LETTER Communicated by Joydeep Ghosh

Training Pi-Sigma Network by Online Gradient Algorithm with Penalty for Small Weight Update

Yan Xiong
xiongyan888@sohu.com
Department of Applied Mathematics, Dalian University of Technology, Dalian 116024, People's Republic of China, and Faculty of Science, University of Science and Technology Liaoning, Anshan, 114051, People's Republic of China

Wei Wu
wuweiw@dlut.edu.cn
Xidai Kang
kxd_005@163.com
Chao Zhang
zhangchao_fox@163.com
Department of Applied Mathematics, Dalian University of Technology, Dalian 116024, People's Republic of China

A pi-sigma network is a class of feedforward neural networks with product units in the output layer. An online gradient algorithm is the simplest and most often used training method for feedforward neural networks. But there arises a problem when the online gradient algorithm is used for pi-sigma networks in that the update increment of the weights may become very small, especially early in training, resulting in a very slow convergence. To overcome this difficulty, we introduce an adaptive penalty term into the error function, so as to increase the magnitude of the update increment of the weights when it is too small. This strategy brings about faster convergence as shown by the numerical experiments carried out in this letter.

1 Introduction

Recently higher-order neural networks have attracted much attention from researchers due to their nonlinear mapping ability with fewer units. Several kinds of neural networks have been developed to use higher-order combinations of input information, including higher-order neural networks (Giles & Maxwell, 1987), sigma-pi neural networks, and product unit neural networks (Durbin & Rumelhart, 1989). The higher-order neural networks considered in this letter are pi-sigma networks (PSN), which were introduced by Shin and Ghosh (1991) and explored by Hussaina and Liatsisb (2002), Li (2002, 2003), Sinha, Kumar, and Kalra (2000), and

Voutriaridis, Boutalis, and Mertzios (2003). PSNs use product cells as the output units to indirectly incorporate the capabilities of higher-order networks while using fewer weights and processing units and have been used effectively in pattern classification and approximation (Shin & Ghosh, 1991; Xin, 2002; Jiang, Xu, & Piao, 2005). An online gradient algorithm with a randomized modification is suggested for training PSN (Shin & Ghosh, 1992). However, we notice that the update increment of the weights may become very small when the magnitude of weights is small, as usually happens early in training, resulting in a very slow convergence. To overcome this difficulty, we introduce an adaptive penalty term into the error function, so as to increase the magnitude of the update increment of the weights when it is too small. This strategy brings about faster convergence, as shown by the numerical experiments carried out in this letter.

The rest of this letter is organized as follows. Section 2 provides a brief introduction to PSN and the online gradient algorithm and specifies the small increment problem. In section 3, we present an online algorithm with a penalty for training PSN. Section 4 provides a few simulation results that illustrate the effectiveness of our approach.

2 PSN and Randomized Online Gradient Algorithm

2.1 Structure of PSN. Assume that the number of neurons for the input, summation, and product layers are P, N, and 1, respectively. The structure is shown in Figure 1. We denote by $w_n = (w_{n1}, w_{n2}, \ldots, w_{nP})^T$ $(1 \le n \le N)$ the weight vector connecting the input and the summation node n, and write $w = (w_1^T, w_2^T, \ldots, w_N^T) \in \mathbb{R}^{NP}$. $\xi = (\xi_1, \ldots, \xi_P)^T \in R^P$ stands for the input vector. Here we have added a special input unit $\xi_P = -1$ corresponding to the biases w_{nP}. The weights on the connections between the product node and the summation nodes are fixed to 1. Let $g : \mathbb{R} \to \mathbb{R}$ be a given activation function for the output layer. For any given input ξ and weight w, the output of the network is

$$y = g\left(\prod_{i=1}^{N}(w_i \cdot \xi)\right). \tag{2.1}$$

2.2 Randomized Online Gradient Algorithm for PSN. The online gradient algorithm is a simple and efficient learning method for feedforward neural networks. Usually PSN and the networks with pi-sigma building blocks are also trained by it but with a randomized modification (or an asynchronous modification, which is not considered in this letter).

