MIXED-MODE LEARNING: A METHOD FOR REDUCING THE NUMBER OF HIDDEN UNITS IN CASCADE CORRELATION

Chin-Chi Teng and Benjamin W. Wah
Center for Reliable and High Performance Computing
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 West Main Street, Urbana, Illinois 61801, USA
wah@manip.crhc.uiuc.edu

Abstract

In this paper, we present a new learning mechanism called mixed-mode learning. This learning mechanism transforms an existing supervising learning algorithm from one that finds a unique one-to-one mapping into an algorithm that finds one of the many one-to-many mappings. Since the objective of learning is relaxed by this transformation, the number of learning epochs can be reduced significantly. We show in this paper that mixed-mode learning can be applied in the well-known cascade correlation learning algorithm to reduce the number of hidden units required for convergence when applied to classification problems whose desired outputs are binary. Our experimental results confirm this reduction, although they show that more learning time is required than that of cascade correlation. In general, reducing the number of hidden units at the expense of additional learning time is usually desirable.

1. Introduction

The *back-propagation learning algorithm* (*BP*) [4] and subsequent improvements are elegant supervised learning algorithms for artificial neural networks (ANNs). These algorithms start with an assigned network configuration and assumed training parameters, and adjust the weights of the configuration by gradient-descent techniques. The major drawback of these algorithms, however, is their long and unpredictable training times.

To cope with the problem of long learning times, Fahlman and Lebiere developed the *cascade-correlation learning algorithm* (*CAS*) [3]. This algorithm differs in many ways from back-propagation. It begins with a minimal network and automatically trains and adds new hidden units one by one to create a multilayered network. The hidden units are trained to maximize the correlation between output-unit values and output-unit errors. Once a new hidden unit has been trained and added to the network, weights connected to the input layer are frozen. Quickprop [2] is then used to update weights connected to the output units. CAS generally requires much shorter time for convergence than that of BP.

During learning, CAS uses a variable network topology, and the number of hidden units when the algorithm terminates is not bounded. In general, increasing the number of hidden units increases the approximation quality of the network with respect to its training patterns, but not always improves its generalization behavior. Reducing the number of hidden units required for convergence is one way to improve its generalization behavior. An additional benefit of smaller networks is that they are faster when used in an application.

In this paper, we present *mixed-mode learning* (*MM*) [1] for improving CAS when applied to classification problems with binary outputs. MM transforms a learning problem from that of finding a one-to-one mapping into one that finds one-to-many mappings. Since the objective of learning is relaxed, it needs less training epochs than the original learning algorithm. For CAS, a reduction in the number of learning epochs usually leads to a reduction in the number of hidden units required for convergence. This is true because the number of units requireded for

Research supported by National Science Foundation Grant MIP 92-18715 and Joint Services Electronics Program Contract JSEP N00014-90-J-1270.

Proceedings of International Symposium on Artificial Neural Networks, December 20-22, 1993, National Chiao Tung University, Hsinchu, Taiwan, ROC

convergence in CAS is a monotonically nondecreasing function of the number of learning epochs.

This paper is organized as follows. Section 2 presents a non-iterative learning algorithm based on linear programming for training single-layer neural networks. Using this algorithm, Section 3 presents MM, our mixed-mode learning mechanism. We then present in Section 4 the application of MM in CAS. Section 5 shows our experimental results, and conclusions are drawn in Section 6.

