear transformations (NLT's) can be learned. In Sec. 7, we illustrate a technique which uses backpropagation (BP) algorithm with forward-backward training to learn input nonlinear transformations. Simulation results of this section indicate that the total network consisting of small BP stages usually converges faster and to a deeper minimum of the error function than a single BP network of the same total siee. Conclusions are given in Sec. 8.

#### 2. PSHNN WITH FORWARD-BACKWARD TRAINING

The system model is shown in Fig.1. In this section, a single output is assumed. In Fig.1, SNN(i) represents the i-th stage neural network. In this paper, the stage neural network is assumed to be trained by the delta rule [9]. The output nodes are assumed to be linear. X(n) is the input vector sequence; d(n) is the desired output sequence; X'(n), Y(n) and Z(n) are obtained by different nonlinear transformations NLT1, NLT2 and NLT3.

We first consider a two-stage PSHNN, and then generalize the properties to n stages. Assuming m training vectors of length p and NLT1 in Fig. 1 to be the identity operator (X(n)=X'(n)), we define

$$X = \begin{bmatrix} x_1^t \\ x_2^t \\ \vdots \\ \vdots \\ x_m^t \end{bmatrix},$$
$$Y = \begin{bmatrix} y_1^t \\ y_2^t \\ \vdots \\ \vdots \\ y_m^t \end{bmatrix},$$

$$D_1^1 = \begin{bmatrix} d(1) & d(2) & \cdots & d(m) \end{bmatrix}^t,$$
$$W_1 = \begin{bmatrix} a_1 & a_2 & \cdots & a_p \end{bmatrix}^t,$$
$$W_2 = \begin{bmatrix} b_1 & b_2 & \cdots & b_p \end{bmatrix}^t.$$

X and Y are m  $\times$  p matrices. Each row of X or Y represents an input vector of **SNN1** or **SNN2**, respectively.  $D_1^1$  is the desired output vector of length m. Using the delta rule to train SNN1 corresponds ideally to finding the least-squares solution for  $XW_1 = D_1^1$ . The output of SNN1 is  $o_1^1$  which can be expressed as [10]

$$o_1^1 = XX^+ D_1^1 = AD_1^1, \tag{1}$$

where  $X^+$  is the generalized inverse of X, and the projection operator A is  $XX^+$ , which is positive semidefinite [4].

The error vector of SNN1 is

$$e_1^1 = D_1^1 - o_1^1 = (I - A)D_1^1.$$
(2)

We use  $e_1^1$  as the desired output for SNN2, to be also trained by the delta rule. The output of SNN2 after training can be expressed as

$$o_2^1 = YY^+ e_1^1 = Be_1^1, (3)$$

where we define  $YY^+ \triangleq B$ , which is also positive and semidefinite. Then,

$$e_2^1 = e_1^1 - o_2^1 = (I - B)e_1^1.$$
(4)

With two stages,  $o_1^1 + o_2^1$  is the output, and the system error  $e_f$  is

$$e_f = D_1^1 - (o_1^1 + o_2^1) = e_2^1.$$
(5)

The above results can be considered to be the first sweep in a number of sweeps of forward-backward training. In the second sweep, the desired vector for SNN1 is set equal to

$$D_1^2 = o_1^1 + e_2^1. (6)$$

The new output of SNN1 is

$$o_1^2 = A(o_1^1 + e_2^1) = o_1^1 + Ae_2^1, \tag{7}$$

because A is the projection operator,  $o_1^1$  is in the space spanned by A, and  $Ao_1^1 = o_1^1$ .

The new error signal for SNN1 is

$$e_1^2 = D_1^2 - o_1^2 = (I - A)e_2^1.$$
(8)

After a straightforward derivation, we get

$$e_1^2 = D_1^1 - (o_1^2 + o_2^1). \tag{9}$$

If we terminate the training at this point, the system output is  $o_1^2 + o_2^1$ . Therefore  $e_1^2$  is just the error of the system. If we continue to train SNN2, the new desired signal for SNN2 is

$$D_2^2 = o_2^1 + e_1^2. \tag{10}$$

The output of SNN2 becomes

$$o_2^2 = BD_2^2 = o_2^1 + Be_1^2, \tag{11}$$

since  $o_2^1$  is in the space spanned by B. The error vector for **SNN2**, is

$$e_2^2 = D_2^2 - o_2^2 = (I - B)e_1^2.$$
(12)

Using the same derivation leading to Eq.(9), we get

$$e_2^2 = D_1^1 - (o_1^2 + o_2^2), \tag{13}$$

where  $e_2^2$  is the error signal of the system at the end of the second sweep.

At the n-th sweep, the desired output signal for SNN1 is

$$D_1^n = o_1^{n-1} + e_2^{n-1}.$$
(14)

After training, the output of SNN1 is

$$o_1^n = AD_1^n = o_1^{n-1} + Ae_2^{n-1}.$$
 (15)

The error vector is

$$e_1^n = D_1^n - o_1^n = (I - A)e_2^{n-1}.$$
 (16)

The error vector can also be written as

$$e_1^n = D_1^1 - (o_1^n + o_2^{n-1}).$$
(17)

At the n-th sweep, the desired signal for SNN2 is

$$D_{n}^{n} \equiv o_{n}^{n-1} + e_{1}^{n}. \tag{18}$$

The output is

$$o_2^n = BD_2^n = o_2^{n-1} + Be_1^n.$$
(19)

The error is

$$e_2^n = D_2^n - o_2^n \equiv (I-B)e_1^n. \tag{20}$$

Again, we note that

$$e_2^n = D_1^1 - (o_1^n + o_2^n), \tag{21}$$

where  $e_2^n$  is the system error after the nth sweep.

