Initializing the weights in multiplayer network with quadratic sigmoid function

Abstract – A new method of initializing the weights in back propagation networks using the quadratic threshold activation function with one hidden layer is developed. The weights can be directly obtained even without further learning. This method relies on general position assumption of the distribution of the patterns. This method can be applied to many pattern recognition tasks. Finally, simulation results are presented, showing that this initializing technique generally results in drastic reduction in training time and the number of hidden neurons.

1 Introduction

Using neural networks for pattern recognition applications is more and more attractive. The main advantages of using neural networks are their massive parallelism, adaptability, and noise tolerance, etc (B. Widrow, and R. Winter, 1988)(J. J. Hopfield, 1982)(T. J. Sejnowski, and C. R. Rosenberg, 1986). One of the most popular neural networks is the back propagation (BP) or multilayer perceptron (MLP) neural network. The most commonly used activation function of BP is the hard \-limited threshold function and sigmoid function.

Using the hard-limited activation function in every neuron, the upper bound of the number of hidden neurons in a single-hidden layer network required for solving a general-position two-class classification problem is $\left\lceil \frac{K}{n} \right\rceil$, where K is the number of patterns, n is the input dimension. If without the general-position constraint, the upper bound of the number of the hidden neurons is $K - 1$.

Recently, a new quadratic threshold activation function is proposed (C. C. Chiang, 1993). By using it in each neuron, it is shown that the upper bound of the number of hidden neurons required for solving a given two-class classification problem can be reduced by one half compared with the conventional multilayer perceptrons which use the hard-limited threshold function, The results are given in table 1. Since the quadratic function is a little bit more complicated than the hard-limited threshold function, the learning is much difficult for the BP network with the quadratic function. Both the learning period and convergence properties are not

good enough to obtain effective results and can be observed in typical simulations. To relieve this learning difficulty, a new method for initializing weights in BP networks using the quadratic threshold activation function with one hidden layer is presented. The method based on the use of Gauss elimination; it is applicable to many classification tasks. The paper is organized as follows: the basic quadratic threshold function is described in section 2; this new initialization method is addressed in section 3; and finally, simulation results are shown in section 4.

Activation Function \ Problem	General-position	Not general-position
Hard-limited	$\left\lceil \frac{K}{n} \right\rceil$	K-1
QTF	$\left\lceil \frac{K}{2n} \right\rceil$	$\left\lceil \frac{K}{2} \right\rceil$

Table 1. Number of hidden neurons required

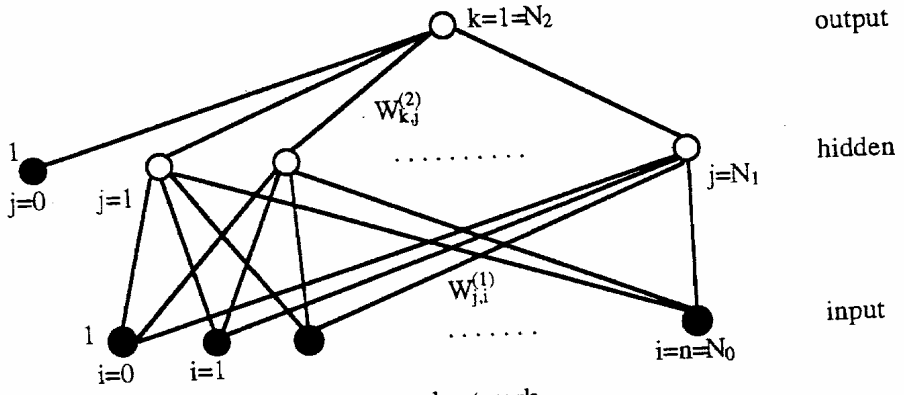


Figure 1. the one-hidden-layer neural network

section 3; and finally, simulation results are shown in section 4.

2 Quadratic Threshold Function

The quadratic threshold function (QTF) is defined as (C. C. Chiang, 1993)

Quadratic Threshold Function: $$f(net,\theta)=\begin{cases}0, & if\ net^2>\theta\\ 1, & if\ net^2\le\theta\end{cases}$$

In (C. C. Chiang, 1993), an upper bound on the number of hidden neurons required for implementing arbitrary dichotomy on a K-element set S in E^n is derived under the constraint that S is in general position.