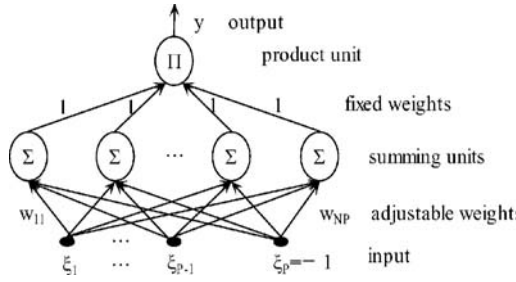

Figure 1: A pi-sigma network.

Let $\{\xi^j, O^j\}_{j=1}^J \subset \mathbb{R}^P \times \mathbb{R}$ be the training examples. The usual mean square error function for the network is

$$E(w) = \frac{1}{2}\sum_{j=1}^{J}(O^j - y^j)^2 = \frac{1}{2}\sum_{j=1}^{J}\left(O^j - g\left(\prod_{i=1}^{N}(w_i \cdot \xi^j)\right)\right)^2. \tag{2.2}$$

The gradient of the error function with respect to w_n $(n = 1, 2, \ldots, N)$ is

$$\begin{aligned} D_n E(w) = &-\sum_{j=1}^{J}\left(O^j - g\left(\prod_{i=1}^{N}(w_i \cdot \xi^j)\right)\right) g' \\ &\times \left(\prod_{i=1}^{N}(w_i \cdot \xi^j)\right)\left(\prod_{\substack{i=1 \\ i\neq n}}^{N}(w_i \cdot \xi^j)\right)\xi^j. \end{aligned} \tag{2.3}$$

The updating rule of the online gradient algorithm with a randomized modification for the weight vector of PSN is as follows (Shin & Ghosh, 1991). Start from an arbitrary initial guess w^0. At the mth step $(m = 0, 1, \ldots)$ of the training iteration of the weights, we randomly choose an input example ξ^j and a summing unit n and update the corresponding connecting weights while the weights of the other summing units remain unchanged, so the online gradient algorithm iteratively updates the weights in the fashion that

$$w_n^{m+1} = w_n^m + \Delta_j w_n^m \tag{2.4a}$$

$$w_q^{m+1} = w_q^m, \quad q \neq n \tag{2.4b}$$

where

$$\Delta_j w_n^m = \eta \left(O^j - g\left(\prod_{i=1}^{N} \left(w_i^m \cdot \xi^j \right) \right) \right) g' \times \left(\prod_{i=1}^{N} \left(w_i^m \cdot \xi^j \right) \right) \left(\prod_{\substack{i=1 \\ i \neq n}}^{N} \left(w_i^m \cdot \xi^j \right) \right) \xi^j \tag{2.5}$$

and $\eta > 0$ is the learning rate.

In this letter, we use C for a general positive constant, which may be different in different places but is independent of p, n, j, m, and η.

2.3 Influence of Small Weights on the Training. Condition A on the activation function will be needed in our analysis: $|g(t)|$ and $|g'(t)|$ are uniformly bounded for $t \in \mathbb{R}$. It is satisfied by, for example, sigmoid functions, which are the most popular activation functions.

Now let us illustrate the influence of the small weights on the training process. The following proposition estimates the magnitudes of the increment of the weight vector $\Delta_j w_n^m$ and the error function $\Delta E(w^m)$, respectively.

Proposition 1. *Let condition A be satisfied and the sequence $\{w^m\}_{m=1}^{\infty}$ be generated from the learning algorithm, 2.4. Then there exists a positive constant C independent of η, n, p, j, and m, such that*

$$\left\| \Delta_j w_n^m \right\| \leq C \eta \delta_m^{N-1} \tag{2.6}$$

and

$$|\Delta E(w^m)| \leq C \eta \delta_m^{2(N-1)}, \tag{2.7}$$

where $\delta_m = \max_{1 \leq n \leq N} \|w_n^m\|$, and δ_m^{N-1} and $\delta_m^{2(N-1)}$ denote the $(N-1)$th and $2(N-1)$th powers of δ_m, respectively.