From Eq.(2) and Eq.(4), we get

$$||e_1^1||^2 = (D_1^1)^t (I-A)(D_1^1), \qquad (22)$$

$$||e_{2}^{1}||^{2} = (e_{1}^{1})^{t} (I-B)(e_{1}^{1}) \leq ||e_{1}^{1}||^{2}.$$
(23)

$$||e_1^2||^2 = (e_2^1)^t (I-A)(e_2^1) \le ||e_2^1||^2, \qquad (24)$$

$$||e_{2}^{2}||^{2} = (e_{2}^{1})^{t} (I-B)(e_{2}^{1}) \leq ||e_{2}^{1}||^{2}.$$
(25)

From Eq.(16) and Eq.(20). we conclude that

$$||e_1^n||^2 = (e_2^{n-1})^t (I-A)(e_2^{n-1}) \le ||e_2^{n-1}||^2,$$
(26)

$$||e_{2}^{n}||^{2} = (e_{1}^{n})^{t}(I-B)(e_{1}^{n}) \leq ||e_{1}^{n}||^{2}.$$
(27)

Therefore,

$$||e_{2}^{n}||^{2} \leq ||e_{1}^{n}||^{2} \leq ||e_{2}^{n-1}||^{2} \leq \cdots \leq ||e_{2}^{1}||^{2} \leq ||e_{1}^{2}||^{2} \leq ||e_{1}^{1}||^{2}.$$
(28)

We will see in the next section that

$$\lim_{n \to \infty} ||e_1^n||^2 = ||e||^2,$$
(29)

$$\lim_{n \to \infty} ||e_2^n||^2 = ||e||^2 , \qquad (30)$$

where  $||e||^2$  is the square error sum of the function-link network which has the same input NLT's as used in the PSHNN.

# **3.** ASYMPTOTIC PROPERTIES OF A TWO-STAGE PSHNN WITH FORWARD-BACKWARD TRAINING

Consider a function-link network as shown in Fig. 2. Let X denote an input vector, Y be a nonlinear transformation of X and D be the desired output vector. X and Y are  $m \times n$  matrices, D is an  $m \times 1$  vector, and W is a  $2n \times 1$  weight matrix.

Using the delta rule to train W corresponds approximately to finding the least square solution for

$$(X, Y)W=D,$$

where (X, Y) denotes the concatenation of X and Y. The least square solution is

$$\overline{W} = (X, Y)^+ D,$$

where  $(X, Y)^+$  is the pseudo-inverse of (X, Y). The output vector is

$$o = (X, Y)(X, Y)^+ D,$$

Therefore, the error vector is

$$e = D - o = (I - (X, Y)(X, Y)^+)D.$$
 (31)

If we use PSHNN with forward-backward training, Eqs. (2), (4), (8), (13) and  $D_1^1 = D$  in this case lead to

$$e_1^1 = (I - XX^+)D,$$
 (32)

$$e_2^1 = (I - YY^+)(I - XX^+)D, \tag{33}$$

$$e_1^2 = (I - XX^+)[(I - YY^+)(I - XX^+)]D, \qquad (34)$$

$$e_2^2 = [(I - YY^+)(I - XX^+)]^2 D, \qquad (35)$$

$$e_1^n = (I - XX^+)[(I - YY^+)(I - XX^+)]^{n-1}D, \qquad (36)$$

$$e_2^n = [(I - YY^+)(I - XX^+)]^n D, (37)$$

$$e_1^{n+1} = (I - XX^+)[(I - YY^+)(I - XX^+)]^n D.$$
(38)

We will need the following properties to prove the main theorem of this section:

**Property 1** The null space  $N(XX^t + YY^t)$  is equivalent to the intersection of the null space  $N(XX^t)$  and the null space  $N(YY^t)$ .

Proof:

(i) For any vector  $y \in N(XX^t) \cap N(YY^t)$ it is obvious that  $y \in N(XX^t + YY^t)$ . (ii) For any vector  $y \in N(XX^t + YY')$  $(XX^t + YY^t)y=0$  $\implies XX^ty=-YY^ty$ Therefore,  $y^tXX^ty=-y^tYY'y$ Since  $XX^t$  and YY' are positive semidefinite  $y \in N(XX^t)$  and  $y \in N(YY^t)$ 

In addition, the following properties are needed:

**Property 2** The projection operators  $P_{N(XX')}$  and  $P_{N(YY')}$  satisfy  $\lim_{n \to \infty} (P_{N(XX')}P_{N(YY')})^n = P_{N(XX')} \cap^{N(YY')},$ (39)

which can be found in Nakano [11]. This property tells us that the projection not in the intersection of  $N(XX^t)$  and  $N(YY^t)$  will gradually vanish as **n** goes to infinity. The projection in the intersection of  $N(XX^t)$  and  $N(YY^t)$  will be preserved.

#### **Property 8**

$$P_{N(XX')}P_{N(XX')} \cap N(YY') = P_{N(XX')} \cap N(YY'), \qquad (40)$$

which can be found in Hartwing and Drazin [12] and Nakano [11].

Next, we will state and prove the main theorem:

**Theorem 1** 

$$\lim_{n \to \infty} e_1^{n+1} = \lim_{n \to \infty} e_1^n = e, \qquad (41)$$

$$\lim_{n \to \infty} e_2^n = e. \tag{42}$$

Proof:

The projection matrices are

$$(I-XX^+) \stackrel{\Delta}{=} P_{N(XX^t)},$$
  
$$(I-YY^+) \stackrel{\Delta}{=} P_{N(YY^t)}.$$

Comparing Eqs. (31), (37) and (38), sufficient conditions for Eq. (41) and Eq. (42) to hold are

$$\lim_{n \to \infty} (I - XX^+) [(I - YY^+)(I - XX^+)]^n = [I - (X, Y)(X, Y)^+],$$
(43)

$$\lim_{n \to tw} [(I - YY^+)(I - XX^+)]^n = [I - (X, Y)(X, Y)^+].$$
(44)

Using the projection operators, we get

$$[(I-YY^{+})(I-XX^{+})]^{n} = (P_{N(YY')}P_{N(XX')})^{n}.$$
(45)

From Property 1, we have

$$N(XX^t) \cap N(YY^t) = N(XX^t + YY^t) = N((X, Y)(X, Y)^t).$$

Therefore,

$$P_{N(XX^{i}) \cap N(YY^{i})} = P_{N((X,Y)(X,Y)^{i})}.$$
(46)

We know that

$$P_{N((X,Y)(X,Y)^{i})} = [I - (X,Y)(X,Y)^{+}]$$
(47)

From Eqs. (39), (45), (46) and (47), we conclude that

$$\lim_{n \to \infty} [(I - YY^+)(I - XX^+)]^n = [I - (X, Y)(X, Y)^+].$$

Eq. (44) to be proved follows directly from Property 3:

$$\lim_{n \to \infty} (I - XX^+) [(I - YY^+)(I - XX^+)]^n = [I - (X, Y)(X, Y)^+] \square$$

The theorem proved above means that, as **n** grows larger, the error vectors  $e_1^n$  and  $e_2^n$  approach the error vector e for the pseudoinverse solution if a single total network was built without stages with the total input vector.