2. Supervised Learning of a Single-Layer Neural Network as a Linear Programming Problem

An application suitable for supervised learning can be modeled as a mapping of an *input pattern matrix P* (with *k* patterns, each with *m* values) into an *output pattern matrix D* (with *k* patterns, each with *n* values). *P* is, therefore, a *k*-by-*m* matrix, and *D*, a *k*-by-*n* matrix. Let P^T and D^T be the transposes of *P* and *D*. We have

$$P^T = [\, p_0 \; p_1 \; \cdots \; p_{k-1} \,], \quad \text{and} \quad D^T = [\, d_0 \; d_1 \; \cdots \; d_{k-1} \,], \tag{1}$$

where p_i is the *i*-th input pattern

$$p_i = [\, p_{i,0} \; p_{i,1} \; \cdots \; p_{i,m-1} \,],$$

and d_i is the corresponding *i*-th output pattern

$$d_i = [\, d_{i,0} \; d_{i,1} \; \cdots \; d_{i,n-1} \,].$$

The single-layer neural network to be learned performs a mapping from *P* to *D*.

Assume initially that the number of output units is one ($n = 1$). Since the classification problems that we study in this paper have binary outputs, we assume that the *40-20-40 criterion* is applied; that is, an output is considered a logic ONE if it is larger than 0.6, and ZERO if smaller than 0.4.

When the sigmoid function is used as the activation function, the problem of learning the weights of a single-layer single-output neural network is the same as getting a weight matrix $W_{m\times 1}$ where

$$W^T = [\, w_0 \; w_1 \; \cdots \; w_{m-1} \,], \tag{2}$$

such that

$$p_i \cdot W \; \begin{cases} \geq S^{-1}(0.6) & \text{if } d_i = 1 \\ \leq S^{-1}(0.4) & \text{if } d_i = 0 \end{cases}, \quad \text{for all } i = 0,\, 1,\, 2,\, ..,\, k-1, \tag{3}$$

where $S^{-1}(x)$ is the inverse sigmoid function. Since $S^{-1}(0.6) = -S^{-1}(0.4)$, Eq. (3) can be represented in matrix form as follows:

$$\hat{P} \cdot W \geq \begin{bmatrix} S^{-1}(0.6) \\ S^{-1}(0.6) \\ \vdots \\ S^{-1}(0.6) \end{bmatrix}_{k\times 1}, \tag{4}$$

where

$$\hat{P}^T = [\, \hat{p}_0 \; \hat{p}_1 \; \cdots \; \hat{p}_{k-1} \,] \quad \text{and} \quad \hat{p}_i = \begin{cases} p_i & \text{if } d_i = 1 \\ -p_i & \text{if } d_i = 0 \end{cases}. \tag{5}$$

The process of obtaining *W* that satisfies Eq. (4) is very similar to that of finding a feasible solution of a linear program. The only difference is that elements of *W* can be negative, whereas variables in linear programming problems have to be positive. To deal with the problem of unrestricted variables, we transform the elements of *W* into x_i, where

$$x_j = w_j + \eta, \quad \text{such that} \quad x_j, \eta \geq 0, \quad \text{and} \quad j = 0, .., m-1. \tag{6}$$

Hence, Eq. (4) becomes

$$\left[\begin{array}{c|c} & y_0 \\ & y_1 \\ \hat{P} & y_2 \\ & \cdot \\ & \cdot \\ & y_{k-1} \end{array}\right]_{k\times(m+1)} \left[\begin{array}{c} X \\ \hline \eta \end{array}\right]_{(m+1)\times 1} \geq \begin{bmatrix} S^{-1}(0.6) \\ S^{-1}(0.6) \\ \cdot \\ \cdot \\ \cdot \\ S^{-1}(0.6) \end{bmatrix} \tag{7}$$

where

$$X^T = [\, x_0\ x_1\ \cdots\ x_{m-1}\,], \text{ and } y_i = \sum_{j=0}^{k-1} -\hat{p}_{i,j}\,. \tag{8}$$

Since variables x_i and η in Eq. (6) are positive, the values of variables that satisfy Eq. (6) can be obtained by the simplex method. The following example illustrates the procedure for transforming supervised learning into a linear programming problem.

