Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

A Sensitivity-Based Training Algorithm with Architecture Adjusting for Madalines

Yanjun Liu, Xiaoqin Zeng and Shuiming Zhong
Institute of Pattern Recognition and Intelligent System
Hohai University
Nangjing, P R China
{ liuyanjun, xzeng, zhongyi}@hhu.edu.cn

Shengli Wu
School of Computing and Mathematics
University of Ulster
United Kingdom
s.wu1@ulster.ac.uk

***Abstract*—How to design proper architectures of neural networks for solving given problems is an important issue in neural network research. Nowadays, the existing training algorithms of neural networks only focus on adjusting neural networks' weights to improve training accuracy, and few of them adaptively adjust the networks' architecture. However, the architecture is indeed very critical for training neural networks to have high performance and needs to be coped with in the training process. In this paper, we present a new training algorithm of Madalines, which takes not only weight but also architecture adjusting into consideration. The algorithm can thus train Madalines with smaller architecture and higher generalization ability. Experimental results have demonstrated that our algorithm is effective.**

***Keywords*—Neural network, Madaline, training algorithm, architecture, sensitivity**

I. INTRODUCTION

Although the research on artificial neural networks has made great progress and been widely applied to many fields, like pattern recognition, intelligent control, data mining and so on, we still face a series of problems in designing and training neural networks, such as architecture design, improvement on training accuracy and avoidance of overtraining, which are closely related to one another and could seriously influence the performance of trained neural networks. The existing training algorithms usually aim at improving training accuracy by only adjusting the weight without considering the architecture. What is a proper architecture of a neural network for a given application? Unfortunately, the answer to this question is in general not known. On the one hand, a network with a larger size may be trained quickly and fit training data accurately. But, it may cost more in implementation and computation and have bad performance in generalization due to over fitting training data. On the other hand, a network with a smaller size may cost less in both implementation and computation and further may have good performance in generalization. But, it may learn very slowly or even be unable to learn at all. This paper discusses how to involve architecture adjusting into the training algorithm of Madalines so that it can train a Madaline not only satisfying the required training accuracy but also having smaller architecture and higher generalization performance.

A Madaline is a discrete feedforward multilayer neural network with supervised learning mechanism, which is suitable for handling many of the inherently discrete tasks, such as logical calculation, signal processing and pattern recognition etc. Furthermore, its discrete feature can facilitate hardware implementation with less cost, reduce computation complexity, and be computationally simple to understand and interpret. Theoretically, a Madaline can be regarded as a special case of continuous feedforward multiplayer neural networks. It is well known that continuous feedforward multiplayer neural networks are the most mature in techniques, for example the well-known back-propagation training algorithm has been proposed. Unfortunately, most of the techniques can't be directly applied to Madalines due to their hard-limit activation function that is not differentiable. In literature, there are some studies on Madalines' training algorithm. The most popular one is the MRII algorithm [1, 2] proposed by Winter and Widrow in 1988, which trains a Madaline by iteratively adapting the weights of some neurons according to the Minimal Disturbance Principle. But the algorithm's success rate is very low and this hampers its application.

The contribution of this paper is that it proposes a new training algorithm of Madalines, which adjusts not only the weight but also the architecture by employing a sensitivity measure of the networks' neurons. The algorithm automatically search for both proper weight and as small as possible architecture so as to find a suitable Madaline for the given training data. The trained Madaline can satisfy required training accuracy and meanwhile has higher generalization performance. Since by now there has been no practical way to determine Madalines' architecture, especially the number of hidden neurons, it is of great significance that the algorithm can release Madaline users from the burden of designing the network's architecture and simply requires the users to only regard the network as a black box. Some computer simulations were run to verify the effectiveness of the algorithm. The experimental results show that the algorithm can, under a given training accuracy requirement and a casually given architecture, train Madalines with smaller architecture and higher generalization performance.