#### 4. ASYMPTOTIC PROPERTIES FOR AN N-STAGE NETWORK

When the number of stages is 2, forward-backward training is the same as circular training discussed in Ref. [10]. In the circular training algorithm with **n** stages, after training SNN(n), we train SNN(1). In forward-backward training, we will train SNN(n-1) after training SNN(n), followed by SNN(n-2) and so on. From the first stage to the last stage, we have a forward path training, and then from the last stage to the first stage, we have a backward path training. One sweep training consists of a forward path and a backward path training. We will call this training procedure the forward-backward training algorithm.

For the sake of brevity, we will discuss the 3-stage PSHNN. All the properties of the 3-stage network can be derived for the n-stage network in the same way. Referring to Fig.1 and supposing X=X', we define  $N[XX^t]=A$ ,  $N[YY^t]=B$ , and  $N[ZZ^t]=C$  to represent the null space of (xx'), (YY') and  $(ZZ^t)$ , respectively. After the first stage is trained, the error vector is

$$e_{1f}^{1} = [P_A]D, \tag{48}$$

where  $P_A$  is the projection matrix of A, and D is the desired output vector. The superscript of the error vector denotes the number of sweeps, the arabic number on the subscript denotes the number of stages, and the letter "f" on the subscript means forward path training. Following the same procedure as in Section 3, we have

$$e_{2f}^{1} = [P_{B}P_{A}]D, \tag{49}$$

$$e_{3f}^{1} = [P_C P_B P_A] D.$$
<sup>(50)</sup>

After training three stages in the forward path, we transmit the error of the third stage to the second stage and modify the desired output of the second stage in order to train the second stage, and get the error vector

$$e_{2b}^1 = [P_B P_C P_B P_A] D, \tag{51}$$

where the letter "b" in the subscript means backward training path. After training the second stage, we train the first stage and get the error vector

$$e_{1b}^1 = [P_A P_B P_C P_B P_A] D.$$
<sup>(52)</sup>

Now, the first sweep is over, and the second sweep starts.

Following the same procedure as above, we get the following error vectors in the second sweep:

$$e_{1f}^{2} = P_{A} [P_{A} P_{B} P_{C} P_{B} P_{A}] D$$
$$= [P_{A} P_{B} P_{A} P_{B} P_{A}] D$$
$$= e_{1b}^{1}, \qquad (53)$$

$$e_{2f}^{2} \equiv P_{B} P_{A} [P_{A} P_{B} P_{C} P_{B} P_{A}]^{D},$$

$$(54)$$

$$e_{3f}^2 = P_C P_B P_A [P_A P_B P_C P_B P_A] D, \tag{55}$$

$$e_{2b}^2 = P_B P_C P_B P_A [P_A P_B P_C P_B P_A] D, \tag{56}$$

$$e_{1b}^2 = [P_A P_B P_C P_B P_A]^2 D$$
$$= e_{1f}^3.$$
(57)

After the nth sweep training, the error vector of the first stage becomes

$$e_{1b}^{n} = e_{1f}^{n+1} = [P_A P_B P_C P_B P_A]^{n} D.$$
(58)

Similar to the derivation of Eq.(31), the error vector for a 3-stage functionlink network is

$$e = [I - (X, Y, Z)(X, Y, Z)^{+}]D$$
  
=  $[P_{N(XX^{i} + YY^{i} + ZZ^{i})}]D,$  (59)

where  $N(XX^{t}+YY^{t}+ZZ^{t})$  denotes the null apace of  $(XX^{t}+YY^{t}+ZZ^{t})$ .

We also need the following properties:

**Property 1.a** The null space  $N(XX^{t}+YY^{t}+ZZ^{t})$  is equivalent to the intersection of the null space  $N(XX^{t})$ , the null space  $N(YY^{t})$  and the null space  $N(ZZ^{t})$ .

Proof:

(i) For any vector  $a \in N(XX^t) \cap N(YY^t) \cap N(ZZ^t)$ , it is obvious that  $a \in N(XX^t + YY^t + ZZ^t)$ . (ii) For any vector  $a \in N(XX^t + YY^t + ZZ^t)$ , then  $(XX^t + YY^t + ZZ^t)a = 0$ . Therefore,  $a^t(XX^t + YY^t + ZZ^t)a = 0$ ,  $= > a^tXX^ta + a^tYY^ta + aZZ^ta = 0$ . Because  $(XX^t)$ ,  $(YY^t)$ , and  $(ZZ^t)$  are positive semidefinite, we have  $a^tXX^ta = 0$ ,  $a^tYY^ta = 0$  and  $a^tZZ^ta = 0$ . These imply  $a \in N(XX^t)$ ,  $a \in N(YY^t)$ , and  $a \in N(ZZ^t)$ .

Property 2.a

$$\lim_{n \to \infty} (P_A P_B P_C P_B P_A)^n = P_A \bigcap^B \bigcap^C$$
(60)

which was proved by Pyle [13].

From Eq.(59) and property 1.a, we get

$$e = (P_{N(XX^{i}+YY^{i}+ZZ^{i})})D = (P_{A \cap B \cap C})D.$$
(61)

By using Property 2.a, Eq.(58) and Eq.(61), we obtain the main theorem of this section:

#### **Theorem 2**

$$\lim_{n \to \infty} e_{1b}^n = e.$$
 (62)

Since Property 2.a still holds for the intersection of **n** projection matrices, the generalization of Theorem 2 to the n-stage PSHNN with forward-backward training is obvious.