Definition 1 *A K-element set S in E^n is in general position if no (j+1) elements in S in a (j-1)-dimensional linear variety for any j where $2 \le j \le n$.*

Proposition 1 (S. C. Huang, and Y. F. Huang, 1991, Proposition 1) *Let S be a finite set in E^n and S is in general position. Then, for any J-element subset S_1 of S, where $1 \le J \le n$, there is a hyperplane, which is an (n-1)-dimensional linear variety of E^n and no other elements of S.*

In (E. B. Baum), Baum proved that if all the elements of a K-element set S in E^n is in general position, then a single-hidden-layer MLP with $\left\lceil \frac{K}{n} \right\rceil$ hidden neurons using the hard-limited threshold function can implement arbitrary dichotomies defined on S. In (C. C. Chiang, 1993), it is proved that a two-layered (one hidden layer) MLP with at most $\left\lceil \frac{K}{2n} \right\rceil$ hidden neurons, which use the QTF, is capable of implementing arbitrary dichotomies of a K-element set S in E^n if S is in general position.

Since the quadratic threshold function is non-differentiable. To ease the derivation, we use the quadratic sigmoid function (QSF) as follows:

Quadratic Sigmoid Function: $$f(net,\theta)=\frac{1}{1+\exp(net^2-\theta)}$$

3 Description of this Method

Consider a classification problem consisting in assigning K vectors of R^n to 2 predetermined classes. Let the given training set $H = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K\} = \{H_0, H_1\}$ is partitioned into $K_0 \le K$ training vectors in subset H_0 corresponding to class 0, and $K_1 \le K$ training vectors in set H_1 corresponding to class 1, where $K_0 + K_1 = K$. And $H_0 = \{\mathbf{p}^1, \mathbf{p}^2, \ldots, \mathbf{p}^{K0}\}$, $H_1 = \{\mathbf{q}^1, \mathbf{q}^2, \ldots, \mathbf{q}^{K1}\}$. The classification can be implemented within a two-layer neural network with $N_0 + 1$ input units, $N_1 + 1$ hidden neurons, and N_2 output units, as illustrated in Figure 1. ($N_0 = n$, $N_1 = \left\lceil \frac{K}{2n} \right\rceil$, $N_2 = 1$).

According to Proposition 1, for any n-element subset S'_1 of S_1, which contains elements belong to for example class 1, there is a hyperplane which is a (n-1)-dimensional linear variety of E^n containing S'_1 and no other elements of S. We can use the Gauss elimination to solve the linear equations of the hyperplane that the n patterns of class 1 lie on.

Let us now describe the initialization scheme more formally. Let the input and hidden layers have each a bias unit with constant input activation value of 1, and the matrices of weights $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$ are between the input and the hidden layer, and between the hidden and the output layer, respectively.

$$W^{(2)} = \begin{pmatrix} W_{1,0}^{(1)}, W_{1,1}^{(1)}, \cdots, W_{1,n}^{(1)} \\ W_{2,0}^{(1)}, W_{2,1}^{(1)}, \cdots, W_{2,n}^{(1)} \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ W_{N_1,0}^{(1)}, W_{N_1,1}^{(1)}, \cdots, W_{N_1,n}^{(1)} \end{pmatrix} = \begin{pmatrix} W_{1,0}^{(1)}, W_1^{(1)} \\ W_{2,0}^{(1)}, W_2^{(1)} \\ \cdots\cdots \\ W_{N_1,0}^{(1)}, W_{N_1}^{(1)} \end{pmatrix}$$

Where $W_{j,i}^{(1)}$ represents the connection from the i-th input unit to the j-th hidden neuron, for $j = 1, 2, \ldots, N_1$, $i = 0, 1, \ldots, n$; $W_{1,j}^{(2)}$ represents the connection from the j-th hidden unit to the output neuron, where $j = 0, 1, 2, \ldots, N_1$;

$$W^{(2)} = (W_{1,0}^{(2)}, W_{1,1}^{(2)} \cdots, W_{1,N_1}^{(2)}) = (W_{1,0}^{(2)}, W_1^{(2)})$$