The rest of this paper is arranged as follows. Section II briefly introduces the Madaline's model and the notations used in the following sections. Section III discusses the technical

978-1-4244-2794-9/09/$25.00 ©2009 IEEE

essentials which are the sensitivity measure of Adalines, the improved training procedure based on MRII with only weight adjusting, and architecture pruning. Then in Section IV we present our training algorithm by assembling the essentials. The experimental verifications are presented in Secction V. Section VI finally concludes the paper.

II. THE MADALINE MODEL AND NOTATIONS

The Madaline considered in this paper is a kind of feedforward multilayered neural network that employs a supervised training mechanism to establish a mapping between its input and output. An Adaline (Adaptive linear elements) is a basic building block of Madalines with discrete input and output. Each element of the input takes on a bipolar value of either +1 or -1 and is associated with an adjustable weight of real number. The working process of an Adaline is that the summation of weighted input elements plus a bias is computed first, producing an analog value, which is then fed into an uncontinuous activation function to yield a bipolar output. The activation function adopted in our research is the commonly used symmetrical hard-limit function:

$$f(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases} \quad (1)$$

Generally, a Madaline may have L $(L \geq 1)$ layers, and each layer l $(1 \leq l \leq L)$ has n^l $(n^l \geq 1)$ Adalines. The form $n^0 - n^1 - \ldots - n^L$ is used to represent a Madaline with a given architectural configuration, in which each n^l $(0 \leq l \leq L)$ not only stands for a layer from left to right including the input layer, but also indicates the number of Adalines in the layer. n^0 is an exception, which refers to the dimension of input vectors. n^L refers to the output layer. Links in a Madaline only exist between Adalines of two adjacent layers, and there is no link between Adalines in the same layer and in any two non-adjacent layers. All Adalines in a layer are fully linked from all the Adalines in the immediately preceding layer and to all the Adalines in the immediately succeeding layer. Since the number of Adalines in layer $l-1$ is equal to the output dimension of that layer, which is in turn equal to the input dimension of layer l, the input dimension of layer l is n^{l-1}. For Adaline i in layer l $(1 \leq i \leq n^l)$, its input vector is $X^l = (x_1^l, \ldots, x_{n^{l-1}}^l)^T$, its incoming weight vector is $W_i^l = (w_{i1}^l, \ldots, w_{in^{l-1}}^l)^T$, its bias is b_i^l, its output is $y_i^l = f(X^l W_i^l + b_i^l)$, and its outgoing weight vector is $V_i^i = (v_{il}^l, \ldots, v_{in^{l+1}}^l)^T$. For layer l, all its Adalines have the same input vector X^l that is the output of immediately preceding layer, its incoming weight set is $W^l = \{W_1^l, \ldots, W_{n^l}^l\}$, outgoing weight set is $V^l = \{V_1^l, \ldots, V_{n^l}^l\}$ $(l < L)$, and output vector is $Y^l = (y_1^l, \ldots, y_{n^l}^l)^T$, which is the input of its immediately succeeding layer. For an entire Madaline, the input vector is X^1 or Y^0, its weight is $W = W^1 \cup \ldots \cup W^L$, and its output is Y^L. Let $\Delta X^l = (\Delta x_1^l, \ldots, \Delta x_{n^{l-1}}^l)^T$ and $\Delta W_i^l = (\Delta w_{i1}^l, \ldots, \Delta w_{in^{l-1}}^l)^T$ be the variation of input and weight vectors at layer l, and $X'^l = (x'^l_1, \ldots, x'^l_{n^{l-1}})^T$ and $W'^l_i = (w'^l_{i1}, \ldots, w'^l_{in^{l-1}})^T$ be the corresponding varied input and weight vectors respectively.

III. THE TECHNICAL ESSENTIALS OF THE TRAINING ALGORITHM

A. The sensitivity measures of Adalines

Usually, the sensitivity measure of a Madaline reflects the effects of its parameters' variation on its output [3, 4]. In Madalines' training, it is expected that an adaptation of an Adaline's weights can result in a change of the Adaline's output and then a change of the corresponding Madaline's output. Therefore, the sensitivity of Adalines with respect to weight variation can be a useful measure for selecting appropriate Adaline for weight adaptation during the training process. Taking all possible inputs into consideration, we adopt the following definition for the sensitivity of Adalines to weight variation.