The results of Theorem 1 of **Sec.** 3 is based on the **two-stage** PSHNN. For the two-stage PSHNN, circular training is the same as the forward-backward training. An interesting question is whether circular training gives the same results **as** forward-backward training for the n-stage networks. This is conjectured to be true since many experiments show that [13]

$$\lim_{n \to \infty} (P_C P_B P_A)^n = P_A \bigcap^B \bigcap^C C.$$
(63)

Experimentally, we have also observed that circular training gives the same results **as** forward-backward training.

### 6. ASYMPTOTIC PROPERTIES FOR THE SUBOPTIMAL SOLUTIONS

In Sec. 4, we discussed the asympttic property of PSHNN with forwardbackward training when each stage gives the exact least-squares solution. In this section, we generalize the asympttic property to the suboptimal leastsquares solution due to the use of the delta rule. We discuss the case of the two-stage PSHNN, and the results can be easily extended to the n-stage PSHNN.

Assuming a two-stage network, the square error sum  $||e_2^1||^2$  in Eq. (23) is based on the optimal least-squares solution for the second stage. The leastsquares error vector  $e_2^1$  is in the null space of [YY']. Defining  $\xi_{ls} \triangleq ||e_2^1||^2$ , Eq. (23) can be written as

$$\xi_{ls} = ||(I - YY^{\dagger})e_1^1||^2 = ||P_{N(YY^{\dagger})}e_1^1||^2, \qquad (64)$$

where  $P_{N(YY')}$  is the projection matrix to the null space of  $YY^t$ .

In reality, the square error sum we get by using the delta rule is based on a suboptimal leastsquares solution. The suboptimal square error sum denoted as  $\xi_{ls}$  can be expressed as [14]

$$\hat{\xi}_{ls} = m(\xi_{\min} + \xi_{exc}), \tag{65}$$

where m denotes the number of input vectors.  $\xi_{min}$  is the minimum mean square error (MSE) by solving the normal equation

$$E[Y_{N}(n)Y_{N}(n)^{t}]W_{N}=E[e_{1}^{1}(n)Y_{N}(n)], \qquad (66)$$

where  $Y_N(n) = [y(n), y(n-1), \dots, y(n-N+1)]^t$ , and N denotes the number of weights of SNN2 of Fig.1;  $\xi_{exc}$  is due to the actual LMS weights jitter, and is sometimes referred to as the excess MSE. If we assume the sequence y(n) is stationary and ergodic, then  $m\xi_{\min}$  in Eq. (65) gradually approaches the optimal square error sum  $\xi_{ls}$  as m grows. Thus, approximating  $m\xi_{\min}$  by  $\xi_{ls}$ , Eq. (65) can be written as

$$\hat{\xi}_{ls} = \xi_{ls} + m \xi_{ezc} \tag{67}$$

 $\xi_{ezc}$  is proportional to gain  $\eta$  used in training. Choosing smaller  $\eta$  achieves better suboptimal square error sum  $\xi_{ls}$ , but then the learning rate is slower. So, there is a trade-off involved in choosing the value of  $\eta$ .

We show below that the error reduction properties derived in Sec. 2 still hold in practise with the square error sum  $\hat{\xi}_{ls}$  based on a suboptimal leastsquares solution.

For the sake of brevity, we consider a two-stage PSHNN with **NLT1** being the identity operator.  $D_1^1$  is the desired vector for the first stage network in the first sweep. The output vector of the first stage based on the optimal leastsquares solution is

$$o_1^1 = P_{col[XX^i]} D_1^1 . (68)$$

The output vector  $\hat{o}_1^1$  based on the suboptimal least-squares solutions  $W'_1$  is written as

$$\hat{o}_1^1 = X W_1'$$
 (69)

This shows that  $\hat{o}_1^1 \in col[XX^t]$ .  $\hat{o}_1^1$  can be written as

$$\hat{o}_{1}^{1} = P_{col[XX']} D_{1}^{1} + b_{1}^{1} , \qquad (70)$$

where the vector  $b_1^1$  also belongs to the column space of  $[XX^t]$ . This is graphically shown in Fig. 3. The magnitude of  $b_1^1$  can be written as

$$[|b_{1}^{1}||=c_{1}^{1}||P_{col[XX']}D_{1}^{1}|], \qquad (71)$$

where  $c_1^1$  satisfies  $0 < c_1^1 < 1$  in practise. Thus the error vector of SNN1 in the first sweep is

$$\hat{e}_{1}^{1} = P_{N[XX']} D_{1}^{1} - b_{1}^{1} . \qquad (72)$$

 $\hat{e}_1^1$  is also the desired vector for the second stage network in the first sweep. Referring to Fig. 4, and using the same procedure as above, we get the suboptimal output vector  $\hat{o}_2^1$  of SNN2 in the first sweep as

$$\hat{o}_{2}^{1} = P_{col[YY']} \hat{e}_{1}^{1} + b_{2}^{1} , \qquad (73)$$

where the vector  $b_2^1$  belongs to the column space of  $[YY^t]$ , and the magnitude of  $b_2^1$  is

$$||b_{2}^{1}|| = c_{2}^{1} ||P_{col[YY']} \hat{e}_{1}^{1}||, \qquad (74)$$

where  $c_2^1$  also satisfies  $0 < c_2^1 < 1$  in practise. The error vector of SNN2 in the first sweep is

$$\hat{e}_{2}^{1} = P_{N[YY']} \hat{e}_{1}^{1} - b_{2}^{1} \quad . \tag{75}$$

Since  $P_{N[YY']} \hat{\epsilon}_1^1$  and  $b_2^1$  are orthogonal to each other, we get

$$||\hat{e}_{2}^{1}||^{2} = ||P_{N[YY']}\hat{e}_{1}^{1}||^{2} + ||b_{2}^{1}||^{2} \\ \leq ||P_{N[YY']}\hat{e}_{1}^{1}||^{2} + ||P_{col[YY']}\hat{e}_{1}^{1}||^{2} = ||\hat{e}_{1}^{1}||^{2} .$$
(76)

Thus,  $\|\hat{e}_{2}^{1}\|^{2}$  is less than  $\|\hat{e}_{1}^{1}\|^{2}$  as long as  $c_{2}^{1}$  is less than 1, which is definitely ture in practise.

