MIXED-MODE SUPERVISED LEARNING ALGORITHMS FOR MULTILAYERED FEED-FORWARD NEURAL NETWORKS

BY

CHIN-CHI TENG

B.S. Eng., National Taiwan University, 1987

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1993

Urbana, Illinois

ABSTRACT

In this thesis, we present a new supervised learning mechanism for feed-forward neural networks. Our learning mechanism is different from those for traditional supervised learning algorithms (such as the back-propagation algorithm) that find a one-to-one mapping between the given input pattern matrix and the desired output pattern matrix. Instead, it finds one of the one-to-many mappings between the input matrix and an intermediate output matrix and transforms the intermediate output matrix into the desired output matrix using linear algebra techniques. The major contributions of this thesis are the design and experimentation of a new supervised learning mechanism for improving the performance of current supervised learning algorithms. These improvements include:(1) reduction of the learning time for some existing supervised learning algorithms (such as backpropagation and Quickprop); (2) reduction of the number of hidden units needed for convergence for the cascade-correlation learning algorithm. Further, our extensive experimental results show that our learning algorithm converges to within a reasonable range of error after a few training epochs, making it suitable for dynamic real-time applications in which the network may have to be re-trained periodically.

ACKNOWLEDGEMENTS

I express my sincere gratitude to all of the people who have helped me in the course of my graduate study. My thesis advisor, Professor Benjamin W. Wah, was always available for discussions and encouraged me to explore new ideas. I am grateful to Lon-Chan Chu, Arthur Ieumwananonth, Kumar Ganapathy, Yi Shang, Alan Y. Chang and all of my colleagues in the Center for Reliable and High-performance Computing for their invaluable suggestions and references.

Finally, I express my appreciation for the financial support provided by the National Aeronautics and Space Administration Contract NCC 2-481 and Joint Services Electronics Program Contract N00014-a0-J-1270.

TABLE OF CONTENTS

CHAPTER PAGE

1. INTRODUCTION 1
   1.1. Motivation 1
   1.2. Previous Work 3
   1.3. Approaches 4
   1.4. Organization of This Thesis 5
2. MODEL OF ARTIFICIAL NEURAL NETWORKS 6
   2.1. Classification of Artificial Neural Networks 6
   2.2. Multilayered Perceptrons 7
3. NONITERATIVE LEARNING ALGORITHMS FOR SINGLE-LAYER NEURAL NETWORKS 11
   3.1. Matrix Representations for Multilayered Perceptrons 11
   3.2. Transformation of Supervised Learning into Solving a Set of Linear Equations .. 14
   3.3. Transformation of Supervised Learning into Solving a Linear Programming Problem 19
4. MIXED-MODE SUPERVISED LEARNING ALGORITHM 25
   4.1. Structure of the Mixed-Mode Learning Algorithm 26
   4.2. Improved Mixed-Mode Supervised Learning Algorithm 31
      4.2.1. Description of the IMM learning algorithm 31
      4.2.2. Characteristics of the IMM learning algorithm 37
   4.3. Applications of MM and IMM Learning Algorithms 38
5. EXPERIMENTAL RESULTS 41
   5.1. Benchmarks Problems 41
      5.1.1. The XOR problem 41
      5.1.2. Sonar problem 42

5.1.3. Waveform problem 43
5.1.4. Two-spiral problem 44
5.1.5. Nonlinear function mapping problem 44
5.2. Experimental Results on Reduction of Learning Time 44
5.3. Convergence Property of the Mixed-Mode Supervised Learning 49
5.4. Experimental Results on Reduction of Hidden Units 52
6. CONCLUSIONS 56
REFERENCES 57

CHAPTER 1.

INTRODUCTION

In this thesis, we present a new supervised learning mechanism for feed-forward neural networks. Our learning mechanism is different from those for traditional supervised learning algorithms, such as the back-propagation algorithm, that find a one-to-one mapping between the given input pattern matrix and the desired output pattern matrix. Instead, it finds one of the many one-to-many mappings between the input matrix and an intermediate output matrix and transforms the intermediate output matrix into the desired output matrix using linear algebra techniques. The contributions of this thesis are the design and experimentation of a new supervised learning mechanism for improving the performance of current supervised learning algorithms. These improvements include:

(1) reduction of learning time for some existing supervised learning algorithms (such as back-propagation and Quickprop);

(2) reduction of the number of hidden units required for convergence for the cascade-correlation learning algorithm.

1.1. Motivation

For the past decade, people have shown intensive interest in artificial neural networks (ANN). The reasons that ANNs are of interest interest are as follows. First, ANNs have the property of massive parallelism. It is believed that ANNs have the potential for pursuing high-performance computing. Second, there are many important applications for ANNs. In the beginning, researchers used ANNs for speech and image recognition [13]. Later, ANNs were used widely in approximating a complicated and ill-defined function, given the values of the function at various points in the dependent variable space. This can be applied to many important engineering problems, such as nondestructive testing of materials. Both problems described above can be solved by using *supervised learning* to train an ANN for a specific application.

We can divide existing supervised learning algorithms into two classes depending on the configuration of an ANN during learning. The first class is the set of fixed-topology supervised learning algorithms. These algorithms start with an assigned network configuration, choose some values of training parameters, and train the network by applying gradient-descent techniques. After years of research, these learning algorithms are still too slow to be applied in real-world applications. For example, the back-propagation algorithm, the most popular learning algorithm for ANNs, may require thousands of epochs to learn a desired behavior even for a very simple problem. Hence, improving the learning speed becomes a major research issue for these learning algorithms.

The second class of supervised learning algorithms is the set of dynamic-topology supervised learning algorithms. These algorithms start with a small network configuration. During learning, they may change the original configuration into a more suitable one according to the current status of learning. The most promising algorithm in this class is the *cascade-correlation algorithm* [7]. It begins with a minimal network, then automatically trains and adds new hidden units, creating a multilayered structure. Compared with fixed-topology supervised learning algorithms, the learning time of the cascade-correlation algorithm is much shorter. However, since the cascade-correlation algorithm trains a network with more hidden units, it may increase the approximation quality with respect to the training patterns, but does not improve the network's generalization behavior. Reducing the number of hidden units required to converge is one way to improve the generalization behavior of ANNs trained by the cascade-correlation algorithm. An additional benefit of smaller networks is that they require less computing time when used in an application.