Definition: The sensitivity of an Adaline is defined as the probability of output inversions of the Adaline due to its weight variation with respect to all inputs, which is expressed as:

$$s(\Delta W, W, b) = E_X\left(\frac{1}{2}\left|f(X(W+\Delta W)+b) - f(XW+b)\right|\right) \quad (2)$$

In (2), for simplicity, the superscript that marks the Adaline's layer and the subscript that marks the Adaline's order in a layer are omitted because the Adaline's position in the network can be ignored without losing generality. In a similar way, the sensitivity of Adalines with respect to input variation can be defined and expressed as:

$$s(\Delta X, W, b) = E_X\left(\frac{1}{2}\left|f((X+\Delta X)W+b) - f(XW+b)\right|\right) \quad (3)$$

(3) reflects the variation degree of the Adaline's output due to its input variation, and will be employed as the basis measure in the architecture pruning process.

[5] has given an algorithm to compute (2), in which the sensitivity is quantified by establishing a geometric model of hypercube and using analytical geometry and tree techniques. For the details of the algorithm, please refer to [5].

Obviously, the difference between (2) and (3) is that they deal with different parameters' variation. The former focuses on weight variation while the latter focuses on input variation. Although (3) is different from (2) in variation, it can be easily transformed from (3) to (2), namely from input variation to weight variation. Because of the bipolar feature of Adalines' input, the variation of an Adaline's input element can only

result in either $x'_j = x_j$ or $x'_j = -x_j$ Therefore, an affected product in summation $\sum_{j=1}^{n} x'_j w_j$ can be expressed as $x'_j w_j = (-x_j) w_j = x_j(-w_j) = x_j w'_j$, this means that x'_j can easily be transformed to w'_j, which is equivalent to a change of the sign of w_j. For this reason, the Adaline's sensitivity with respect to input variation can be attributed to the Adaline's sensitivity with respect to weight variation. So, the algorithm in [5] can also be used to compute (3).

B. The improved training algorithm based on MRII

In [1] and [2], Winter and Widrow presented the MRII algorithm for training Madalines. Different from the steepest descent rule, MRII is one of error correction learning rules. In MRII, the central idea is the Minimal Disturbance Principle which means that the established mapping for previously trained input samples should be disturbed as little as possible when a correction to the network is needed. The absolute value of summation of weighted input is used in MRII as one of the main factor of confidence level, which is a kind of measure to carry out the principle.

How to choose the Adaline to update its weights is one of the crucial issues in Madaline training algorithms. Since the Adaline that confidence level is closest to zero is easier to be adjusted to reverse its output, Winter and Widrow proposed the Adaline's confidence level as a suitable measure for the Minimal Disturbance Principle. However we think that the Adaline's confidence level only satisfies the Minimal Disturbance Principle for current input sample and can't reflect disturbance degree for other trained input samples. According to the definition of the sensitivity of Adalines, the sensitivity measure indicates the effect of weight variation on the input-output mapping with respect to all possible input patterns. It means that if an Adaline has the least sensitivity, its output for all input patterns will be varied least with a given weight variation. So it is more suitable for replacing the absolute value of summation of weighted input with the sensitivity to meet the Minimal Disturbance Principle.

Another crucial issue is weight adaptation rule. In the family of the error correction training algorithms, there are many weight adaptation rules, such as the Perceptron rule, the Mays rule, and the LMS rule etc. In our training algorithm, we use the weight adaptation rule introduced in [2], which comes from the Mays' rule, as follows:

$$\Delta W = W(k+1) - W(k) = \frac{\eta d(k)}{n+1} X[L - d(k) X W(k)] \quad (4)$$

Where $W(k)$ is the weight obtained at the k^{th} iteration of adaptation and $W(k+1)$ is the weight after the $k+1^{th}$ iteration of adaptation, $d(k)$ is the desired response, n is the input dimension of Adaline, η is the adaptation constant and L is the adaptation level. If L and η are set to be 1, the output of the Adaline can be reversed in one iteration step.