Let $\theta^{(1)}$, $\theta^{(2)}$ be the vector of θs values of the hidden and output layer, respectively. $\theta_j^{(1)}$ and $\theta_1^{(2)}$ represent the θ value of the j-th hidden neuron $j = 1, 2, \ldots, N_1$ and of the output neuron, respectively.

$$\theta^{(1)} = (\theta_1^{(1)}, \theta_2^{(1)} \cdots, \theta_{N_1}^{(1)}); \quad \theta^{(2)} = (\theta_1^{(2)})$$

If $K_0 \geq K_1$, we use the set of patterns in set H_1 corresponding to class 1 to obtain the weights values for the $\left\lceil \frac{K_1}{n} \right\rceil$ hidden neuron, so the number of the hidden neurons $N_1 = \left\lceil \frac{K_1}{n} \right\rceil \leq \left\lceil \frac{K}{2n} \right\rceil$. $\forall j \in \{1,2,\ldots,N_1\}$, $\forall i \in \{1,2,\ldots,n\}$, let

$$W_{j,0}^{(1)} = 1 \qquad (1)$$

$$W_{1,j}^{(2)} = 1 \qquad (2)$$

$$W_{1,0}^{(2)} = -1 \qquad (3)$$

$$\theta_j^{(1)} = \beta \qquad (4)$$

$$\theta_1^{(2)} = \beta \qquad (5)$$

where $0 < \beta < 1$ is a very small positive number.

In order for each j-th hidden neuron represents a different set of n input patterns, we solve the following equations to get $W_{j,i}^{(1)}$, for each $j = 1, 2, \ldots, \left\lfloor \frac{K_1}{n} \right\rfloor$, $i = 1, 2, \ldots, n$.

$$\begin{cases} \sum_{i=1}^{n} W_{j,i}^{(1)} q_i^{(j-1)n+1} + W_{j,0}^{(1)} = 0 \\ \sum_{i=1}^{n} W_{j,i}^{(1)} q_i^{(j-1)n+2} + W_{j,0}^{(1)} = 0 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \sum_{i=1}^{n} W_{j,i}^{(1)} q_i^{(j-1)n+n} + W_{j,0}^{(1)} = 0 \end{cases} \qquad (6)$$

If $\frac{K_1}{n}$ is a integer (i.e. $\left\lfloor \frac{K_1}{n} \right\rfloor = \left\lceil \frac{K_1}{n} \right\rceil = N_1$), here we done. If $\frac{K_1}{n}$ is not a integer (i.e. $\left\lfloor \frac{K_1}{n} \right\rfloor = \left\lceil \frac{K_1}{n} \right\rceil - 1 = N_1 - 1$), the rest $K_1 - n \times \left\lfloor \frac{K_1}{n} \right\rfloor$ patterns are then represented by the N_1-th hidden neuron using the following formula. For $j = N_1$, $i = 1, 2, \ldots, K_1 - n \times \left\lfloor \frac{K_1}{n} \right\rfloor$.

$$\begin{cases} \sum_{i=1}^{n} W_{j,i}^{(1)} q_i^{n*\left\lfloor \frac{K_1}{n} \right\rfloor+1} + W_{j,0}^{(1)} = 0 \\ \sum_{i=1}^{n} W_{j,i}^{(1)} q_i^{n*\left\lfloor \frac{K_1}{n} \right\rfloor+2} + W_{j,0}^{(1)} = 0 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \sum_{i=1}^{n} W_{j,i}^{(1)} q_i^{K_1} + W_{j,0}^{(1)} = 0 \end{cases} \qquad (7)$$

The equations can solve $W_{N_1,i}^{(1)}$ for $i = 1, 2, \ldots, K_1 - n \times \left\lfloor \frac{K_1}{n} \right\rfloor$. Let the other $n - K_1 + n \times \left\lfloor \frac{K_1}{n} \right\rfloor$ weights connects to the N_1-th hidden neuron be zero.