On the second sweep, the desired vector of SNN1 is  $\hat{e}_2^1 + \hat{o}_1^1$ . Following the **same** procedure as above, the suboptimal output vector  $\hat{o}_1^2$  of SNN1 in the second **sweep** is found as

$$\hat{o}_{1}^{2} = P_{col[XX']}(\hat{e}_{2}^{1} + \hat{o}_{1}^{1}) + b_{1}^{2}$$
  
=  $\hat{o}_{1}^{1} + P_{col[XX']}\hat{e}_{2}^{1} + b_{1}^{2}$ , (77)

and

$$||b_{1}^{2}|| = c_{1}^{2} ||P_{col|[XX']}(\hat{e}_{1}^{2} + \hat{o}_{1}^{1})||, \qquad (78)$$

where  $\hat{o}_1^1 \in col[XX^t]$ ,  $\hat{b}_1^2 \in col[XX^t]$  and  $0 < c_1^2 < 1$ . The error vector  $\hat{e}_1^2$  of SNN1 in the second sweep is

$$\hat{e}_{1}^{2} = (\hat{e}_{2}^{1} + \hat{o}_{1}^{1}) - \hat{o}_{1}^{2}$$

$$= P_{N[XX']} \hat{e}_{2}^{1} - b_{1}^{2} .$$
(79)

The desired vector of SNN2 in the second sweep is  $\hat{e}_1^2 + \hat{o}_2^1$ . The suboptimal output vector  $\hat{o}_2^2$  of SNN2 in the second sweep is

$$\hat{o}_{2}^{2} = P_{col[YY^{i}]}(\hat{e}_{1}^{2} + \hat{o}_{2}^{1}) + b_{2}^{2}$$
$$= \hat{o}_{2}^{1} + P_{col[YY^{i}]}\hat{e}_{1}^{2} + b_{2}^{2} , \qquad (80)$$

and

$$||b_{2}^{2}|| = c_{2}^{2} ||P_{col[YY']}(\hat{e}_{1}^{2} + \hat{o}_{2}^{1})||, \qquad (81)$$

where  $\hat{o}_2^1 \in col[YY^t]$ ,  $b_2^2 \in col[YY^t]$ , and  $0 < c_2^2 < l$ . The error vector  $\hat{e}_2^2$  of SNN2 in the second sweep is

$$\hat{e}_{2}^{2} = (\hat{e}_{1}^{2} + \hat{o}_{2}^{1}) - \hat{o}_{2}^{2}$$
$$= P_{N[YY']} \hat{e}_{1}^{2} - b_{2}^{2} . \qquad (82)$$

Using Eq. (72) and Eq. (75), and letting  $A \stackrel{\Delta}{=} N[XX^t], B \stackrel{\Delta}{=} N[YY^t]$ ; the suboptimal error vector  $\hat{\epsilon}_1^1$  of the first stage in the first sweep becomes

$$\hat{e}_1^1 = P_A D_1^1 - b_1^1 \quad . \tag{83}$$

The suboptimal error vector  $\hat{\boldsymbol{\epsilon}}_{2}^{1}$  of the second stage in the first sweep becomes

$$\hat{e}_{2}^{1} = P_{B}\hat{e}_{1}^{1} - b_{1}^{2}$$
$$= P_{B}P_{A}D_{1}^{1} - P_{B}b_{1}^{1} - b_{2}^{1} . \qquad (84)$$

Using Eq. (79) and Eq. (84), the suboptimal error vector  $\hat{e}_1^2$  of the first stage in the second sweep becomes

$$\hat{e}_{1}^{2} = (P_{A}P_{B})P_{A}D_{1}^{1} - P_{A}P_{B}b_{1}^{1} - P_{A}b_{2}^{1} - b_{1}^{2} , \qquad (85)$$

where  $b_1^2 \in col[XX^t]$ . The suboptimal error vector  $\hat{e}_2^2$  of the second stage in the second sweep becomes

$$\hat{\boldsymbol{\varepsilon}}_{2}^{2} = (P_{B}P_{A})^{2} D_{1}^{1} - (P_{B}P_{A}) P_{B} b_{1}^{1} - (P_{B}P_{A}) b_{2}^{1} - P_{B} b_{1}^{2} - b_{2}^{2} , \qquad (86)$$

where  $b_2^2 \in col[YY^t]$ .

Following the same procedure, the suboptimal error vector  $\hat{\epsilon}_1^n$  of the first stage in the n-th sweep becomes

$$\hat{e}_1^n = (P_A P_B)^{n-1} P_A D_1^1 - \sum_{k=1}^n (P_A P_B)^{n-k} b_1^k - \sum_{k=1}^{n-1} (P_A P_B)^{n-1-k} P_A b_2^k .$$
(87)

The suboptimal error vector  $\hat{e}_2^n$  of the second stage in the n-th sweep becomes

$$\hat{e}_{2}^{n} = (P_{B}P_{A})^{n} D_{1}^{1} - \sum_{k=1}^{n} (P_{B}P_{A})^{n-k} P_{B} b_{1}^{k} - \sum_{k=1}^{n} (P_{B}P_{A})^{n-k} b_{2}^{k} , \qquad (88)$$

where  $b_1^i \in col[XX^t]$ , and  $b_2^i \in col[YY^t]$  for any positive integer i. Since the directions of  $b_1^i$  and  $b_2^i$  are random, the magnitudes of the summation terms in Eq. (87) and Eq. (88) are small in the mean sense. Therefore, the first term on the right hand side of Eq. (87) or Eq. (88) can be considered as the dominant term in real-world applications. Then, the error reduction property of Eq. (28) in Sec. 2 still holds for this suboptimal case.