It is straightforward for us to compute weight variation, namely ΔW, of Adalines in a layer according to (4), and then compute their sensitivity values with the algorithm given in [3]. After that, confidence levels of Adalines can be obtained and are used to sort the Adalines in the layer.

Summarizing the descriptions above, the improved training algorithm based on Adaline's senstivity can be programmed as follows.

1) *Randomly select a sample from the training data set;*
2) *If the Madaline responses correctly to the current sample, go to step 4;*
3) *From the first layer to the last layer, do:*
   a) *Obtain weight variations that cause output inversion of the Adalines in current layer by (4), and compute their sensitivity and confidence level values;*
   b) *Sort the Adalines of current layer according to their confidence level values;*
   c) *For all possible k-wise (k is from 1 to* $\min\left\{3, \left\lceil \frac{n^l}{2} \right\rceil\right\}$*) trials in the* $\left\lceil \frac{n^l}{2} \right\rceil$ *least confidence level do:*

      If the trial can reduce the output errors of the Madaline, accept the trail, adapt weights of the Adalines involved in the trial, and go to step 2; otherwise reject the trial, restore the outputs of the Adalines involved in the trial to their previous values, and continue to do the next trial;

   d) *Continue the loop to deal with the next layer.*
4) *Go to step 1 unless training accuracy meets the given requirement for all training samples or training epochs meet the given number.*
5) *Output obtained weights and biases, and stop.*

C. The architecture pruning

As mentioned previously, a neural network with smaller size of architecture can have many advantages. So it is desired in the training process to find a network that is not only trainable with given training data and required training accuracy but also has as small as possible size of architecture. In our research, we merge an architecture pruning procedure into the above training algorithm. The key question in architecture pruning is how to locate the least relevant Adalines in a well trained Madaline so that the pruning of them will cause as little performance loss as possible and be easy to compensate for the loss. One reasonable answer to the question is that a relative relevance measure of Adalines needs to be established. Once more, the sensitivity of Adalines can be employed for this purpose.

The sensitivity can reflect an Adaline's response to its input variation. Under a given input variation, the Adaline with sensitivity being zero or a very small value contributes less in the entire network since its output is approximately constant to the variation in its input. But, due to the sensitivity only having relation to the Adaline's incoming weights and bearing no relation to its outgoing weights, the sensitivity itself is not

enough to reflect the influence of the Adaline on its immediately succeeding layer. It is noticed that the outgoing weights may also play an important role in determining the input of the Adalines in the succeeding layer. Even if the sensitivity is very small, it may be amplified by large magnitude of the outgoing weights and thus cause a large variation to the input of the succeeding Adalines. In order to overcome this shortcoming, [6] takes both the sensitivity and the outgoing weights into consideration to establish a relevance measure for an Adaline. The relevance of an Adaline is defined as follows.

Definition: The relevance of an Adaline is defined as the multiplication of its sensitivity by the summation of the absolute values of its outgoing weights, which is expressed as:

$$r_i^l = \bar{s}_i^l * \sum_{j=1}^{n^{l+1}} \left| v_{ij}^l \right|. \tag{5}$$

Where v_{ij}^l is an element of outgoing weight V_i^l; and $\bar{s}_i^l$ is the sensitivity, which is an average of those sensitivity values that are obtained by iteratively computing the sensitivity with only one input element varied once for all input elements. In this way we could restrict the sensitivity with input varying on a small scale because of the bipolar feature of the input elements.

Obviously, the smaller the value of r_i^l is, the less variable the inputs of the succeeding Adalines are. Hence, the relevance measure is more accurate than the sensitivity in reflecting the effect of an Adaline on its succeeding Adalines. With the relevance measure it is available for locating the least relevant Adaline in a hidden layer.