From the above discussion, we identify the two following research problems for existing supervised learning algorithms:

(1) how to improve the learning speed for fixed-topology learning algorithms; and

(2) how to reduce the number of units required to converge for dynamic-topology learning algorithms.

1.2. Previous Work

Backpropagation [18] is one of the most popular learning algorithms for fixed-topology multilayered feed-forward networks. Basically, it is a gradient descent technique to minimize some error criteria. From experience, it can be very slow for practical applications.

Many improvement strategies have been developed to speed up backpropagation. One way to optimize back-propagation learning is to automatically find proper values for its learning rate. Easton and Olivier suggested a calculation of the learning rate for backpropagation based on the assumption that similar training patterns result in similar gradients [5]. Darken and Moody decreased the learning rate during training [4]. Chan and Fallside adapted both learning rate and momentum during training [1]. A second way to speed up backpropagation is to adapt its local learning rate. In local adaptation, an independent learning rate is used for every connection, and the goal is to find an optimal learning rate for every weight [20, 22]. In addition to the two approaches discussed above, there are many other ways to improve back-propagation. For example, Fahlman developed *Quickprop* [8] by appling a collection of different heuristics for speeding up backpropagation. Riemiller and Braun used an adaptive version of the *Manhattan-Learning* rule [17] , called *RPROP*. In contrast to other learning algorithms, RPROP uses a fixed update step size not influenced by the magnitude of the gradient. Only the sign of the derivative is used to find the proper update direction. However, the learning performance of most of the above improvement strategies is not always predictable. Like standard backpropagation, they occasionally require long training times for some practical tasks.

Learning algorithms based on dynamic network topologies are broadly divided into two classes, *start big and remove* [12, 15, 19] and *start small and add* [7, 21]. The *cascade-correlation learning* algorithm [7] , developed by Fahlman and Lebiere, is one of the most promising; hence, this is the only algorithm in the class that we study in this thesis. So far, there is no existing work on reducing the size of a converged network in the cascade-correlation algorithm.

In this thesis, we present a new supervised learning mechanism. This mechanism improves the learning speed for some existing supervised learning algorithms. It also allows the cascade-correlation algorithm to reduce the number of hidden units required for convergence.

1.3. Approaches

For supervised learning, the application considered can be modeled as a mapping of an *input pattern matrix* P that consists of k patterns, each with m values, into an *output pattern matrix* D that consists of k patterns, each with n values. P is, therefore, a k-by-m matrix, and D, a k-by-n matrix. That is,

$$P = \begin{bmatrix} p_0 \\ p_1 \\ \cdot \\ \cdot \\ p_{k-1} \end{bmatrix}, \qquad D = \begin{bmatrix} d_0 \\ d_1 \\ \cdot \\ \cdot \\ d_{k-1} \end{bmatrix}, \tag{1.1}$$

where p_i is the i-th input pattern (a vector of m values), and d_i is the corresponding i-th output pattern (a vector of n values). The neural network to be learned performs a mapping from P to D.

Our new learning mechanism is different from the traditional approaches described in Section 1.2. Existing supervised learning algorithms focus on finding a one-to-one mapping between the input matrix and the corresponding output matrix. They generally compare the actual output matrix and the desired output matrix, and depending on the error, decide how to adapt the weights of the connections (as done in the back-propagation algorithm), or how to adapt the network configuration (as done in the cascade-correlation algorithm). The number of learning epochs is largely due to the convergence time required for finding one-to-one mappings. (Note : presenting all the training patterns to the network once is called one epoch of training.)

The number of learning epochs can be significantly reduced 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 one of them is found. The new learning mechanism studied in this thesis exploits this property. It 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 (called I_{real}). It then transforms I_{real} to D by finding the least-squared error solution of a set of linear equations or solving a linear programming problem.

Reducing the number of learning epochs is important. For fixed-topology learning algorithms, the reduction in the number of learning epochs can result in lower learning time. For the cascade-correlation algorithm, the reduction in the number of learning epochs can reduce the number of hidden units required for convergence, since the number of units required for convergence is a monotonically nondecreasing function of learning epochs.

1.4. Organization of This Thesis

The thesis is organized into seven chapters. Chapter 2 defines the model of an ANN and points out the scope of this thesis. Chapter 3 presents two noniterative learning algorithms for single-layer neural networks: one uses the technique of linear space mapping, and the other uses linear programming. These two noniterative learning algorithms are employed to transform an iterative supervised learning algorithm that finds one-to-one mappings into one that finds one-to-many mappings. The structure of our new learning mechanism is shown in Chapter 4. Chapter 5 shows the experimental results and compares them with those of existing supervised learning algorithms. Finally conclusions are drawn in Chapter 6.

CHAPTER 2.

MODEL OF ARTIFICIAL NEURAL NETWORKS

In this chapter, we first classify neural networks and identify the scope of neural networks that this research emphasizes (multilayered feed-forward ANNs for pattern classifiers). We then present the basic operations of multilayered feed-forward ANNs and briefly introduce three general supervised learning algorithms: backpropagation, Quickprop, and cascade-correlation. These three algorithms will be compared with our proposed learning mechanism in Chapter 5.

2.1. Classification of Artificial Neural Networks

A neural network can be classified in two ways: application-oriented classification and functional classification [14]. In an application-oriented classification, we categorize the neural networks by its application domain. There are five major application fields for neural networks: (1) *auto-associative memory*, (2) *hetero-associative memory*, (3) *pattern classifier*, (4) *dimensionality-reduction systems*, and (5) *optimization and constraint-satisfaction systems*. Among these different neural network applications, pattern classifiers are considered very important and have been widely used in image analysis, image understanding, speech understanding, and so on. This thesis considers only neural networks for the application of pattern classifiers.

In functional classification, a neural network is specified by its learning algorithm, net topology (i.e., connectivity), and activation. The mechanism to train a pattern classifier is supervised learning. In supervised learning, the training set specifies a desired output pattern for each input pattern. Learning tries to minimize the error between the actual output and the desired output. The target of supervised learning is to learn from examples and to apply globally the knowledge learned. Feed-forward connections are usually employed for supervised learning. Their characteristics are that the links are unidirectional with no cyclic connections. Figure 2.1 shows examples of feed-forward and feedback connections.