Proof. Estimate 2.6 can be obtained directly from the definition of δ_m, equation 2.5, and the Cauchy-Schwartz inequality.

To verify equation 2.7, we first expand $\prod_{i=1}^{N}(w_i^{m+1}\cdot\xi^j)$ at $\prod_{i=1}^{N} w_n^m\cdot\xi^j$ by the Taylor theorem:

$$g\left(\prod_{i=1}^{N}\left(w_i^{m+1}\cdot\xi^j\right)\right)=g\left(\left(\left(w_n^m+\Delta_j w_n^m\right)\cdot\xi^j\right)\prod_{\substack{i=1\\ i\neq n}}^{N}\left(w_i^m\cdot\xi^j\right)\right)$$

$$=g\left(\prod_{i=1}^{N}\left(w_i^m\cdot\xi^j\right)\right)+g'\left(t_{n,j}^m\right)\left(\prod_{\substack{i=1\\ i\neq n}}^{N}\left(w_i^m\cdot\xi^j\right)\right)\left(\Delta_j w_n^m\cdot\xi^j\right),\qquad(2.8)$$

where $t_{n,j}^m$ is a real number between $\prod_{i=1}^{N}(w_i^{m+1}\cdot\xi^j)$ and $\prod_{i=1}^{N}(w_i^m\cdot\xi^j)$. According to equation 2.6 and the Cauchy-Schwartz inequality, we have

$$\left|g\left(\prod_{i=1}^{N}\left(w_i^{m+1}\cdot\xi^j\right)\right)-g\left(\prod_{i=1}^{N}\left(w_i^m\cdot\xi^j\right)\right)\right|\le C\eta\delta_m^{2(N-1)}\max_{\substack{x\in R\\ 1\le j\le J}}\{g'(x),\ \|\xi^j\|^N\}$$

$$\le C\eta\delta_m^{2(N-1)}.\qquad(2.9)$$

This inequality, together with equation 2.2, implies

$$|\Delta E(w^m)|=|E(w^{m+1})-E(w^m)\left|\le\frac{1}{2}\sum_{j=1}^{J}\right|\left(O^j-g\left(\prod_{i=1}^{N}\left(w_i^{m+1}\cdot\xi^j\right)\right)\right)^2$$

$$-\left(O^j-g\left(\prod_{i=1}^{N}\left(w_i^m\cdot\xi^j\right)\right)\right)^2\left|=\frac{1}{2}\sum_{j=1}^{J}\right|g\left(\prod_{i=1}^{N}\left(w_i^{m+1}\cdot\xi^j\right)\right)$$

$$-g\left(\prod_{i=1}^{N}\left(w_i^m\cdot\xi^j\right)\right)\left|\left|O^j-g\left(\prod_{i=1}^{N}\left(w_i^{m+1}\cdot\xi^j\right)\right)+O^j-g\left(\prod_{i=1}^{N}\left(w_i^m\cdot\xi^j\right)\right)\right|\right.$$

$$\le CJ\eta\delta_m^{2(N-1)}\le C\eta\delta_m^{2(N-1)}.$$

It is well known that the initial guess w^0's, and hence δ_0, are required to be small in magnitude in order to avoid premature saturation of the weights. Then, we notice from equations 2.6 and 2.7 that $\Delta_j w_n^m$ and $\Delta E(w^{m+1})$ are even smaller when δ_m is small, resulting in a very slow convergence early in training. The situation will get worse for big N, as illustrated in Figure 2a. By proposition 1, a possible remedy might be to use a larger learning rate η to accelerate the convergence. This strategy is actually used in Shin and Ghosh (1992, 1995). But a constant large learning rate may lead to oscillation of the error function when the weights are not small, as illustrated in Figure 2(b).Thus, an adaptive variable learning rate seems desirable but is