In pratise, if **n** is large enough such that  $(P_B P_A)^n = P_{A \cap B}$ , and m > n, we can rewrite Eq. (87) and Eq. (88) as follows:

$$\hat{e}_{1}^{m} = e - \sum_{k=m-n+1}^{m} (P_{A}P_{B})^{m-k} b_{1}^{k} - \sum_{k=m-n+1}^{m-1} (P_{A}P_{B})^{m-1-k} P_{A} b_{2}^{k} , \qquad (89)$$

and

$$\hat{e}_{2}^{m} = e - \sum_{k=m-n+1}^{m} (P_{B}P_{A})^{m-k} P_{B}b_{1}^{k} - \sum_{k=m-n+1}^{m} (P_{B}P_{A})^{m-k}b_{2}^{k} , \qquad (90)$$

The error vector e in Eq. (89) and Eq. (90) is the vector in Eq. (31), which is the optimal least-squares error vector of the function-link network as shown in Fig. 2. We also see that no matter how big m is, there are at most  $\mathbf{n}$  vectors in each summation term of Eq. (89) and Eq. (90).

#### **8. EXPERIMENTAL RESULTS**

The theoretical results discussed above were tested with a speech signal **sampled** at **10 khz**. 100 Samples were used to train the network by the delta rule. **The** gain factor we used in the experiments was 0.001. No momentum term was used. The input pointwise nonlinear transformations used in the experiments are the following:

(A) SIGMOID 1 (Sig. I) : (0 < y < 1)  $y = \frac{1}{1 + e^{-x}}$ (B) SIGMOID 2 (Sig. II) : (-1 < y < 1)  $y = 2 \times \text{sigmoid } (x) - 1$ (C) THRESHOLD 1 (Th. I):  $y = 1 \text{ if } x \ge 0$  y = 0 if x < 0(D) THRESHOLD 2 (Th. II):  $y = -1 \text{ if } x \ge 0$  y = 1 if x < 0(E) SQUARE :  $y = z^{2}$ In the experiments, we first normalized the

In the experiments, we first normalieed the input data in the range  $\{-1,1\}$ . Five weights were used for each stage of a two-stage PSHNN. Ten weights were used for the function-link network. The initial matrix of the network was set equal to the covariance matrix of the input data.

Table 1 are the results of the function-link network with the ten weights listed as a function of the five types of NLT's.

Tables 2 thru 6 are the results of the two-stage PSHNN with forwardbackward training. Table 2 is for Sig.I, Table 3 for Sig.II, Table 4 for Th.I, Table 5 for Th.II, and Table 6 for the square NLT.

Tables 2 and 3 for **Sig.I** and **Sig.II** cases show that the PSHNN with forward-backward training has more error reduction and faster convergence rate than the function-link network. With **Th.II** and square **NLT's**, the PSHNN and the function-link network are about the same both in error reduction and convergence rate. With **Sig.II** NLT, there is negligible error reduction both in the PSHNN and the function-link network. This is because the input data was normalized in the range  $\{-1,1\}$ , and this causes x and y to be almost the same in this range.

**Tables** 7 and Table 8 are the results of the function-link network with three-stage input vectors of length 5 concatenated **as** a total input vector to the network. Tables 9 thru 11 show the error reduction performance of the corresponding three-stage PSHNN with forward-backward training. In the first stage, 100 iterations were used during the first sweep, and 300 iterations were used during the succeeding sweeps. The number of iterations of the second and the **third** stages were 500, and 900, respectively. In Tables 9, 10 and 11, the **notations** used mean errlf =  $||e_{1f}||^2$ , err2f =  $||e_{2f}||^2$ , err3f =  $||e_{3f}||^2$ , and err2b =  $||e_{2b}||^2$ . The superscript "i" denotes the number of sweeps as in Section 2. From Tables 7 and 8, we see that the convergence rate is rather slow for the function-link networks. Comparing Tables 7 and 8 to Tables 9, 10 and 11, we observe that PSHNN with forward-backward training is superior to the function-link network in terms of both convergence rate and error reduction.

#### 7. LEARNING INPUT NLT BY BACKPROPAGATION WITH FORWARD-BACKWARD TRAINING

The input NLT's of previous sections are all point-wise nonlinear transformations. Any input NLT is guaranteed to achieve error reduction [3], but it is important to learn which input NLT is optimal in error reduction. We can use backpropagation (BP) algorithm to learn the input NLT's. The BP algorithm involves a multi-layer system [9]. The goal of the BP algorithm is the same as the delta rule, namely, minimizing the square error sum. In this system, a nonlinear activation function is usually used at each layer. The activation function should be differentiable and usually monotone nondecreasing. The actual output of the jth node in the kth layer is

$$O_k(j) = f\left(\sum_{i=1}^{N_{k-1}} W(j,i) O_{k-1}(i)\right),$$

where  $N_{k-1}$  is the number of output nodes of the (k-1)th layer, and  $O_{k-1}$  is the output vector of the (k-1)th layer. f(.) is the nonlinear activation function.

Each stage of PSHNN can be any type of neural network. In this section, BP **stages** are utilized together with forward-backward training discussed in the previous sections. In other words, we modify the PSHNN architecture by using a multi-layer neural network trained by the BP algorithm at each stage instead of each stage of PSHNN being a two-layer network trained by the delta rule. With respect to Fig. 1, the first layer of the network is the input layer, and the input vector X is fed into each stage of the network. The outputs of the second layer are vectors ( $X^{\prime}$ , Y, and Z) of Fig. 1 which can be considered as the results of nonlinear transformations of each stage network (NLT1, NLT2, and NLT3 respectively). In order to comply with the error reduction properties discussed in the previous sections, we use linear activation function in the output layer.

The computer experiments discussed below indicate that the error reduction properties of forward-backward training hold in this case as well. The results are shown in Tables 12 thru 15. The same speech data of the previous sections is used. The length of the input layer at each stage is five, and a gain factor of **0.5** is **used** throughout.