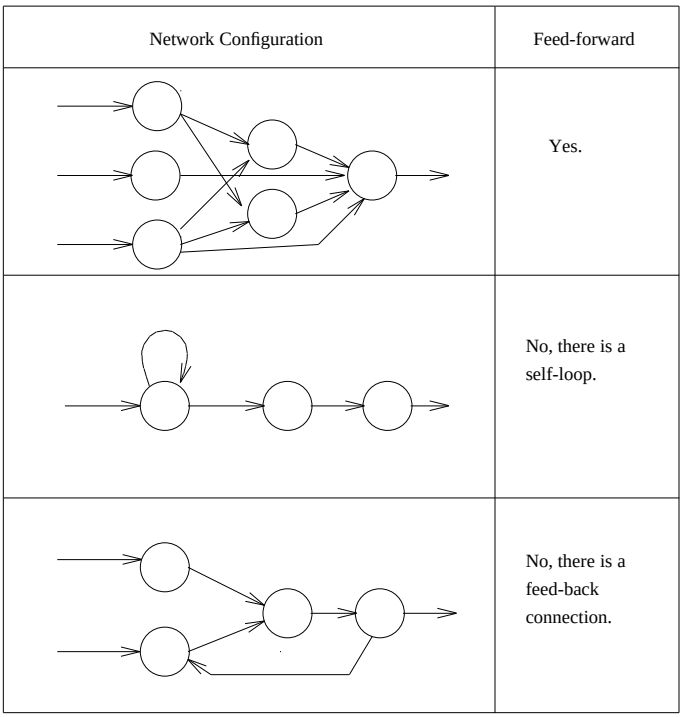


Figure 2.1 : Connectivity of Neural Networks.

Each neuron uses an activation function to map inputs to outputs. One of the most popular activation functions is the sigmoid function. This is a smooth nonlinear continuous function that is differentiable everywhere. This function is defined as

$$\text{sigmoid}(x) = \frac{1}{(1 + e^{-x})} \tag{2.1}$$

The output of the sigmoid function will be approximately equal to *zero* if its input value approaches negative infinity, and to *one* if its input value approaches positive infinity.

2.2. Multilayered Perceptrons

Multilayered feed-forward neural networks used for pattern classifiers are usually called *multilayered perceptrons*. The neurons in a multilayered perceptron are of three types : *input units*, *hidden units*, and *output units*. Input units are located in the input layer and receive inputs from the external environment; output units send outputs of neural networks to the

external environment; and hidden units are invisible to the external environment. Figure 2.2 shows an example of a multilayered perceptron.

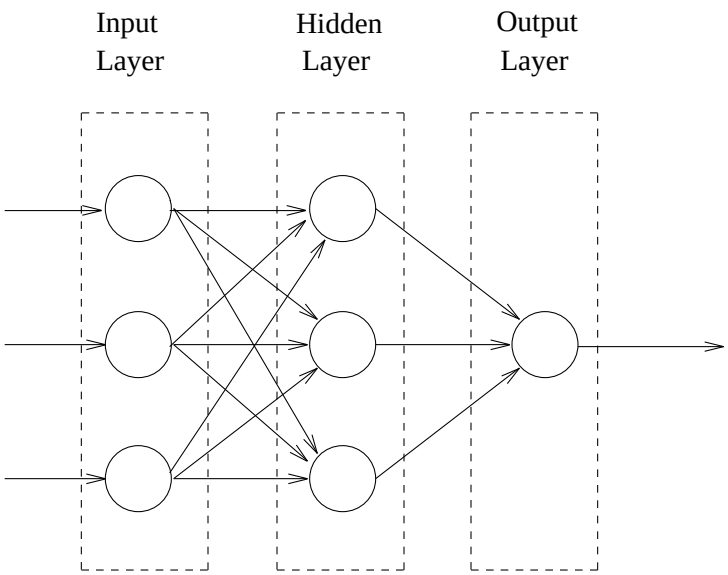


Figure 2.2 : Example of a Multilayered Perceptron.

For each hidden unit or output unit, its output o_j (the output value of unit j) is activated by the following equation:

$$o_j = \text{sigmoid}(\sum_i w_{i,j} \cdot o_i + \theta_j), \text{ for all unit } i \text{ connected to unit } j. \quad (2.2)$$

where $w_{i,j}$ is the weight of the connection from unit i to unit j, and θ_j is the bias of unit j.

The basic algorithm for training a multilayered perceptron is *backpropagation* (BP), which operates in two phases. In the *forward pass*, the activation pattern of an input vector is propagated through the network to produce an output. The error between desired outputs and actual outputs is calculated as follows.

$$E = \frac{1}{2}\sum_k E_k = \frac{1}{2}\sum_k (t_k - o_k)^2, \text{ for all unit } k \text{ in output layer} \quad (2.3)$$

where o_k is the actual output of output unit k and t_k is the desired output of output unit k. In the *backward pass*, E is propagated from the output layer to the input layer and is used to update the weights of connections according to the delta rule :

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} , \tag{2.4}$$

and

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t) , \tag{2.5}$$

where the step size η is called the *learning rate*. To speed up training, we can add a *momentum* term $\alpha \Delta w_{ij}(t-1)$. The delta rule then becomes

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t) + \alpha \Delta w_{ij}(t-1) . \tag{2.6}$$

The addition of the momentum has two effects: (1) oscillations in the proximity of a local minima are damped, and (2) convergence is accelerated in wide plateaus where the gradient is very small.

Quickprop [6] is another commonly used training algorithm for ANNs. It uses a collection of heuristics for optimizing backpropagation. In the back-propagation phase, the gradient of error function E is proportional to $o_i(1 - o_i)$, where o_i is the actual output value of unit i. When o_i is close to 0.0 or 1.0, the term of $o_i(1 - o_i)$ is very small, slowing down gradient descent. To avoid this condition, Quickprop uses a modified gradient as follows:

$$\frac{\partial E}{\partial w_{ij}} \propto o_i(1 - o_i) + 0.1 . \tag{2.7}$$

Second, a new error function is used in Quickprop. This function ignores an error when its absolute value is less than 0.1, and units with a quadratic error less than 0.01 are not trained further. This modification avoids over-training units in the network. As a third heuristic, Quickprop uses a small weight decay in the gradient to prevent weights from growing too large. Other approaches to speed up BP have been stated in Section 1.2. To the best of our knowledge, there are no variations of BP that have stable learning performance.

The cascade-correlation algorithm [7] differs in many ways from Quickprop and backpropagation. It begins with a minimal network, and then 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 the unit output value and the output error. Once a new hidden unit has been trained and added to the network, its input-side weights are frozen. Then Quickprop is employed to update its output-side weights. The cascade-correlation algorithm has faster learning time, but uses a variable topology, and the number of hidden units when the algorithm terminates is not bounded. In essence, it increases the approximation quality of the network with respect to its training patterns, but does not improve its generalization behavior.