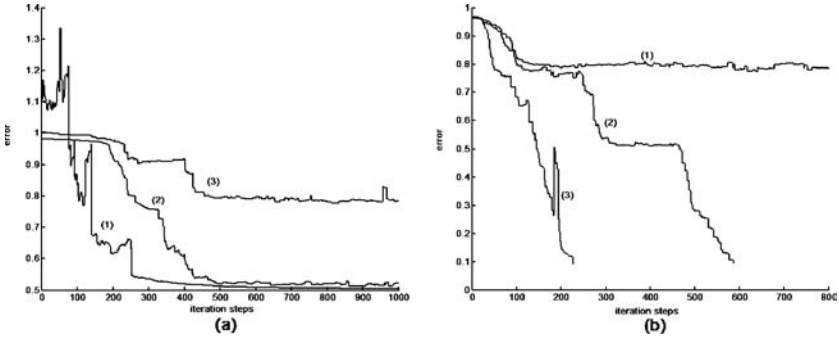

Figure 2: The behaviors of PSN without penalty for a three-dimensional parity problem (cf. section 4.2). The number of neurons for the input and product layers are $P = 4$ and 1, respectively. The initial weights are selected randomly within the interval $[-0.15,\ 0.15]$. (a) The results of PSN with different numbers of summing units. The learning rate is 0.8 for (1) and 0.3 for (2) and (3). (1) $N = 2$; (2) $N = 3$; (3) $N = 4$. (b) The results of PSN with different learning rates. The number of summing units is $N = 3$: (1) $\eta = 0.5$; (2) $\eta = 1.0$; (3) $\eta = 1.5$.

not easy to define directly. The above argument motivates us to introduce in the next section a penalty for small weights into the error function, so as to resolve the problem in an adaptive manner.

3 Online Gradient Algorithm with a Penalty

Recall that the original task of the network training is to find w^* such that $E(w^*) = \min_w E(w)$. Now we add a constraint on $\prod_{i=1}^{N}(w_i^m \cdot \xi^j)$ to obtain a constrained optimization problem,

$$\begin{aligned} &\min_w E(w) \\ &s.t. \left| \prod_{i=1}^{N} (w_i \cdot \xi^j) \right| > \gamma, \quad j = 1, 2, \ldots, J, \end{aligned} \tag{3.1}$$

where $\gamma > 0$ is an appropriate constant. This constrained optimization problem can be converted into a nonconstrained problem by using a Lagrangian multiplier λ:

$$\min_w \widetilde{E}(w) \equiv \min_w \left\{ E(w) + \frac{\lambda}{2} \sum_{j=1}^{J} p_j(w) \right\}, \tag{3.2}$$

where $\lambda > 0$ is the penalty coefficient, and $p_j(w)$ $(j = 1, 2, \ldots, J)$ is the penalty function defined as

$$p_j(w) = \begin{cases} \left(\gamma - \left|\prod_{i=1}^{N}(w_i \cdot \xi^j)\right|\right)^2, & -\gamma < \prod_{i=1}^{N}(w_i \cdot \xi^j) < \gamma \\ 0, & \text{elsewhere} \end{cases}.$$

Finally, we solve the new nonlinear optimization problem, equation 3.2, by the online gradient algorithm. So the gradient of the error function with respect to w_n $(n = 1, 2, \ldots, N)$ is replaced by

$$D_n\widetilde{E}(w) = -\sum_{j=1}^{J}\left(O^j - g\left(\prod_{i=1}^{N}(w_i \cdot \xi^j)\right)\right)g'\left(\prod_{i=1}^{N}(w_i \cdot \xi^j)\right)\left(\prod_{\substack{i=1\\ i\neq n}}^{N}(w_i \cdot \xi^j)\right)\xi^j + \frac{\lambda}{2}\sum_{j=1}^{J} D_n p_j(w), \tag{3.3}$$

where

$$D_n p_j(w) = \begin{cases} -2\left(\gamma - \prod_{i=1}^{N}(w_i \cdot \xi^j)\right)\prod_{\substack{i=1\\ i\neq n}}^{N}(w_i \cdot \xi^j)\xi^j, & 0 \le \prod_{i=1}^{N}(w_i \cdot \xi^j) < \gamma \\ 2\left(\gamma + \prod_{i=1}^{N}(w_i \cdot \xi^j)\right)\prod_{\substack{i=1\\ i\neq n}}^{N}(w_i \cdot \xi^j)\xi^j, & -\gamma < \prod_{i=1}^{N}(w_i \cdot \xi^j) < 0 \\ 0, & \text{elsewhere} \end{cases}.$$