Example 1. Consider a single-layer neural network for solving a problem whose input matrix P and desired output matrix D are

$$P = \begin{bmatrix} -15 & -5 \\ -1 & 1 \\ -1 & -3 \end{bmatrix} \quad \text{and} \quad D = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}. \tag{9}$$

Eq's (3) and (4) are applied to obtain

$$\hat{P} = \begin{bmatrix} -15 & -5 \\ -1 & 1 \\ -1 & -3 \end{bmatrix}, \tag{10}$$

and a set of inequalities:

$$\begin{bmatrix} -15 & -5 & 20 \\ -1 & 1 & 0 \\ -1 & -3 & 4 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ \eta \end{bmatrix} \geq \begin{bmatrix} 0.41 \\ 0.41 \\ 0.41 \end{bmatrix}, \tag{11}$$

where x_0, x_1, $\eta \geq 0$. The simplex method is then applied to obtain the following feasible solution.

$$[\, x_0\ \ x_1\ \ \eta\,]^T = [\, 0.0\ \ 0.41\ \ 0.0\,]^T. \tag{12}$$

The weight matrix W is, therefore,

$$W = \begin{bmatrix} x_0 - \eta \\ x_1 - \eta \end{bmatrix} = \begin{bmatrix} -0.41 \\ 0.0 \end{bmatrix}. \tag{13}$$

We can verify the result by computing

$$D_{real} = sigmoid\,(P \cdot W) = [\, 1.0\ \ 0.6\ \ 0.6\,]^T. \tag{14}$$

In the above example, we assume the number of output units to be one, *i.e.*, D is a column vector. If the number of output units is n (that is, the desired output matrix D is a k-by-n matrix), we can decompose the learning problem into n sub-problems. In each sub-problem, one of the column vectors of D is used to get a matrix $\hat{P}$ by applying Eq. (3). $\hat{P}$ is then used to get the corresponding column vector of weight matrix W.

For the case of one output unit with binary output values, the linear programming formulation will allow a weight matrix W to be found such that $sigmoid(P \cdot W) \approx D$ if and only if

$$R_D^k \cap \text{span}\{P\} \neq \phi, \tag{15}$$

where

$$R_D^k = \{[x_0 \ \cdots \ x_{k-1}]^T \mid sign(x_i) = sign(d_i - 0.5)\}, \quad \text{where } d_i = 0 \text{ or } 1 \text{ for all } i = 0, 1, ..., k-1, \tag{16}$$

and span$\{P\}$ is the m-dimensional space spanned by the k input patterns. In supervised learning, Eq. (15) is seldom satisfied; *i.e.*, Eq. (4) usually has no feasible solution. However, the original learning problem only requires finding a set of weights such that the number of correct output patterns is maximum. Hence, we can change the objective of our simplex formulation to finding a set of values for all variables such that the number of inequalities that are satisfied is maximized. Assume

$$A = \left[\begin{array}{c|c} & y_0 \\ & y_1 \\ \hat{P} & \cdot \\ & \cdot \\ & \cdot \\ & y_{k-1} \end{array}\right]_{k \times (m+1)}, \quad V = \left[\begin{array}{c} X \\ \hline \eta \end{array}\right]_{(m+1) \times 1}, \quad B^T = \left[\begin{array}{c} b_0 \\ \cdot \\ \cdot \\ \cdot \\ b_{k-1} \end{array}\right], \quad \text{and } b_i = S^{-1}(0.6). \tag{17}$$

To obtain weights such that the number of correct output patterns is maximized, we can formulate the corresponding optimization problem as follows.

$$\text{Maximize} \sum_{i=0}^{k-1} u_0 \left(\sum_{j=0}^{m} a_{i,j} v_j - b_i \right) \tag{18}$$

$$\text{such that} \sum_{j=0}^{m} a_{i,j} v_j \ \geq \ b_i$$

$$\text{where } v_j \geq 0, \quad i = 0, .., k-1, \quad j = 0, ..., m,$$

$$\text{and } u_0(\bullet) \text{ is a step function with transition at 0.}$$

The overhead for solving the above nonlinear optimization problem is very high, and the process is, therefore, not practical. To reduce this overhead, we use a heuristic to obtain a proper set of weights when there is no feasible solution. The heuristic is similar to *Phase I* of the simplex method. First, we add a slack variable y_i to every constraint inequality, changing every inequality into an equality as follows:

$$\sum_{j=0}^{m} a_{i,j} v_j - y_i - b_i = 0, \ \text{where } y_i \geq 0, \ \text{and } i = 0, ..., k-1. \tag{19}$$