CHAPTER 3.

NONITERATIVE LEARNING ALGORITHMS FOR SINGLE-LAYER NEURAL NETWORKS

In this chapter, we present two noniterative learning algorithms for single-layer neural networks, that is, neural networks without hidden units. The first noniterative learning algorithm transforms supervised learning into solving a solution of a set of linear equations, and the second transforms supervised learning into solving a feasible solution of a set of inequalities using linear programming techniques. For convenience of addressing these noniterative learning algorithms, a matrix representation for neural networks will be introduced in the first section. These two noniterative learning algorithms are important for transforming supervised learning from finding one-to-one mappings to finding one-to-many mappings. This transformation allows the number of learning epochs to be significantly reduced. This concept will be presented in the next chapter.

3.1. Matrix Representations for Multilayered Perceptrons

Supervised learning of a neural network can be modeled as a mapping of an *input pattern matrix* P that consists of k patterns, each with m values, into an *output pattern matrix* D that consists of k patterns, each with n values, where P is, therefore, a k-by-m matrix, and D, a k-by-n matrix. That is,

$$P = \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{k-1} \end{bmatrix}_{k \times m}, \qquad D = \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{k-1} \end{bmatrix}_{k \times n}, \tag{3.1}$$

where p_i is the i-th input pattern (a vector of m values), and d_i is the corresponding i-th output pattern (a vector of n values). The neural network to be learned performs a mapping from P to D.

In a similar way, given a multilayered feed-forward ANN, we can represent it by a set of weight matrices.

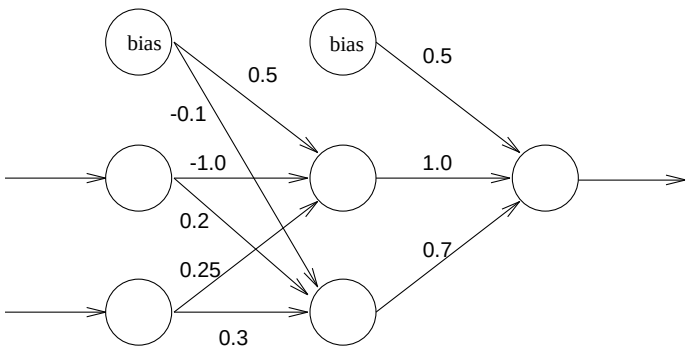
Definition. $W^{i,j}$ is defined as the *weight matrix* between layer i and layer j. The element $(W^{i,j})_{k,l}$ of weight matrix $W^{i,j}$ is the weight of the connection between the *k-th* unit of layer i and the *l-th* unit of layer j. In general, the input layer is layer 1, and the first unit of each layer is the bias of the layer.

Example 3.1 shows how weight matrices can be used to represent a multilayered feed-forward neural network.

Example 3.1. Given a multilayered perceptron with one hidden layer, two input units, two hidden units, and one output unit, its weight matrices are as follows:

$$W_{1,2} = \begin{bmatrix} 0.5 & -0.1 \\ -1.0 & 0.2 \\ 0.25 & 0.3 \end{bmatrix} \quad W_{2,3} = \begin{bmatrix} 0.5 \\ 1.0 \\ 0.7 \end{bmatrix}. \tag{3.2}$$

The neural network specified by the two matrices above is graphically represented as follows:



From the matrix model for neural networks, neural network computations are equivalent to matrix multiplications. For example, given a single-layer feed-forward neural network, we can represent it as a weight matrix $W^{1,2}$. Since we designate the first row of a weight matrix as the bias, we have to modify P, the input matrix, by adding a column of 1's. That is,

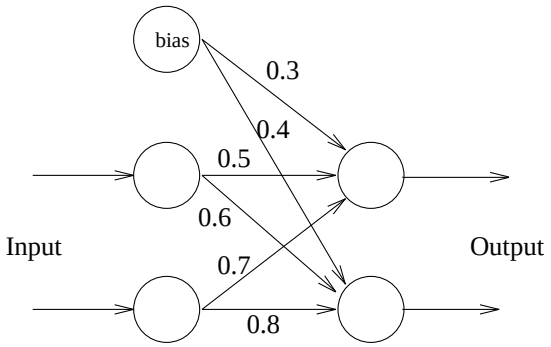
$$\tilde{P} = \left[\begin{array}{c|c} \begin{matrix}1.0\\1.0\\.\\.\\1.0\end{matrix} & P \end{array}\right] \tag{3.3}$$

The real output pattern matrix D_{real} can then be calculated by matrix multiplication as follows.

$$((D_{real})_{k\times m})_{i,j} = f((\tilde{P}W^{1,2})_{i,j}) \quad \text{for all } i = 1, 2, ..., k \text{ and } j = 1, 2, ..., m, \tag{3.4}$$

where f(x) is the activation function of the output units.

Example 3.2. Given a single-layer neural network,



its weight matrix can be represented by the following weight matrix:

$$W^{1,2} = \begin{bmatrix} 0.3 & 0.4 \\ 0.5 & 0.6 \\ 0.7 & 0.8 \end{bmatrix} \tag{3.5}$$

If the activation function of units is sigmoid, and the input matrix P is

$$P = \begin{bmatrix} 0.0 & 1.0 \\ 0.0 & 0.0 \end{bmatrix}, \tag{3.6}$$

then the modified input matrix, $\tilde{P}$, with bias included is

$$\tilde{P} = \left[\begin{array}{c|cc} 1.0 & & \\ 1.0 & & P \end{array}\right] = \begin{bmatrix} 1.0 & 0.0 & 1.0 \\ 1.0 & 0.0 & 0.0 \end{bmatrix}, \tag{3.7}$$

and the output matrix D_{real} is

$$D_{real} = sigmoid(\tilde{P}W^{1,2}) = sigmoid\left(\begin{bmatrix}1.0 & 1.2\\ 0.3 & 0.4\end{bmatrix}\right) = \begin{bmatrix}0.731 & 0.769\\ 0.574 & 0.599\end{bmatrix} \quad (3.8)$$

3.2. Transformation of Supervised Learning into Solving a Set of Linear Equations

In the last section, we discussed the computations of an ANN as a sequence of matrix multiplications. In this section, we show that, by representing supervised learning of a single-layer ANN as a single matrix multiplication, we can find the weights of the ANN by solving a set of linear equations. Assuming $P_{k\times m}$, an input pattern matrix, $D_{k\times n}$, a desired output pattern matrix, no bias weights, and an activation function $f(x)=x$, then the learning problem of this single-layer neural network is the same as solving a set of linear equations

$$P \cdot W = D \quad (3.9)$$

to obtain the weight matrix $W_{m\times n}$ [18]. If a bias is included, then the learning problem is the same as solving $\tilde{P}\ W = D$.