In place of equations 2.4a and 2.5, we have, respectively,

$$w_n^{m+1} = w_n^m + \widetilde{\Delta}_j w_n^m \tag{3.4}$$

and

$$\widetilde{\Delta}_j w_n^m = \eta\left(\left(O^j - g\left(\prod_{i=1}^{N}(w_i^m \cdot \xi^j)\right)\right)g'\left(\prod_{i=1}^{N}(w_i^m \cdot \xi^j)\right)\left(\prod_{\substack{i=1\\ i\neq n}}^{N}(w_i^m \cdot \xi^j)\right)\xi^j - \frac{\lambda}{2}D_n p_j(w^m)\right) \tag{3.5}$$

From equations 3.1 and 3.2, we see that γ decides when the penalty starts to activate, and λ determines how strong the activation is. Because we aim to penalize only the small weight update, the value of γ should not

be too big. On the other hand, λ needs to be large as usually required in optimization problems. A practical choice of γ and λ can be made in terms of experience or simulation experiment. Our experiments below indicate that the values of γ and λ can be chosen in a comparatively wide range.

4 Simulation Results

In this section, we test the performance of our penalty approach by two simulation examples. The training iteration process stops when one of the following criteria is met:

Criterion 1: The iteration step is less than 5000, and $\|\triangle_j w_n^m\| < 0.00001$ for three successive iterations.
Criterion 2: $\|\triangle_j w_n^m\| < 0.00001$ is not valid for three successive iterations, but the iteration step has reached 5000.

4.1 Example 1: Function Approximation Problems. First, we trained the PSN to approximate the function $f(x) = \cos(\pi x/2),\ x \in [-1,\ 1]$. The aim of this test is to compare the behaviors of the online gradient algorithms with or without penalty. The number of neurons for the input, summation, and product layers are $P = 2,\ N = 3$ and 1, respectively. The output activation function is $g(x) = \frac{1}{1+e^{-x}}$. The parameters in this example take the values $\eta = 0.15,\ \gamma = 0.3$, and $\lambda = 40$, and the initial weights are selected randomly within the interval $[-0.15,\ 0.15]$. Figure 3a shows the change of the error function $E(w)$ in the case without penalty, where the curve decreases slowly and keeps flat until the iteration step 4012. The value of the product unit is accordingly small, as shown in Figure 3b. The result of the online gradient algorithm with a penalty is shown in Figures 3c and 3d. The new error function $\tilde{E}(w)$ defined in equation 3.2 decreases fairly fast and meets Criterion 1 at iteration step 468.

Next, we use the 2D Gabor function to show the function approximation capability of PSN with penalty. The 2D Gabor function has the following form (Shin & Ghosh, 1992):

$$h(x, y) = \frac{1}{2\pi(0.5)^2} \cdot e^{\left(-\frac{x^2+y^2}{2(0.5)^2}\right)} \cdot \cos(2\pi(x + y)).$$

Thirty-six input points are selected from an evenly spaced 6×6 grid on $-0.5 \le x \le 0.5$ and $-0.5 \le y \le 0.5$. Eighteen input points are randomly selected from the 36 points as the training set. The remaining 18 points are used for testing the generalization. The number of neurons for the input, summation, and product layers are $P = 3,\ N = 6$ and 1, respectively. The parameters in this example take the values $\eta = 0.2,\ \gamma = 0.06$, and $\lambda = 15$, and the initial weights are selected randomly within the interval $[-2,\ 2]$.