If $K_0 < K_1$, we use the set of patterns in set H_0 corresponding to class 0 to obtain the weights values for the $\left\lceil \frac{K_1}{n} \right\rceil$ hidden neuron, so the number of the hidden neurons

$N_1 = \left\lceil \frac{K_0}{n} \right\rceil < \left\lceil \frac{K}{2n} \right\rceil$. Everything is the same except that let

$$W_{1,0}^{(2)} = 0 \qquad (8)$$

and **q** is replaced by **p**. We now solve the following equations to get $W_{j,i}^{(1)}$, for each j $= 1, 2, \ldots, \left\lfloor \frac{K_n}{n} \right\rfloor$, $i = 1, 2, \ldots, n$.

$$\begin{cases} \sum_{i=1}^{n} W_{j,i}^{(1)} p_i^{(j-1)n+1} + W_{j,0}^{(1)} = 0 \\ \sum_{i=1}^{n} W_{j,i}^{(1)} p_i^{(j-1)n+2} + W_{j,0}^{(1)} = 0 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \sum_{i=1}^{n} W_{j,i}^{(1)} p_i^{(j-1)n+n} + W_{j,0}^{(1)} = 0 \end{cases} \qquad (9)$$

If $\frac{K_0}{n}$ is not a integer, the weights connects inputs and the N_1-th hidden neuron can be obtained the same way as for the case $K_0 \geq K_1$.

To test the results, when $K_0 \geq K_1$, with these initial values of the weights, an input $\mathbf{x} \in H_1$ such:

$$W_j^{(1)} \cdot x + W_{j,0}^{(1)} = 0$$

means **x** lies on the hyperplane that the j-th hidden neuron represents. This input will cause only the j-th hidden unit to have an activation value close to 1, since

$$net_j^{(1)} = W_j^{(1)} \cdot x + W_{j,0}^{(1)} = 0,$$

where, $net_j^{(1)}$ represent the net input magnitude of the j-th hidden neuron. And

$$-\sqrt{\beta} < net_j^{(1)} = 0 < \sqrt{\beta}$$

The other hidden neurons $t \neq j$ will not get activated because

$$net_t^{(1)} = W_t^{(1)} \cdot x + W_{j,0}^{(1)} \neq 0,$$

and

$$\begin{aligned} &if \quad net_t^{(1)} > 0: \quad -\sqrt{\beta} < net_t^{(1)} \sim< \sqrt{\beta} \quad ; \\ &if \quad net_t^{(1)} < 0: \quad -\sqrt{\beta} \sim< net_t^{(1)} < \sqrt{\beta} \end{aligned}$$

Consequently, provided the activation of the activation of the j-th hidden neuron is

close to 1 with a QSF and all other neurons are 0, the output neuron will also get activated, since

$$net_1^{(2)} = W_1^{(2)} \cdot y + W_{1,0}^{(2)} = 1 * y_i - 1 = 0,$$

$$-\sqrt{\beta} < net_1^{(2)} = 0 < \sqrt{\beta}$$

where **y** represent the output vector of the hidden layer, $net_1^{(2)}$ represent the net input magnitude of the output neuron. If $K_0 < K_1$, there is a similar situation.

4 Simulations

This section provides the results of applying the above method for the initialization of three simple problems. The first is the XOR problem, the second is the parity problem, and the third is the gray-zone problem. They are trained with this algorithm using a two-layer neural network. All the simulations have been performed using BP with a momentum term. In this algorithm, the weights and θs are updated after a training pattern, according to the following equations (C. C. Chiang, 1993):

$$\Delta W_{j,i}^{(1)}(t) = -\mu_1 \frac{\partial E}{\partial W_{j,i}^{(1)}(t-1)} + \alpha_1 \Delta W_{j,i}^{(1)}(t-1)$$

$$\Delta W_{1,j}^{(2)}(t) = -\mu_1 \frac{\partial E}{\partial W_{1,j}^{(2)}(t-1)} + \alpha_1 \Delta W_{1,j}^{(2)}(t-1)$$

$$\Delta \theta_j^{(1)}(t) = -\mu_2 \frac{\partial E}{\partial \theta_j^{(1)}(t-1)} + \alpha_2 \Delta \theta_j^{(1)}(t-1)$$

$$\Delta \theta_1^{(2)}(t) = -\mu_2 \frac{\partial E}{\partial \theta_1^{(2)}(t-1)} + \alpha_2 \Delta \theta_1^{(2)}(t-1)$$

where $j = 0, 1, 2, \ldots, N_1$; $i = 0, 1, \ldots, n$; $E = \frac{1}{2}(o-d)^2$; o = the actual output; d = the desired output; t = a discrete time index; α_1 = the momentum coefficient for weights; α_2 = the momentum coefficient for θs; μ_1 = the learning rate for weights; μ_2 = the learning rate for θs.