If the activation function of an output unit is a sigmoid function, we can derive the modified desired output matrix D_0, where $sigmoid((D_0)_{i,j}) \approx (D)_{i,j}$. If D is binary, then D_0 can be defined as follows [9].

$$(D_0)_{i,j} = \begin{cases} \tau & \text{if } (D)_{i,j}=1 \\ -\tau & \text{if } (D)_{i,j}=0 \end{cases}, \quad (3.10)$$

where τ is a large enough positive number. As a result, the learning problem becomes the problem of solving a set of linear equations:

$$P \cdot W = D_0 \ . \quad (3.11)$$

We can see that if a weight matrix W is obtained such that $\tilde{P}\ W = D_0$, then the output matrix will be almost equal to D, since $sigmoid(\tau)\approx 1$ and $sigmoid(-\tau)\approx 0$. The above method can be also applied in similar fashion, if the activation function of an output unit is a step function.

The modification procedure for the desired output matrix D proposed by Goggin *et al.* [9] is only suitable for the case in which the desired output is binary. If the desired output is any value between 0.0 and 1.0, we can modify matrix D into D_0 as follows. Let S^{-1} be the inverse function of the sigmoid function.

$$(D_0)_{i,j} = \begin{cases} 3 & \text{if } (D)_{i,j} \geq S^{-1}(3) \\ -3 & \text{if } (D)_{i,j} \leq S^{-1}(-3) \\ S^{-1}((D)_{i,j}) & \text{otherwise} \end{cases} \tag{3.12}$$

Again, the above transformation will let $sigmoid\,(D_0) \approx D$.

In short, the noniterative learning algorithm presented above approximates a nonlinear activation function and uses linear algebra to transform the supervised learning algorithm into solving a solution of a set of linear equations. Unlike general supervised learning, it is not iterative, so that its computational complexity is deterministic. When training single-layer ANNs, this noniterative learning algorithm is usually faster than a general iterative supervised learning method. However, this technique is not universal as it can only be applied to single-layer ANNs.

In a matrix representation for ANNs, the input matrix $P_{k\times m}$ is composed of m column vectors in a k-D space. Multiplying matrix $P_{k\times m}$ by a weight matrix will result in a vector that is a linear combination of the columns of $P_{k\times m}$. In neural network training, the number of training patterns is usually larger than that of input units, that is, $k > m$. Therefore, the column space of $P_{k\times m}$ is a subspace of the entire k-D space since $k > m$. For example, in the XOR problem, the input matrix P is

$$P = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}. \tag{3.13}$$

The column space of P is a 2-D surface in a 4-D space. Hence, it is very likely that a given output matrix D_0 will not lie in the column space of P. As a result, we usually can obtain the least-squared solution for the set of linear equations $P \cdot W = D_0$ but not its exact solution. There are two general methods for obtaining least-squared solutions of linear equations. The

first one uses the concept of *pseudo-inverse* [2]. The least-squared solution W^+ of $P \cdot W = D_0$ is equal to

$$W^+ = ((P^T\ P)^{-1}\ P^T)\ D_0\ , \tag{3.14}$$

where A^T is the transpose matrix of A. The pseudo-inverse method has a shortcoming that if matrix P does not have full rank, then $(P^T \cdot P)$ will be singular. Hence, we cannot obtain the pseudo-inverse of P, since there is no inverse for matrix $(P^T \cdot P)$.

The second method for obtaining least-squared solutions uses *singular value decomposition* (SVD) [16]. The procedure is as follows. First, input matrix $P_{k\times m}$ is decomposed into three matrices by singular value decomposition :

$$P_{k\times m} = U_{k\times k} \cdot \Omega_{k\times m} \cdot V^T{}_{m\times m}\ , \tag{3.15}$$

where U and V are orthogonal matrices, and Ω is a diagonal matrix. The least-squared solution W^+ is then derived.

$$W^+{}_{m\times n} = V_{m\times m} \cdot \Omega^+{}_{m\times k} \cdot U^T{}_{k\times k} \cdot D_{0k\times n}, \quad \text{where } (\Omega^+)_{i,i} = \frac{1}{(\Omega)_{i,i}}. \tag{3.16}$$

The method based on singular value decomposition is more general than the pseudo-inverse method, since there is no constraint that P has to be full rank. However, singular value decomposition is computationally expensive as it is iterative. Fortunately, input matrix P has full rank in almost all training problems in neural networks, because the number of training patterns is usually much larger than the number of input units. Hence, we usually use the pseudo-inverse method to solve the set of linear equations $P \cdot W = D_0$ for training single-layer neural networks. In the case of P without full rank, we then use singular value decomposition.

The noniterative supervised learning algorithm described above finds the projection of matrix D_0 on subspace span$\{P\}$ in order to obtain a proper weight matrix. We call this method a *linear mapping* method. Example 3.3 shows how the linear mapping method works.

Example 3.3. Consider a single-layer neural network for solving the AND problem. The input matrix P and the desired output matrix D_0 are given as follows:

$$P = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \quad D = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} . \tag{3.17}$$

Since we want to apply the linear mapping method to obtain the weight matrix, we have to solve the set of equations

$$\tilde{P} \cdot W = D_0 = \begin{bmatrix} -10 \\ -10 \\ -10 \\ 10 \end{bmatrix} , \tag{3.18}$$

and obtain the least-squared solution

$$W^+ = ((P^T P)^{-1} P^T) D_0 = \begin{bmatrix} 10 \\ 10 \\ -15 \end{bmatrix} . \tag{3.19}$$

We can verify the result as follows:

$$D_{real} = sigmoid(P \cdot W^+) = sigmoid\left(\begin{bmatrix} -15 \\ -5 \\ -5 \\ 5 \end{bmatrix} \right) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} . \tag{3.20}$$

The linear mapping method has two important characteristics. First, in terms of linear space, the least-squared solution W^+ of Eq. (3.11) exists such that $P \cdot W^+ = D_0$ if only if

$$\text{span}\{D_0\} \subseteq \text{span}\{P\}. \tag{3.21}$$

That means that if the column space of input matrix P contains the column space of the modified desired output matrix D_0, then we will be able to obtain an actual output matrix that is very close to the desired output matrix.