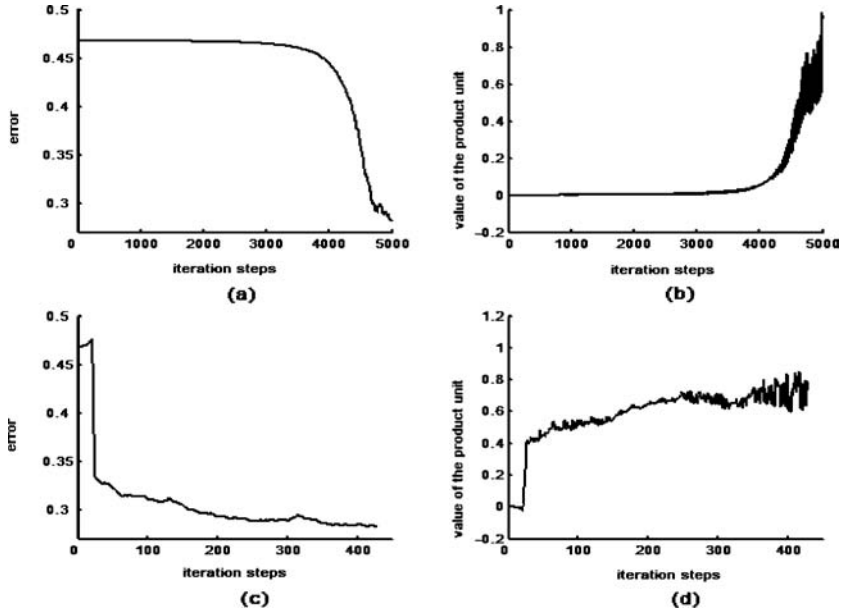

Figure 3: Performances of online gradient algorithm (a, b) without penalty and (c, d) with penalty.

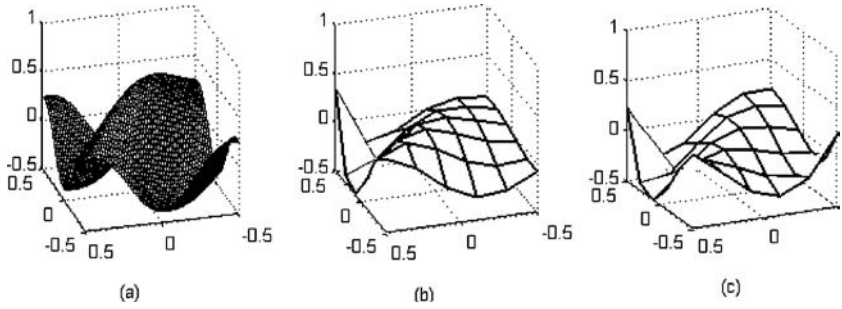

Figure 4: The approximation results for Gabor function: (a) Gabor function; (b) actual network outputs for PSN approximation with penalty; (c) actual network outputs for PSN approximation without penalty.

The output activation function is $g(x) = \tanh(x)$. The errors of PSN with penalty for training and testing inputs achieve 0.098 and 0.89 after 2351 epochs, while those of PSN without penalty are 0.1 and 0.88 after 4673 epochs. Figure 4 shows the Gabor function and the actual network outputs.

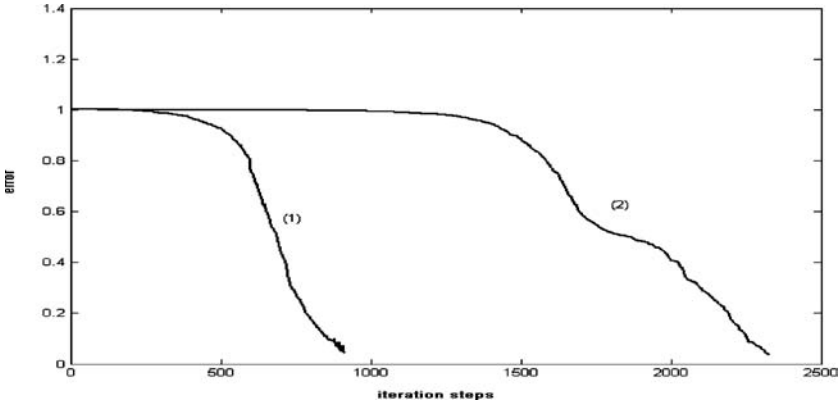

Figure 5: Performances of the online gradient algorithm (1) with penalty and (2) without penalty.

Again we see from the experiment that our penalty algorithm can expedite the training of PSN.