Table 12 shows how error was reduced as a function of the number of iterations with a single BP network having 12 hidden units. The corresponding **PSHNN's** with the same number of interconnection weights were chosen as **2**-stage, **3-stage** and 4-stage networks in which each stage had 6, 4, and 3 hidden nodes, respectively, and their training was based on backpropagation. Tables 13, 14 and **15** show how error was reduced stage by stage and sweep by sweep of forward-backward training. **1000** forward-backward sweeps with the 2-stage network, **750** forward-backward sweeps with the **3-stage network** and 666

forward-backward sweeps with the 4-stage network are equivalent to 50000 iterations of the corresponding total BP network without stages since 50 iterations were used to train each stage of the PSHNN's. It is observed that the error reduction properties of the PSHNN's with two stages and three stages are better than those of the single BP network. The PSHNN's achieve the same error performance at about 600 sweeps with 2 stages and at 423 sweeps with 3 stages PSHNN as the single BP network achieves with 50000 iterations. Both the 2-stage and the 3-stage PSHNN's had a reduction of learning time by about 40%. It also appears that both the 2-stage and the 3-stage PSHNN's converge towards a deeper minimum than the single stage BP network, but this is not true with the 4-stage PSHNN. The 3-stage PSHNN performs best in term of deeper minimum and faster convergence rate. More experiments with different sets of data are needed to substantiate these properties. However, we think that this is the case since the same properties were observed in other applications with systems having

nonlinearities [15], [16].

21

#### 8. CONCLUSIONS

We showed theoretically that PSHNN's with forward-backward training of **n-stage** networks will achieve the same error reduction as the function-link networks with the pseudoinverse solutions. In practice, experimental results show that PSHNN's in many cases have faster convergence rate and better numerical error reduction than function-link networks. The property that **PSHNN's** can divide a large size network into several smaller size networks which **car** learn faster and more easily in training and operate in **parallel** in testing is believed to be significant for real-time implementation.

We proved in Ref. [3] that the PSHNN with any input nonlinear transformation have better performance than one-stage networks. By using additional neural networks, one can learn input NLT's at every parallel stage of PSHNN's. The PSHNN with BP stages and forward-backward training is one effective solution to this problem. When backpropagation is to be used, experiments indicate that better performance in terms of a deeper minimum and convergence rate is achieved when a single BP network is replaced by a PSHNN of equal complexity in which each stage is a BP network of smaller complexity than the single BP network.

With these properties, PSHNN's with continuous inputs and outputs and forward-backward training are expected to be useful in various applications of neural networks, adaptive signal processing, system identification and adaptive control.

#### REFERENCES

- 1. O. K. Ersoy, D. Hong, 'Parallel, Self-Organizing, Hierachical Neural Networks", *IEEE Tran. Neural Networks*, Vol. 1, No. 2, pp. 167-178, June 1990.
- 2. O. K. Ersoy, D. Hong, 'Parallel, Self-Organizing, Hierachical Neural Networks 11", to appear in *IEEE Tran. Industrial Electronics*, Special Issue on Neural Networks.
- 3. O. K. Ersoy and 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.
- 4. R. O. Duda, P.E. Hart, "Pattern Classification and Scene Analysis", John Wiley & Sons Inc., pp. 159-162, 1973.
- 5. C.L. Giles, T. Maxwell, "Learning, Invariance and Generalization in Higher Order Networks," *Applies Optics*, Vol. 26, No.23, pp. 4972-4978, December 1987.
- **6.** Y-M. Pao, "Adaptive Pattern Recognition and Neural Networks", Addison-Wesley Pub. Company, Inc., 1989.
- 7. G.Widrow, M.E. Hoff, "Adaptive Switching Circuits," Inst. Radio Engineers Western Electronic Show and Convention Record, Part 4, pp. 96-104, 1960.
- 8. O. K. Ersoy and S-W. Deng, "F'arallel, Self-Organizing, Hierachical Neural Networks, with Continuous Inputs and Outputs<sup>N</sup>, *Purdue University Tech. Report,* No. TR-EE-91-51, December 1991, and submitted to *IEEE Tran. Neural Networks,* December 1991.
- **9.** D. E. Rumelhart, "Parallel Distributed Processing", The **MIT** Press, Cambridge Mass., 1988.
- 10. S-W. Deng, O. K. Ersoy, "F'arallel, Self-Organizing, Hierachical Neural Networks with Circular Training", *Purdue University Tech. Report* No. TR-EE-91-16, April 1991.
- 11. H. Nakano, "Spectral Theory in the Hilbert Space", Japan Society for the Promotion of Science, 1953.
- 12. R.E. Hartwing and M.P. Drazin, "Lattice Properties of the \*-Order for Complex Matrices", J. of Math. Analysis and Applications, Academic Press, Inc., 1982.
- 13. L. D. Pyle, "A Generalized Inverse ε-Algorithm for Constructing Intersection Projection Matrices with Applications", Numerische Mathematik 10, pp 86-102, 1967.

- 14. S. T. Alexander, "Adaptive Signal Processing, Theory and Applications", Springer-Verlag, New York, pp. 68-85, 1986.
- 15. S. Aghagolzadeh, O. K. Ersoy, "Optimal Multistage Transform Image Coding", IEEE Tran. Circuits and Systems for Video Technology, December 1991.
- 16. O. K. Ersoy, J. Y. Zhuang, J. Brede, "An Iterative Interlacing Approach to the Synthesis of Computer-Generated Holograms", *Purdue University Tech. Report,* No. TR-EE-90-59, November 1990, and submitted to *Applied Optics.*

type of NTL	err	number of iterations
Sig.I	2.1344	1000
Th.II	2.027	1000
Sig.II	2.1291	1000
Th.I	2.0459	600
Sqre.	1.8862	1000

Table 1. Performance of the Function-Link Network in Speech Prediction  $(err = ||e||^2)$ .