Second, the solutions found by the linear mapping method are not optimal. This is true because the projection of D_0 on the column space of P may not be in the desired subspace, even though the column space of P overlaps the desired subspace. Example 3.4 illustrates this situation.

Example 3.4. Consider a single-layer neural network for solving a given problem whose input matrix P and desired output matrix D are

$$P = \begin{bmatrix} -15 & -5 \\ -1 & 1 \\ -1 & -3 \end{bmatrix} \quad \text{and} \quad D = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}. \tag{3.22}$$

After applying the linear mapping method, we obtain a weight matrix,

$$W^{+} = ((P^{T} P)^{-1} P^{T}) D_0 = \begin{bmatrix} -0.278 \\ -1.389 \end{bmatrix}. \tag{3.23}$$

We can verify the result as follows.

$$D_{real} = sigmoid(P \cdot W^{+}) = sigmoid\left(\begin{bmatrix} 11.1 \\ -1.1 \\ 4.4 \end{bmatrix} \right) \approx \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}. \tag{3.24}$$

Obviously, D_{real} obtained in Eq. (3.24) is not the same as the desired output matrix D. However, some weight matrices exist which can satisfy $sigmoid(P \cdot W) = D_{real}$, since the column space of P contains some vectors that can be close to D after being activated by the sigmoid function. For example, if

$$W = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \tag{3.25}$$

then

$$D_{real} = sigmoid(P \cdot W) = sigmoid\left(\begin{bmatrix} 15 \\ 1 \\ 1 \end{bmatrix} \right) \approx \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}. \tag{3.26}$$

From this example, we can see that the linear mapping method cannot obtain the optimal solution. Better solutions may be found by other iterative supervised learning algorithms.

3.3. Transformation of Supervised Learning into Solving a Linear Programming Problem

In contrast to the last section, we present in this section the transformation of the supervised learning of single-layer ANNs into a set of linear inequalities rather than a set of linear equations. We then use the *simplex method* to obtain a feasible solution for this set of linear inequalities. This feasible solution will be used as the weights of neural networks. This transformation is only applied to those application problems whose outputs are binary.

Given $P_{k\times m}$, an input pattern matrix, and $D_{k\times 1}$, a one-column desired output pattern matrix, they can be represented as

$$P = \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{k-1} \end{bmatrix}_{k\times m}, \qquad D = \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{k-1} \end{bmatrix}_{k\times 1}, \tag{3.27}$$

where p_i is the i-th input pattern (a vector of m values), and d_i is the corresponding i-th output pattern (a vector of one value). The 40-20-40 criterion is applied here, that is, if an output is larger than 0.6, we consider it as a logic ONE, and if an output is smaller than 0.4, then we consider it as a logic ZERO. While the sigmoid function is used as the activation function, the learning problem of a single-layer neural network is the same as obtaining a weight matrix $W_{m\times 1}$ such that

$$p_i \cdot W = \begin{cases} \geq S^{-1}(0.6) & \text{if } d_i = 1 \\ \leq S^{-1}(0.4) & \text{if } d_i = 0 \end{cases}, \text{ for all } i = 0, 1, 2, .., k-1, \tag{3.28}$$

where $S^{-1}(x)$ is the inverse function of the sigmoid function. Since $S^{-1}(0.6) = -S^{-1}(0.4)$, Eq. (3.28) can be represented in matrix form as follows.

$$\hat{P} \cdot W \geq \begin{bmatrix} S^{-1}(0.6) \\ S^{-1}(0.6) \\ \vdots \\ S^{-1}(0.6) \end{bmatrix}_{k\times 1}, \tag{3.29}$$

where

$$\hat{P} = \begin{bmatrix} \hat{p}_0 \\ \hat{p}_1 \\ \cdot \\ \cdot \\ \hat{p}_{k-1} \end{bmatrix}_{k \times m} \tag{3.30}$$

and

$$\hat{p}_i = \begin{cases} p_i & \text{if } d_i = 1 \\ -p_i & \text{if } d_i = 0 \end{cases} . \tag{3.31}$$

We can see that if a weight matrix W satisfies Eq. (3.29), then the element of the actual output matrix D_{real} is

$$(D_{real})_{i,1} = sigmoid((P \cdot W)_{i,1}) = \begin{cases} \geq 0.6 & \text{if } d_i = 1 \\ \leq 0.4 & \text{if } d_i = 0 \end{cases} . \tag{3.32}$$

Obtaining a weight matrix W, satisfying Eq. (3.20), is very similar to finding a feasible solution of a linear program. The only difference is that elements in weight matrix W can be negative, while variables in linear programming problems have to be positive. To cope with this problem, we transform the elements of W as follows. Given a matrix W,

$$W = \begin{bmatrix} w_0 \\ w_1 \\ \cdot \\ \cdot \\ w_{m-1} \end{bmatrix}_{m \times 1} , \tag{3.33}$$

let

$$w_i = x_i - y_i , \quad \text{for all } i = 0, 1, .., m-1, \text{ where } x_i \geq 0, \text{ and } y_i \geq 0. \tag{3.34}$$

Hence, Eq. (3.29) becomes

$$\left[\hat{P} \;\middle|\; -\hat{P} \right]_{k \times 2m} \cdot \left[\frac{X}{Y} \right]_{2m \times 1} \geq \begin{bmatrix} S^{-1}(0.6) \\ S^{-1}(0.6) \\ \cdot \\ \cdot \\ S^{-1}(0.6) \end{bmatrix}_{k \times 1} , \tag{3.35}$$

where

$$X = \begin{bmatrix} x_0 \\ x_1 \\ \cdot \\ \cdot \\ \cdot \\ x_{m-1} \end{bmatrix}_{m\times 1}, \qquad Y = \begin{bmatrix} y_0 \\ y_1 \\ \cdot \\ \cdot \\ \cdot \\ y_{m-1} \end{bmatrix}_{m\times 1}. \tag{3.36}$$

Since variables x_i and y_i in Eq. (3.35) are all positive, the values of variables that can satisfy Eq. (3.35) can be obtained using linear programming. Example 3.5 illustrates the procedures for transforming supervised learning into a linear programming problem.