Next, we attach an artificial variable z_i to each constraint equation, where

$$z_i = b_i + y_i - \left(\sum_{j=0}^{m} a_{i,j} v_j \right) \quad \text{and} \quad z_i \geq 0, \quad i = 0, ..., k-1, \tag{20}$$

in such a way that the set of equalities in Eq. (20) always has a feasible solution. Note that the trivial feasible solution is to set all v_i and y_j to zero. We then solve the following linear optimization problem using the simplex method.

$$\text{Minimize} \sum_{i=0}^{k-1} z_i \tag{21}$$

$$\text{such that } z_i = b_i + y_i - \left(\sum_{j=0}^{m} a_{i,j} v_j \right)$$

$$\text{where } v_j, y_i, z_i \geq 0, \ i = 0, ..., k-1, \ j = 0, ..., m.$$

Since we minimize $\sum z_i$, the optimal solution of Eq. (21) should have a small value for each z_i. Therefore, it is very likely that the inequality $z_i \le y_i$ can be satisfied, and the original constraint inequalities listed in Eq. (18) are satisfied with a large probability. This is true because if $z_i \le y_i$, then

$$\sum_{j=0}^{m} a_{i,j} v_j = y_i + b_i - z_i \ge b_i. \tag{22}$$

Note that the more inequalities in Eq. (18) are satisfied, the more correct output patterns are obtained in the output layer.

3. Mixed-Mode Learning Mechanism

The objective of the *mixed-mode learning mechanism* (MM) is to reduce the number of training epochs. This can be achieved if the output matrix is flexible; that is, learning is faster if there is a large pool of desired output matrices, and learning can stop whenever any one of them is found. By exploiting this property, MM first transforms the original problem of finding a one-to-one mapping from P to D into one that finds a one-to-one mapping from P to one of a large set of possible output matrices I_{real}. It then transforms I_{real} to D by using the non-iterative learning algorithms for single-layer neural networks described in Section 2.

We use Figure 1 to illustrate how MM works. Given an application problem with input matrix P and desired output matrix D, an existing supervised learning algorithm is used to train the original network. During training, a *monitor* is used to extract *intermediate output matrix* I_{real} periodically, and apply the linear programming method described in Section 2 to map I_{real} to D, where I_{real} is the set of output values of input and/or hidden units that are connected to the output units. An element $(I_{real})_{i,j}$ in matrix I_{real} is the output of the j-th unit that is connected to the output layer when the i-th training pattern is applied.

MM requires a smaller number of training epochs since it has a relaxed objective. Learning in MM only tries to get an intermediate output matrix I_{real} that can satisfy

$$R_{d_i}^{k} \cap \text{span}\{I_{real}\} \ne \phi \quad \text{for all } i = 0, 1, ..., k-1 \tag{23}$$

where d_i is the i-th column vector of D. Since there are potentially many I_{real}'s satisfying this criterion, learning involves finding one of the one-to-many mappings and is much easier. The additional overhead incurred by an extra subnet involves solving a feasible solution of a set of linear inequalities.