The removal of an Adaline will more or less cause a change in the performance of the Madaline. In order to avoid the loss as much as possible in the established performance, it is necessary to consider some compensations for the loss. One way is to adjust the biases of the Adalines in the immediately succeeding layer. If the average output of the given Adaline is $\bar{y}_i^l$, which can be approximately calculated with training samples, then for each Adaline, j $(1 \le j \le n^{l+1})$, in the next layer, $l+1$, its bias could be adjusted by

$$b_j^{l+1} = b_j^{l+1} + \bar{y}_i^l * v_{ij}^l. \tag{6}$$

Another way is to retrain the pruned Madaline, which will be discussed in the next section. For more detailed description of the pruning techniques, please refer to [6].

IV. The Training Algorithm

In this section, we assemble the technical essentials discussed in the last section to program an algorithm for training Madalines. The algorithm merges the architecture pruning into the improved training algorithm. For a given application, the input dimension and the number of Adalines of output layer could be determined with the application domain knowledge. According to the Stone-Weierstrass theorem [7], if the number of hidden Adalines is large enough, only one hidden layer is needed for a Madaline. So the algorithm only needs to consider automatically searching for proper number of hidden Adalines without the need of considering the number of hidden layers. Under a given training data set and required training accuracy as well as a casually given architecture, the algorithm will finally yields a Madaline with the smallest possible architecture and suitable weights. Below is the training algorithm:

1) *Organize a Madaline with the preliminarily given architecture and randomly assign the initial weights and biases;*
2) *Use the given training data set and the algorithm of Section III's part B to train the Madaline;*
3) *If the Madaline in training can meet the required training accuracy, then go to step 6 to prune it;*
4) *If there is saved Madaline, then restore the last saved Madaline and go to step 8 to stop;*
5) *Increase the number of hidden Adalines with random weights and go to step 2;*
6) *Save the trained Madaline, such as its architecture, weights and biases, compute all hidden Adalines' sensitivity and relevance values, and remove the Adaline with the least relevance value;*
7) *Adjust the bias of each Adaline in the next layer of the pruned Madaline, and go to step 2 to retrain the pruned Madaline.*
8) *Output finally obtained architecture, weights and biases, and stop.*

V. Experimental Verifications

This section presents some experiments that were carried out to verify the effectiveness of the algorithm given in the last section. In our experiments, four representative problems were chosen. One is the XOR problem; another is the implementation of a logical function; the third is a classification problem from UCI repository [8]; and the fourth is an emulation problem. Four tables, each of which corresponding to one of the four problems, are given to show the training performance of the algorithm with architecture adjusting in comparison with that of the algorithm without architecture adjusting.

In the experiments, we selected the initial weight and bias randomly, set the training accuracy to be 100 percent to meet the given training data, and forced the training epochs not to be more than 1000. In the following tables, we define the success rate as the proportion of the convergent Madalines to all the Madalines involved, and the generalization performance is the correctness rate of a convergent Madaline on the testing data set. For each Madaline in training, we trained it 20 times with the same initial architecture but different random weights and biases. The presented results are the averages of the 20 runs' results.

As a matter of convenience, in the tables of this section, we abbreviate the algorithm without architecture adjustment to Algorithm I, the algorithm with architecture adjustment to Algorithm II, the generalization performance of a trained

Madaline to Gen, and the initially given and finally trained architecture to Initial Arc and Final Arc**.**

Experiment A:

In this part, the experimental results for solving the XOR problem are presented in Table I. In the experiment, 20 Madalines with an initial architecture of 2-3-1 are trained by both Algorithm I and Algorithm II. Table I shows that Algorithm II can achieve 100% success rate (30% succeeded in the architecture of 2-2-1 and 70% in 2-3-1) while Algorithm I can only achieve 90% success rate. Further more, Algorithm II can yield smaller size of Madalines with architecture of 2-2-1, but Algorithm I can't.