Example 3.5. Consider the same problem as in Example 3.4. We want to use a single-layer neural network for solving a given problem whose input matrix P and desired output matrix D are

$$P = \begin{bmatrix} -15 & -5 \\ -1 & 1 \\ -1 & -3 \end{bmatrix}, \quad \text{and} \quad D = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}. \tag{3.37}$$

Equations (3.31) and (3.35) are applied to obtain

$$\hat{P} = \begin{bmatrix} -15 & -5 \\ -1 & 1 \\ -1 & -3 \end{bmatrix}, \tag{3.38}$$

and a set of inequalities

$$\left[\begin{array}{c|c} \hat{P} & -\hat{P} \end{array}\right] \cdot \begin{bmatrix} X \\ \hline Y \end{bmatrix} = \begin{bmatrix} -15 & -5 & 15 & 5 \\ -1 & 1 & 1 & -1 \\ -1 & -3 & 1 & 3 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ y_0 \\ y_1 \end{bmatrix} \geq \begin{bmatrix} 0.41 \\ 0.41 \\ 0.41 \end{bmatrix}, \tag{3.39}$$

where x_0, x_1, y_0, and $y_1 \geq 0$. The simplex method is then applied to obtain a feasible solution as follows:

$$\begin{bmatrix} x_0 \\ x_1 \\ y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} 0.0 \\ 0.41 \\ 0.0 \\ 0.0 \end{bmatrix}. \tag{3.40}$$

The weight matrix W is, therefore,

$$W = \begin{bmatrix} x_0 - y_0 \\ x_1 - y_1 \end{bmatrix} = \begin{bmatrix} -0.41 \\ 0.0 \end{bmatrix} . \tag{3.41}$$

We can verify the result as follows:

$$D_{real} = sigmoid(P \cdot W) = \begin{bmatrix} 1.0 \\ 0.6 \\ 0.6 \end{bmatrix} . \tag{3.42}$$

In the above discussion, we have assumed the number of output units to be one, i.e., D is a one-column matrix. 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 subproblems. In each subproblem, one of the column vectors of D is used to obtain one matrix $\hat{P}$ by applying Eq. (3.31), and $\hat{P}$ is then used to obtain the corresponding column vector of weight matrix W.

For the case of one output unit, 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

$$R_D^k \cap \text{span}\{P\} \neq \phi , \tag{3.43}$$

where

$$R_D^k = \{[x_0, x_1, \ldots, x_{k-1}]^T \mid sign(x_i) = sign(d_i - 0.5),\ 0 \le i \le k-1 \} ,$$

$$\text{and } d_i = 0 \text{ or } 1 . \tag{3.44}$$

In supervised learning, Eq. (3.43) is seldom satisfied; that is, Eq. (3.35) usually has no feasible solution. At this time, the objective has to be changed to finding a set of values for all variables such that the number of inequalities that are satisfied is maximized. That is equivalent to finding a set of weights such that the number of correct output patterns is maximal. Let's set

$$A = \left[\hat{P} \;\middle|\; -\hat{P} \right]_{k \times 2m}, \quad V = \left[\frac{X}{Y} \right]_{2m \times 1}, \quad \text{and } B = \begin{bmatrix} S^{-1}(0.6) \\ S^{-1}(0.6) \\ \vdots \\ S^{-1}(0.6) \end{bmatrix}_{k \times 1} \tag{3.45}$$

For obtaining a set of weights such that the number of correct output patterns is maximal, the optimization problem can be formulated as follows.

$$\text{Maximize } \sum_{i=1}^{k} u_0\left(\sum_{j=1}^{2m} a_{i,j} v_j - b_i \right), \text{ such that} \tag{3.46}$$

$$a_{1,1}v_1 + a_{1,2}v_2 + \cdots + a_{1,2m}v_{2m} \geq b_1$$

$$a_{2,1}v_1 + a_{2,2}v_2 + \cdots + a_{2,2m}v_{2m} \geq b_2$$

$$a_{3,1}v_1 + a_{3,2}v_2 + \cdots + a_{3,2m}v_{2m} \geq b_3$$

$$\vdots$$

$$a_{k,1}v_1 + a_{k,2}v_2 + \cdots + a_{k,2m}v_{2m} \geq b_k$$

where $v_i \geq 0$ for all $i = 1, 2, .., 2m$, and $u_0(x)$ is a step function with transition at 0.

However, the overhead for solving the above nonlinear optimization problem is very high and, therefore, is not practical to be used in neural network learning. 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 solving linear optimization problems. First, we add a slack variable y_i to every constraint inequality, making every inequality to be an equality:

$$a_{i,1}v_1 + a_{i,2}v_2 + \cdots + a_{i,2m}v_{2m} - y_i - b_i = 0,$$

$$\text{and } y_i \geq 0 \text{ for all } i = 1, 2, 3, ..., k. \tag{3.47}$$

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

$$z_i = b_i - a_{i,1}v_1 - a_{i,2}v_2 + \cdots - a_{i,2m}v_{2m} + y_i,$$

$$\text{and } z_i \geq 0 \text{ for all } i = 1, 2, 3, ..., k, \tag{3.48}$$

so that the set of equations in Eq. (3.48) always has a feasible solution. 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.

Minimize $\sum_{i=1}^{k} z_i$, such that (3.49)

$$z_1 = b_1 - a_{1,1}v_1 - a_{1,2}v_2 - \cdots - a_{1,2m}v_{2m} + y_1$$

$$z_2 = b_2 - a_{2,1}v_1 - a_{2,2}v_2 - \cdots - a_{2,2m}v_{2m} + y_2$$

$$z_3 = b_3 - a_{3,1}v_1 - a_{3,2}v_2 - \cdots - a_{3,2m}v_{2m} + y_3$$

$$\vdots$$

$$z_k = b_k - a_{k,1}v_1 - a_{k,2}v_2 - \cdots - a_{k,2m}v_{2m} + y_k$$

where $v_i \geq 0$ for all $i = 1, 2, .., 2m$, and y_j, $z_j \geq 0$, for all $i = 1, 2, .., k$.

Since we minimize $\sum z_i$, the optimal solution of Eq. (3.49) should have a small value for each z_i. Therefore, it is very likely that the inequality $z_i \leq y_i$ can be satisfied, and the original constraint inequalities listed in Eq. (3.46) are satisfied with a large probability. This is true because if $z_i \leq y_i$, then

$$a_{i,1}v_1 + a_{i,2}v_2 + \cdots + a_{i,2m}v_{2m} = y_i + b_i - z_i \geq b_{i.} \tag{3.50}$$

Note that when more inequalities in Eq. (3.46) are satisfied, the number of correct output patterns obtained in the output layer also increases.