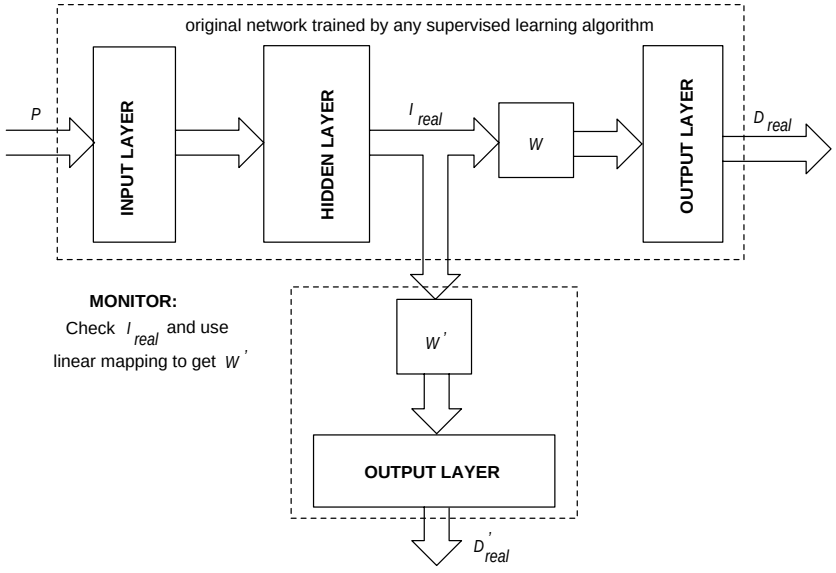

Figure 1. Mixed-mode learning mechanism.

4. Reduction in the Number of Hidden Units for Cascade Correlation

In CAS, the number of hidden units changes in a monotonically non-decreasing fashion as learning progresses. Hence, reducing the number of epochs by MM will lead to an equal or smaller number of hidden units in the resulting network.

There are two training phases in CAS: *TRAIN_INPUT* for adding new hidden units, and *TRAIN_OUTPUT* for training the weights in the output layer. These two phases are executed alternatively. If a monitor is added to CAS, we note (a) that I_{real} cannot be acquired in the *TRAIN_INPUT* phase, as the new hidden unit has not been decided upon, and (b) that I_{real} is frozen in the *TRAIN_OUTPUT* phase. Consequently, we only have to use the monitor in the first epoch of each *TRAIN_OUTPUT* phase, and the monitor is seldom activated when MM is applied in CAS. The detailed procedure for applying MM in CAS is summarized as follows.

Procedure 1. Applying MM in CAS.

Given a training pattern set (including an input matrix P and a desired output matrix D), the procedure has five steps.

1. Train the primary network that includes the input units and the output units by Quickprop. If $\|D_{real} - D\|$ (see Figure 1) is smaller than a prescribed threshold, then stop.
2. Execute the TRAIN_INPUT phase of CAS to add a new hidden unit.
3. Extract I_{real} from the network obtained in Step 2, where I_{real} is the input matrix to the output layer.
4. Using the linear programming method described in Section 2 to obtain the weights of connections to the output layer. If $\|D_{real} - D\|$ (see Figure 1) is smaller than a prescribed threshold, then stop.
5. Execute the TRAIN_OUTPUT phase of CAS. If $\|D_{real} - D\|$ is smaller than a prescribed threshold, then stop; otherwise go to Step 2.

5. Experimental Results

We compare the performance of the original CAS with that of CAS with MM (CAS+MM) using the two-spiral problem as a benchmark. The experiment is repeated with thirty trials, each with different initial weights. Figure 2 shows a plot of normalized number of hidden units versus normalized learning time (CPU seconds). Each point (x, y) in this figure is normalized with respect to the performance of the ''original'' CAS starting with an identical initial configuration and an identical set of random weights. (Using this normalization method, point (1,1) in Figure 2 represents the performance of the ''original'' CAS.) For the set of points generated using MM+CAS, the following normalizations are performed.

$$x = \frac{\text{\# hidden units for net trained by MM+CAS}}{\text{\# hidden units for net trained by CAS}} \quad \text{and} \tag{24}$$

$$y = \frac{\text{learning time for net trained by MM+CAS}}{\text{learning time for net trained by CAS}}. \tag{25}$$