TABLE I. EXPERIMENTAL RESULTS FOR THE XOR PROBLEM

Initial Arc	Algorithm I	Algorithm II	
	Success rate	*Final Arc*	*Success rate*
2-3-1	90%	2-2-1	30%
		2-3-1	70%

Experiment B:

This experiment involves implementing a Madaline to realize the following logical function: $F=(a\vee b)\wedge(c\vee d\vee e)$. Since each logical variable in the expression has bipolar value of +1 or –1, there are altogether 2^5 different input samples. In the experiment, 24 samples were randomly selected as training samples and the left 8 are reserved as testing samples. The training starts with initial architecture of 5-2-1 and 5-3-1. The results of success rate and generalization performance are showed in Table II. From the table, we can see that when the initial architecture is 5-3-1, both the two algorithms can succeed for all the Madalines organized, but the generalization performance of Madalines trained by Algorithm II is most of the time better than that of Algorithm I, and 65% Madalines' architecture can be pruned to 5-2-1. When the initial architecture is 5-2-1, the success rate of Algorithm I declines to 65%, but the Algorithm II can still achieve 100% success rate (90% succeeded in 5-2-1 and 10% succeeded in 5-3-1).

TABLE II. EXPERIMENTAL RESULTS FOR THE LOGICAL FUNCTION FROBLEM

Initial Arc	Algorithm I		Algorithm II		
	Success rate	*Gen*	*Final Arc*	*Success rate*	*Gen*
5-2-1	65%	81.73%	5-2-1	90%	84.72%
			5-3-1	10%	100%
5-3-1	100%	83.75%	5-2-1	65%	89.42%
			5-3-1	35%	85.71%

Experiment C:

In this experiment, we adopted the benchmark data of the Monk's problem from UCI repository [8]. According to the problem's attribute information, except the Id attribute that is unique for each sample, the Monk's problem has seven attributes including an output attribute indicating two classes and six input attributes. Among the input attributes, four of them have three or four possible values, and two of them have two possible values. Since Madalines' input elements are bipolar, we need to use two input elements to represent each attribute with three or four values and one input element to represent each two-valued attribute. So, the experimental networks for the Monk's problem have a 10-dimensional input and 1-dimensional output. We organized Madalines with initial architecture of 10-4-1 and 10-2-1. The results of the success rate and the generalization performance of trained Madalines are showed in Table III. From the table, we can see that when the initial architecture is 10-4-1, all the Madalines can be trained by the two algorithms successfully, but Algorithm II can prune all the Madalines' architecture to 10-3-1 and the generalization performance of the Madalines trained by it is better than that by the Algorithm I. When the initial architecture is 10-2-1, no Madaline can be successfully trained by Algorithm I, but all Madalines can still be trained in architecture 10-3-1 by Algorithm II.

TABLE III. EXPERIMENTAL RESULTS FOR THE MONK1 PROBLEM

Initial Arc	Algorithm I		Algorithm II		
	Success rate	*Gen*	*Final Arc*	*Success rate*	*Gen*
10-2-1	0	-	10-3-1	100%	92.29%
10-4-1	100%	91.15%	10-3-1	100%	92.97%

Experiment D:

In order to further verify the effectiveness of the algorithm with architecture adjusting, we performed some experiments on emulation problem. In the experiments, we set a reference Madaline of 12-3-1 with randomly assigned weights as a teacher, and then obtained the training dataset and testing dataset by randomly selecting possible input patterns, 512 patterns for training and 256 patterns for testing, and their corresponding ideal outputs from the reference Madaline. We organized Madalines with initial architecture of 12-4-1 and 12-3-1. Table IV shows the experimental results. From the table, we can see that when the initial architecture is 12-4-1, the success rate of Algorithm I is 85%, but all the Madalines can be successfully trained by Algorithm II, and 80% of them even have a smaller architecture of 12-3-1, so their generalization performance is also be improved. When the initial architecture is 12-3-1, the success rate of Algorithm I declines to 55%, but Algorithm II can still achieve 100% success rate (95% succeed in 12-3-1, and 5% succeed in 12-4-1).