Table 2. Performance of PSHNN with NLT Sig.I in Speech Prediction  $(\operatorname{err} 1=||e_1^i||^2, err 2=||e_2^i||^2).$ 

n-th			# of iterations	
	err1	err2	stage1	stage2
sweep				
n=1	2.1353	1.9336	100	1000
n=2	1.8718	1.8524	900	100
n=3	1.8460	1.8416	900	100

Table 3. Performance of PSHNN with NLT Sig.II in Speech Prediction  $(\operatorname{err} 1=||e_1^i||^2, \operatorname{err} 2=||e_2^i||^2).$ 

n-th			# of iterations	
sweep	err1	en2	stage1	stage2
n=1	2.1353	2.1396	100	1000
n=2	2.1343	2.1365	900	100
n=3	2.1336	-	900	-

n-th			# of iterations	
	err1	err2	stage1	stage2
sweep				
n=l	2.1352	2.0925	100	200
n=2	2.0699	2.0585	900	200
n=3	2.0514	2.0481	900	200
n=4	2.0457	2.0448	900	200

Table 4. Performance of PSHNN with NLT Th.I in Speech Prediction  $(\operatorname{err} 1 = ||e_1^i||^2, \operatorname{err} 2 = ||e_2^i||^2).$ 

Table 5. Performance of PSHNN with NLT Th.II in Speech Prediction (err  $1=||e_1^i||^2$ , err  $2=||e_2^i||^2$ ).

n-th			# of iterations	
	err1	err2	stage1	stage2
sweep				
n=1	2.1353	2.0282	100	100
n=2	2.0312	2.0250	500	100
n=3	2.0034	-	600	-

Table 6. Performance of PSHNN with NLT Square in Speech Prediction (err  $1=||e_1^i||^2$ , err  $2=||e_2^i||^2$ ).

n-th			# of iterations	
	err1	err2	stage1	stage2
sweep				
n=1	2.1353	1.9326	100	600
n=2	1.8973	1.8896	900	600
n=3	1.8872	1.8867	900	600
n=4	1.8864	1.8863	900	600

Table 7.3-Stage Function-Link Network as a Function of Input Nonlinearity with<br/>900 Iterations (err= $||e||^2$ ).

Type of	NTL	orr
Stage 11	Stage 111	
Sig.I	Th.II	2.0167
Th.I	Sig.I	1.9980
Square	Sig.I	1.8818

Table 8.3-Stage Function-Link Network as a Function of Input Nonlinearity with<br/>2900 Iterations (err= $||e||^2$ ).

Type of	NTL	err
Stage II	Stage III	CII
Sig.I	Th.II	2.0149
Th.I	Sig.I	1.9966
Square	Sig.I	1.8811

n-th	Square Error Sum			
Sweep Training	errlf	err2f	err3f	err2b
n=1 n=2	2.1353 1.8122	1.9377 1.7584	1.8393 1.7543	1.8758 -

Table 9. Performance of PSHNN with NLT1 Sig.I & NLT2 Th.II in Speech Prediction.

Table 10. Performance of PSHNN with NTL1 Th.I & NLT2 Sig.I in Speech Prediction.

n-th		Square E	rror Sum	
Training	err1f	err2f	err3f	err2b
n=1	2.1353	2.0924	1.9210	1.8957
n=2	1.8750	1.8592	1.8264	-

Table 11. Performance of PSHNN with NLTI Square & NLT2 Sig.I in Speech Prediction.

n-th	Square Error Sum			
Sweep Training	err <b>1</b> f	err2f	err3f	err2b
<b>n=</b> 1	2.1353	1.9330	1.6973	1.6812
n=2	1.6705	1.6631	1.6399	-

Table 12.Error Reduction with a Single Stage Network with 12 Hidden UnitsTrained by BP  $(err1=||e_1||^2)$ .

# of iterations	err
1000	1.1454
2000	0.8413
5000	0.6822
10000	0.4464
20000	0.2424
30000	0.2506
40000	0.2205
50000	0.1962

Table 13.Error Reduction with a Two-Stage PSHNN with 6 Hidden Units per SNN<br/>Trained by Forward-Backward BP (err  $1=||e_1||^2$ , err  $2=||e_2||^2$ ).

err1	err2
1.0528	1.0473
0.8962	0.8945
0.6031	0.6023
0.4374	0.4368
0.3367	0.3364
0.2714	0.2711
0.2133	0.2133
0.1927	0.1925
0.1895	0.1962
0.1771	0.1816
0.1731	0.1859
0.1658	0.1708
	err1 1.0528 0.8962 0.6031 0.4374 0.3367 0.2714 0.2133 0.1927 0.1895 0.1771 0.1731 0.1658

Table 14.	Error Reduction with a Three-Stage PSHNN with 4 Hidden Units per						
	SNN Trained by Forward-Backward BP.						

# of	errlf	err2f	err3f	err?h
sweep				CH20
10	1.2380	1.2157	1.2138	1.1982
50	0.6486	0.6464	0.6462	0.6447
100	0.5240	0.5236	0.5236	0.5235
200	0.4488	0.4487	0.4483	0.4484
300	0.2825	0.2823	0.2819	0.2817
423	0.1965	0.1965	0.1962	0.1962
500	0.1705	0.1704	0.1704	0.1703
600	0.1604	0.1604	0.1604	0.1603
700	0.1551	0.1551	0.1551	0.1551
750	0.1529	0.1529	0.1529	0.1529

Table 15.Error Reduction with a Four-Stage PSHNN with 3 Hidden Units per SNN<br/>Trained by Forward-Backward BP.

# of	err1f	err2f	err3f	err4f	err3b	err2h
sweep						0120
10	1.3594	1.3561	1.3238	1.3195	1.2963	1.2914
50	0.6716	0.6707	0.6682	0.6682	0.6662	0.6662
100	0.5121	0.5119	0.5116	0.5116	0.5115	0.5114
200	0.4136	0.4136	0.4134	0.4134	0.4134	0.4132
300	0.3540	0.3540	0.3539	0.3538	0.3538	0.3537
400	0.3093	0.3093	0.3092	0.3091	0.3090	0.3090
500	0.2620	0.2619	0.2618	0.2618	0.2618	0.2617
600	0.2306	0.2306	0.2305	0.2304	0.2303	0.2304
666	0.2210	0.2209	0.2209	0.2208	0.2208	0.2208



Figure 1. Block Diagram of a Three-Stage PSHNN.



Figure 2. Block Diagram of a Function-Link Network.



Figure 3. Graphical Representation of Suboptimal Solution for SNN1.



Figure 4. Graphical Representation of Suboptimal Solution for SNN2.