The exclusive-or (XOR) function has been widely used as a benchmark example to test the neural network's learning ability. Figure 2 depicts the learning curves of the XOR problem for our method, random QSF BP, and random Sigmoid BP. The learning curves of the random QSF BP and the random sigmoid BP are the average results of 50 times training using 50 different sets of initial random weights all between –0.1 and +0.1 to relieve the effect of different initial weights. In the simulations, the parameter were chosen as follows: $\mu_1 = 0.3$, $\mu_2 = 2.0$, $\alpha_1 = 0.3$, $\alpha_2 =$

0.005, $\beta = 0.09$. This figure shows that the evaluated mean squared error decreasing with the epoch number. An "epoch" means the presentations of the whole training set. As shown in Figure 2, our learning speed is far superior to the learning speed of the QSF BP and the random Sigmoid BP with the same learning rate and even though we use less number of hidden neurons.

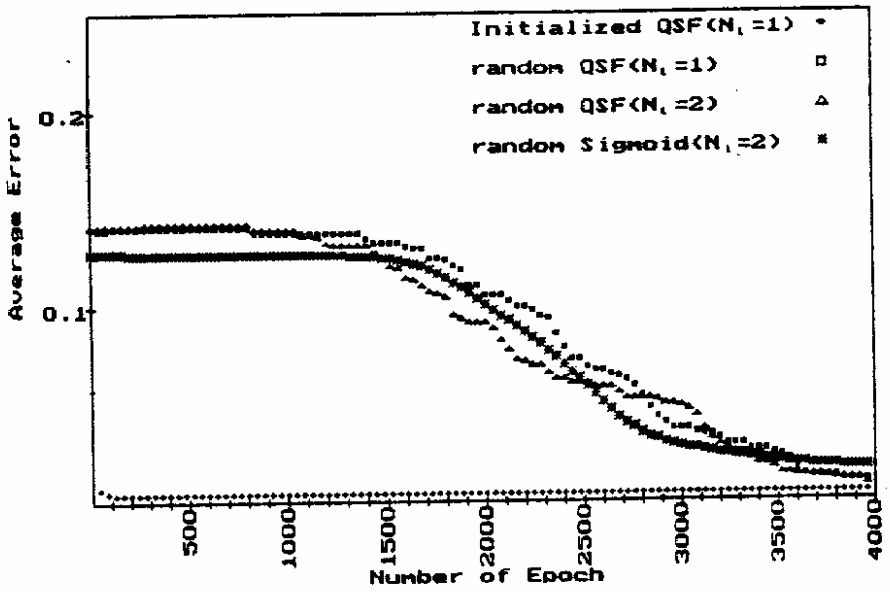


Figure 2. Plots of learning curves for the XOR problem

The parity problem contains 32 8-dimensional binary training patterns. If the sum of every bit of the input pattern is odd, the output is 1, otherwise 0. For example, presentation of 00001111 should result in an output of '0' and presentation of 01000110 should result in an output of '1'. Figure 3 shows the learning curves of the parity problem for our method, random QSF BP, and conventional BP. In the same way, the learning curves of the random QSF BP and conventional BP are the average results of 50 times training using 50 different sets of initial random weights all between –0.1 and +0.1. In the simulations, the parameter were chosen as follows: $\mu_1 = 0.01$, $\mu_2 = 2.0$, $\alpha_1 = 0.001$, $\alpha_2 = 0.005$, $\beta = 0.09$. With the same learning rate and less number of hidden neurons, our learning speed is far superior to the others.