In our experiments, we have found that it is not necessary to activate the monitor early in the learning process, since it is difficult for the monitor to find an I_{real} to map to D at that time. In Figure 2, points indicated by circles are obtained when the monitor is always activated. In contrast, points indicated by deltas are obtained for cases in which the monitor is activated when the error in the original network is smaller than 20%. We see that this results in savings in learning time. Note that the number of hidden units of networks trained by MM+CAS is never larger than those trained by the ''original'' CAS, since learning completes when either the ''original'' CAS completes or the monitor in MM+CAS finds a suitable mapping.

Finally, Table 1 summarizes the average normalized performance of CAS+MM. When the monitor is activated when the error in training is less than 20%, 13 cases lead to 11% reduction in the number of hidden units (as compared to the ''original'' CAS), taking only 95% of the training time. There are 17 cases where no improvement in the number of hidden units is found; the penalty in these cases is an additional 16% training time.

Our empirical results show that, in most trials, MM+CAS converges with less number of hidden units, although learning time is slightly longer. This reduction in the number of hidden units is important, since a trained ANN may have to be applied many times in the future.

6. Conclusions

In this paper, we present a method for reducing the number of hidden units required for convergence in the cascade-correlation learning algorithm. Our method is based on searching an intermediate output matrix that can be mapped to the desired output matrix. The validity of this mapping is verified by linear programming that finds a feasible solution for a set of linear inequalities. From our experimental results, our proposed mechanism leads to reduced number of hidden units but increased computation time. This trade-off is important, since the resulting smaller network is able to generalize better and is faster when deployed in the target application.

References

[1] C.-C. Teng, *Mixed-Mode Supervised Learning Algorithms for Multilayered Feed-Forward Neural Networks*, M.Sc. Thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana-Champaign, IL, August 1993.

[2] S. E. Fahlman, ‘‘Faster-Learning Variations on Back-Propagation: An Empirical Study,’’ *Proc. Connectionist Models Summer School*, pp. 38-51, Morgan Kaufmann, Palo Alto, CA, 1988.

[3] S. E. Fahlman and Christian Lebiere, *The Cascade-Correlation Learning Architecture*, Tech. Rep. CMU-CS-90-100, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, Feb. 1990.

[4] D. E. Rummelhart, J. L. McClelland, and the PDP Group (ed.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, Cambridge, MA 1986.

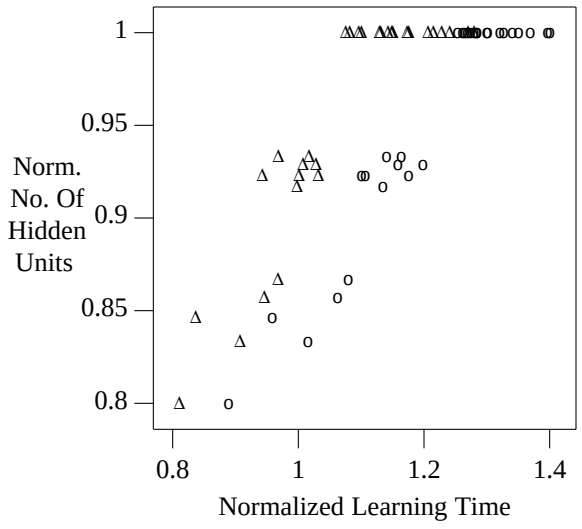

Figure 2. Performance of MM+CAS for solving the two-spiral problem normalized with respect to performance of the original CAS. (Circles represent results when the monitor is always activated; deltas represent results when the monitor is activated when the error in the original network is less than 20%.)

Table 1. Average normalized performance of MM+CAS for solving the two-spiral problem.

Number Of Cases	Monitor Activated When Error < 20%		Monitor Always Activated	
	Units	Time	Units	Time
13	0.893	0.959	0.893	1.091
17	1	1.164	1	1.310