4.2 Example 2: K-Dimensional Parity Problems. The K-dimensional parity problem is a difficult classification problem. There are 2^K-input patterns in the K-dimensional vector space. The values of the components of the input patterns are either $+1$ or -1. The output is 1 when the input pattern contains an odd number of $+1'$s and is -1 otherwise. The famous XOR problem is just the two-dimensional parity problem. In these experiments, the output activation function is $g(x) = \frac{1}{1+e^{-x}}$.

First, we investigate the behaviors of PSN with or without penalty for the three-dimensional parity problem. The number of neurons for the input, summation, and product layers are $P = 4$, $N = 3$ and 1, respectively. The initial weights are selected randomly within the interval $[-0.15,\ 0.15]$. The parameters in this example are $\eta = 0.2$, $\gamma = 0.05$, and $\lambda = 10$. The result is shown in Figure 5.

Next, we investigate the behaviors of the two approaches, using either a large learning rate or a penalty respectively, for the four-dimension parity problem. The number of neurons for the input, summation and product layers are $P = 5$, $N = 4$ and 1, respectively. The parameters are $w \in [-0.2, 0.2]$, $\gamma = 0.05$, and $\lambda = 10$. Ten tests are carried out with different parameters. It is clear from Figure 6 that the penalty approach has better convergence and stability as $\eta \in [0.3, 0.5]$ than the large learning rate approach as $\eta \in [1.0, 1.4]$.

Finally, we choose different values of λ and γ to test their influence on the performance of the online gradient algorithm with a penalty for the four-dimension parity problem. Ten tests are carried out with different values of

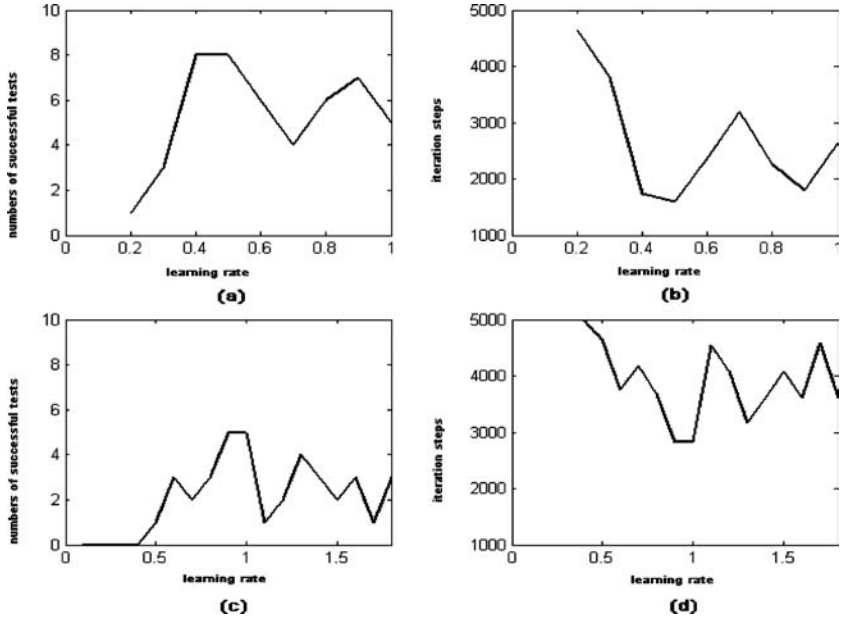

Figure 6: Performances of the online gradient algorithm (a, b) without penalty and (c, d) with penalty.

Table 1: Number of Successful Tests in 10 Tests.

	$\lambda = 4$	$\lambda = 8$	$\lambda = 12$	$\lambda = 16$	$\lambda = 20$	$\lambda = 24$	$\lambda = 28$	$\lambda = 32$	$\lambda = 36$
$\gamma = 0.01$	6	6	1	6	4	8	4	6	2
$\gamma = 0.02$	4	6	4	6	5	6	5	4	4
$\gamma = 0.03$	1	9	5	6	6	5	5	3	4
$\gamma = 0.04$	6	6	5	5	7	4	3	6	5
$\gamma = 0.05$	4	7	6	5	6	8	3	3	2
$\gamma = 0.06$	5	3	5	5	4	2	3	2	2
$\gamma = 0.07$	6	5	5	5	4	5	3	5	3
$\gamma = 0.08$	9	5	5	3	5	1	0	1	1
$\gamma = 0.09$	7	4	1	3	1	3	1	2	1

λ and γ. The number of the successful tests—namely, the tests that stop by criterion 1—in the 10 tests are given in Table 1. The average stopping steps of the 10 tests are showed in Table 2. We observe that λ in [8, 24] and γ in [0.01, 0.05] give better convergence and stability.