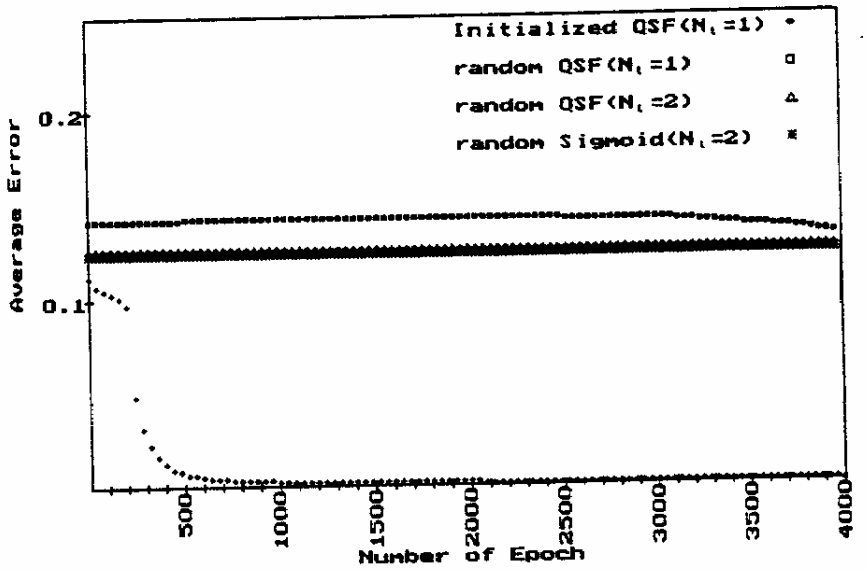


Figure 3. Plots of learning curves for the Parity problem

Figure 4 is the distribution of the training patterns of the gray-zone problem on the 2-dimension plane. Circle and cross present class 0 and class 1, respectively. Figure 5 is the learning curves for our method, random QSF BP, and random sigmoid BP ($\mu_1 = 0.2$, $\mu_2 = 0.9$, $\alpha_1 = 0.02$, $\alpha_2 = 0.005$, $\beta = 0.04$). Figure 6 is the result of using our method after 200 epoches. The white zone is class 0 (i.e. the output is smaller than 0.2), while the black zone is class 1 (i.e., the output is larger than 0.8). Some data is included in neither class 0 nor class 1 (i.e., the output is between 0.2 and 0.8), as shown in the gray zone. Therefore in such training patterns as figure 4, the ambiguous ones will be grouped to the gray zone. So our method can solve the ambiguous problem of the training patterns.

5 Conclusion

In this paper, we propose a new method to initialize the weights of QSF BP neural networks. According to the theoretical results, the upper bound on the number of the hidden neurons is reduced to be a half of the single-hidden-layer MLP used. According to the simulation results, the initialized neural network is far superior to the conventional BP and random QSF BP both in learning speed and network size.