TABLE IV. EXPERIMENTAL RESULTS FOR THE EMULATION PROBLEM

Initial Arc	Algorithm I		Algorithm II		
	Success rate	*Gen*	*Final Arc*	*Success rate*	*Gen*
12-3-1	55%	94.07%	12-3-1	95%	94.20%
			12-4-1	5%	95.31%
12-4-1	85%	93.26%	12-3-1	80%	94.34%
			12-4-1	15%	94.27%
			12-5-1	5%	96.48%

VI. CONCLUSION

In this paper, a new training algorithm of Madalines is proposed, which is built on the basis of a sensitivity measure of Adalines and Madaline architecture pruning techniques. By introducing architecture adjusting into the training process, the algorithm can guarantee to train a Madaline to realize the mapping implied in given training data set and meanwhile have as small as possible architecture.

The experimental results indicate that our algorithm can make the following improvements over the previous Madaline's training algorithms:

- It can greatly improve the training success rate of Madalines, which is very low for other training algorithms of Madalines;
- It can automatically find a suitable architecture for the Madaline during the training process;
- It can improve the generalization performance of the trained Madaline.

There are still some parts in the algorithm that need to be further studied. One is how to determine a proper weight variation to compute the sensitivity value. In the algorithm we used winter's rule [2] to obtain the weight variation, which can make the output reverse in one iteration step so as to reduce the cost of computation. But it does not mean that thus obtained weight variation is reasonably small. Consequently, when we update the weight during training, we are not sure it is the least variation of weight to meet the Minimal Disturbance Principle, which is an important fact for the performance improvement of the training algorithm. Another is that the sensitivity of Adalines can only be employed to determine a sequence of trials on the reversions of some Adalines' outputs, but some of the trials may be rejected if their reversions can not reduce the output errors of the Madaline in training. Conceptually, the sensitivity of a Madaline to weight variation directly reflects the effect of the weight adaptation on the network's output, and it may be more helpful than the sensitivity of an Adaline to locate the weight for adaptation, and thus avoid unnecessary trials.

In our future research, we will try to find other adaption methods for quickly getting the optimal weight variation. In addition, we will explore how to employ the sensitivity measure of Madalines to improve the performance of the Madaline training algorithms.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China under grants 60571948 and 60673186.

REFERENCES

[1] R. Winter, and B. Widrow, "Madaline Rule II: A Training Algorithm for Neural Networks," IEEE International Conference on Neural Networks, vol. 1, pp. 401-408, 1988.

[2] R. Winter, "Madaline Rule II: A New Method for Training Networks for Adalines," PhD Dissertation, Stanford University, 1989..

[3] M. Stevenson, R. Winter and B. Widrow, "Sensitivity of feedforward neural networks to weight errors," IEEE Transactions on Neural Networks, vol. 1, no. 1, pp. 71-80, 1990.

[4] Y. Wang, X. Zeng, D. S. Yeung and Z. Peng, "Computation of Madalines' Sensitivity to Input and Weight Perturbations," Neural Computation, vol. 18, no. 11, pp. 2854-2877, 2006.

[5] X. Zeng, Y. Wang, and K. Zhang, "Computation of Adalines' Sensitivity to Weight Perturbation," IEEE Transactions on Neural Networks, vol. 17, no. 2, pp. 515-518, 2006.

[6] X Zeng, J Shao, Y. Wang and S, Zhong, "A sensitivity-based approach for pruning architecture of Madalines," Neural Computing & Applications, published online, 2008.

[7] N. E. Cotter, "The Stone-Weierstrass Theorem and Its Application to Neural Networks", IEEE Transactions on Neural Networks, vol. 1, no. 4, pp. 290-295 1990.

[8] http://www.ics.uci.edu/~mlearn/MLRepository.html