Table 2: Average Iteration Steps of 10 Tests.

	$\lambda = 4$	$\lambda = 8$	$\lambda = 12$	$\lambda = 16$	$\lambda = 20$	$\lambda = 24$	$\lambda = 28$	$\lambda = 32$	$\lambda = 36$
$\gamma = 0.01$	3124	3447	4808	3893	3871	3051	4459	3539	4428
$\gamma = 0.02$	3976	3439	3961	3350	2562	3864	3405	3598	3734
$\gamma = 0.03$	4859	2834	3506	3038	3131	3175	4066	4242	3615
$\gamma = 0.04$	3700	3663	3660	3869	3725	2732	4111	3036	3173
$\gamma = 0.05$	3807	3123	3133	3253	2516	3168	3959	3909	4449
$\gamma = 0.06$	3625	4030	3367	3475	3456	4245	3903	4173	4248
$\gamma = 0.07$	3596	3393	3379	3252	3677	3743	3986	3116	3812
$\gamma = 0.08$	2886	3235	3788	4100	3429	4566	5000	4563	4575
$\gamma = 0.09$	3253	3559	4558	3952	4697	4554	4693	4491	4534

Acknowledgments

This work was partly supported by the National Natural Science Foundation of China (10471017).

References

Durbin, R., & Rumelhart, D. (1989). Product units: A computationally powerful and biologically plausible extension to backpropagation networks. *Neural Computation, 1,* 133–142.

Giles, C. L., & Maxwell, T. (1987). Learning invariance and generalization in higher-order neural networks. *Applied Optics, 26*(23), 4972–4978.

Hussaina, A. J., & Liatsisb, P. (2002). Recurrent pi-sigma networks for DPCM image coding. *Neurocomputing, 55,* 363–382.

Jiang, L. J., Xu, F., & Piao, S. R. (2005). Application of pi-sigma neural network to real-time classification of seafloor sediments. *Applied Acoustics, 24,* 346–350.

Li, C. K. (2002). Memory-based sigma-pi-sigma neural network. *International Conference on Systems, Man and Cybernetics, 4,* 1–6.

Li, C. K. (2003). A sigma-pi-sigma neural network (SPSNN). *Neural Processing Letters, 17,* 1–19.

Shin, Y., & Ghosh, J. (1991). The pi-sigma network: An efficient higher-order neural network for pattern classification and function approximation. *International Joint Conference on Neural Networks, 1,* 13–18.

Shin, Y., & Ghosh, J. (1992). Approximation of multivariate functions using ridge polynomial networks. In *Proceedings of International Joint Conference on Neural Networks* (Vol. 2, pp. 380–385). Piscataway, NJ: IEEE Press.

Shin, Y., & Ghosh, J. (1995). Ridge polynomial networks. *IEEE Transactions on Neural Networks, 6*(3), 610–622.

Sinha, M., Kumar, K., & Kalra, P. K. (2000). Some new neural network architectures with improved learning schemes. *Soft Computing, 4*(4), 214–223.

Voutriaridis, C., Boutalis, Y. S., & Mertzios, B. G. (2003). Ridge polynomial networks in pattern recognition. In *4th EURASIP Conference Focused on Video I Image*

Processing and Multimedia Communications (Vol. 2, pp. 519–524). Piscataway, NJ: IEEE Press.

Xin, L. (2002). Interpolation by ridge polynomials and its application in neural networks. *Journal of Computational and Applied Mathematics, 144*, 197–209.

Received July 21, 2006; accepted January 24, 2007.