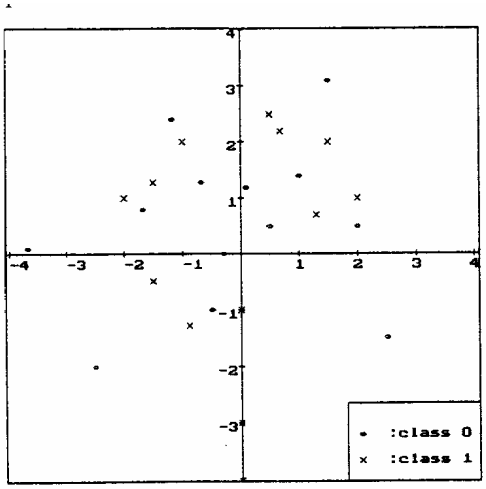


Figure 4. Training patterns of the Grey-Zone problem

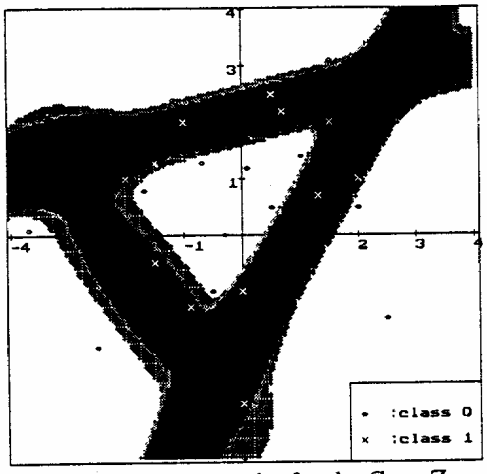


Figure 5. Training results for the Grey-Zone problem

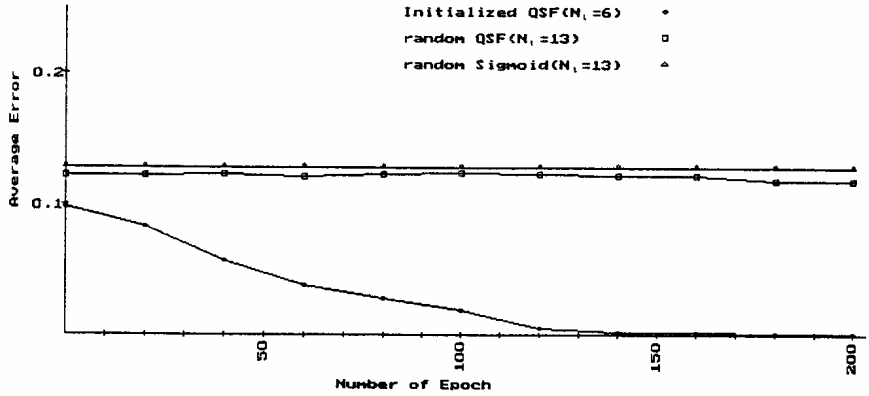


Figure 6. Plots of learning curves for the Grey-Zone problem

Exercises

3.1 Note that the MLP networks using the quadratic threshold activation function defined as

$$f(net,\theta)=\begin{cases}0 & if\ net^2>\theta\\ 1 & if\ net^2\le\theta\end{cases}$$

can implement dichotomy problems. The patterns as shown in Figure P1 have to be classified in two categories using a layered network. Figure P2 is the two-layer classifier using the quadratic threshold activation function. The values in the circles are the values of θ corresponding to the neurons. Figure P3 is the

logic equivalent diagram. (1)~(5) Fill in the appropriate weights in the Figure P2. (6) Fill in the appropriate value of θ in the Figure P2. (7) Write down the correct interpretation of the logic gate in Figure P3. (8) Sketch the partitioning of the pattern space.

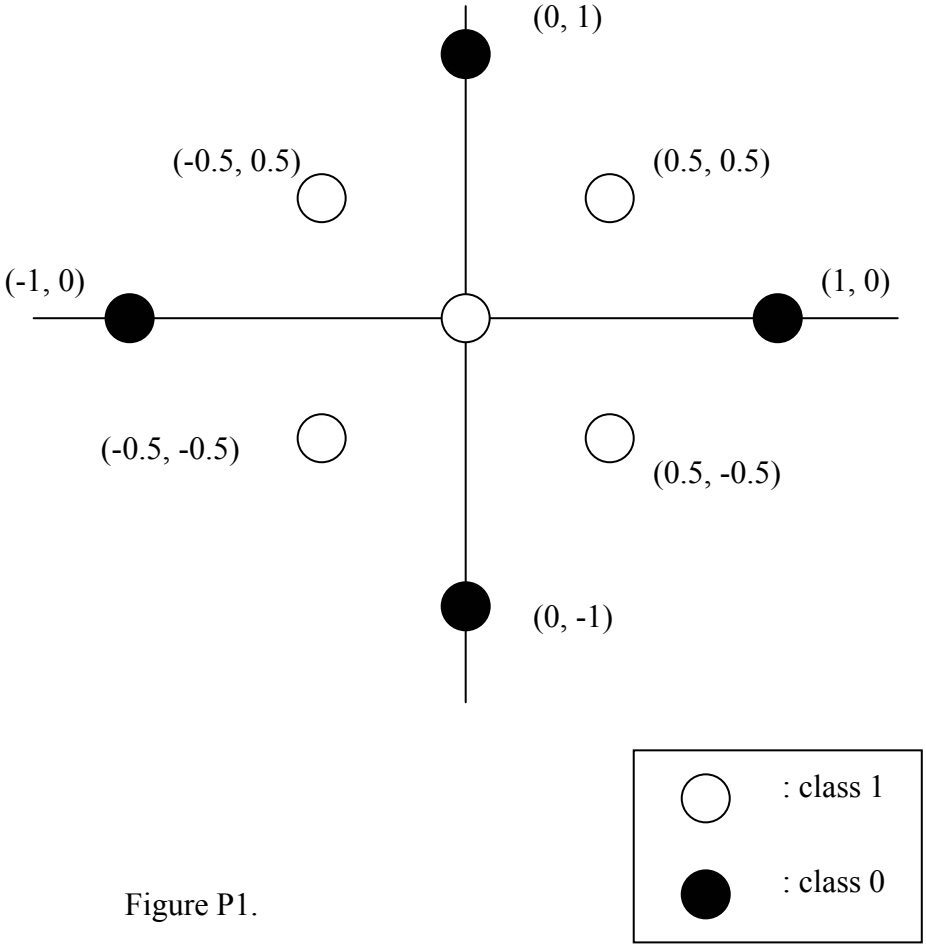


Figure P1.

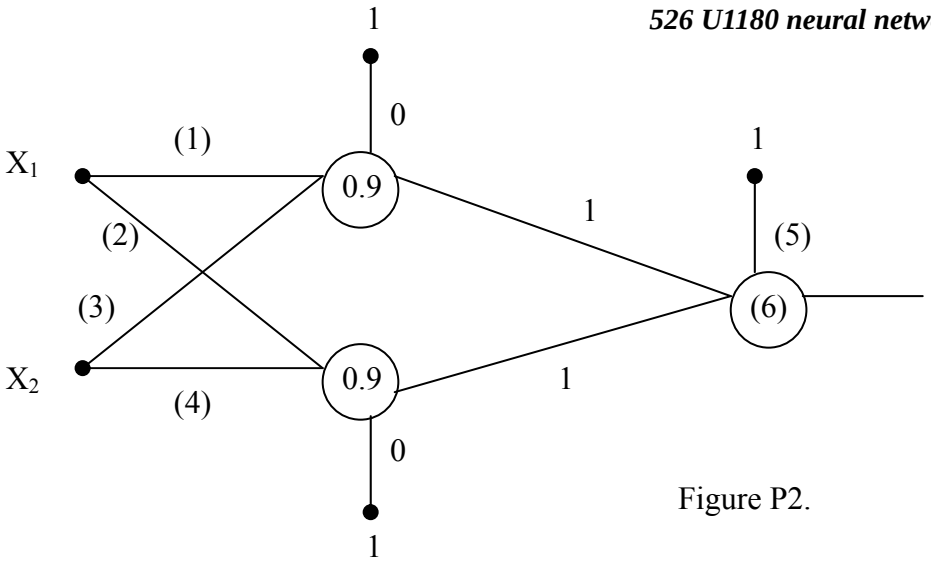


Figure P2.

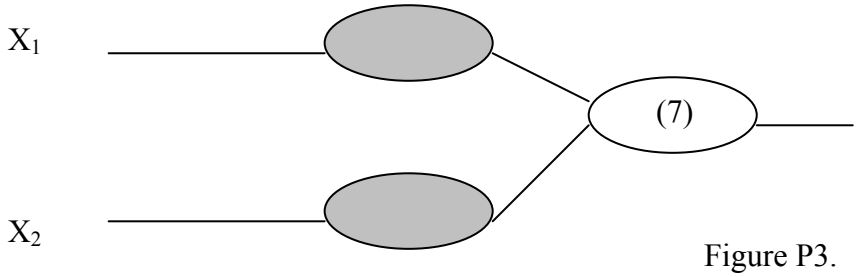
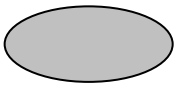


Figure P3.

The truth table of  ::

In	0	1	-1	0.5	-0.5
Out	1	0	0	1	1

1. B. Widrow, and R. Winter, (1988). "Neural nets for adaptive filtering and adaptive pattern recognition," *IEEE Computer*, pp. 25-39, March, 1988.

2. C. C. Chiang, (1993). *The study of supervised-learning neural models,* Thesis of the Degree of Doctor of Philosophy, Dept. of Computer Science and Information Engineering College of Engineering, National Chiao-Tung Univ., Hsin-Chu, Taiwan.

3. E. B. Baum. "On the capability of multilayer perceptron, "

4. J. J. Hopfield, (1982). "Neural networks and physical systems with emergent collective computational abilities," *Proc. of the National Academy of Sciences*,

79:2254-2558.

5. T. J. Sejnowski, and C. R. Rosenberg, (1986). “NETalk: a parallel network that learns to read aloud,” The Johns Hopkins University Electrical Engineering and Computer Science Tech. Report, JHU/EECS-86/01.
6. S. C. Huang, and Y. F. Huang, (1991). “Bounds on the number of hidden neurons in multilayer perceptrons,” *IEEE Trans. on Neural Networks*, vol. 2, pp. 47-55.