CHAPTER 4.

MIXED-MODE SUPERVISED LEARNING ALGORITHM

In this chapter, a new supervised learning mechanism, called *mixed-mode supervised learning*, is presented. The structure of the mixed-mode learning algorithm is described in Section 4.1. We then describe an improved mixed-mode learning algorithm in Section 4.2. The mixed-mode learning algorithm transforms the learning problem from finding a one-to-one mapping into one that finds one-to-many mappings. Since the learning objective is relaxed, it needs fewer training epochs than existing supervised learning algorithms. Before discussing the details of our mixed-mode learning algorithm, we summarize in Table 4.1 the symbols used in this chapter.

Table 4.1: Important symbols and their meanings.

Symbol	Meaning
k	number of training patterns
m	number of input units
n	number of output units
P	input matrix in training set
D	desired output matrix in training set
D_0	modified desired output matrix derived from D by the procedure described in Section 3.2
D_{real}	actual output matrix in the output layer
I	desired intermediate output matrix
I_{real}	actual intermediate output matrix
$W^{i,j}$	weight matrix between layers i and j

4.1. Structure of the Mixed-Mode Learning Algorithm

We have described in Section 3.1 that learning can be modeled as finding a mapping from input matrix P to output matrix D. Since existing supervised learning algorithms focus on finding a one-to-one mapping between the input matrix and the corresponding output matrix, they generally compare the actual output matrix and the desired output matrix, and according to the error, decide how to adapt the weights of the connections (as is done in the back-propagation algorithm), or how to adapt the network configuration (as is done in the cascade-correlation algorithm). The number of learning epochs is largely due to the convergence time required for finding one-to-one mappings.

The number of learning epochs can be significantly reduced 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 one of them is found. The mixed-mode learning algorithm in this thesis exploits this property. It 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 noniterative learning algorithms for the single-layer neural networks described in Chapter 3.

We use Figure 4.1. to illustrate our mixed-mode supervised learning method (MM). Given a network 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} and apply one of two noniterative learning algorithms stated in Chapter 3 to map I_{real} into 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 that j-th unit that is connected to the output layer when the i-th training pattern is applied.

First of all, we consider the case in which the linear mapping method is employed in the monitor. Once I_{real} whose column space contains span$\{D_0\}$ is acquired in the monitor, the linear mapping method can map I_{real} into D_0 by solving the set of linear equations $I_{real} \cdot W = D_0$. This is true because the least-squared solution W^+ exists such that $I_{real} \cdot W^+ = D_0$ if and only if

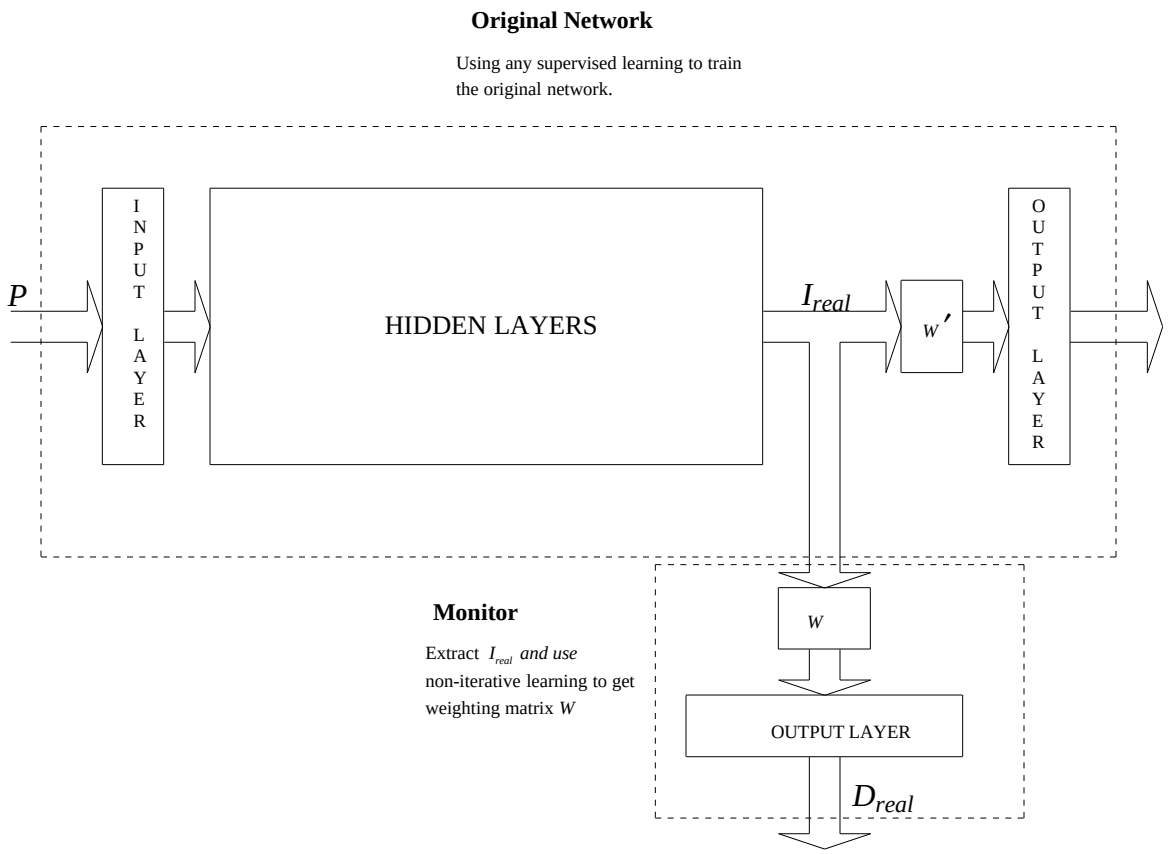


Figure 4.1 : Mixed-Mode Supervised Learning.

$$\text{span}\{D_0\} \subseteq \text{span}\{I_{real}\}. \tag{4.1}$$

At this time the real output matrix D_{real} is

$$D_{real} = \text{sigmoid}(I_{real} \cdot W^{+}) = \text{sigmoid}(D_0) \approx D, \tag{4.2}$$

and learning completes.

This new learning mechanism reduces the number of learning epochs since it has a relaxed learning objective. The objective of common supervised learning is to find a one-to-one mapping between the given input matrix P and the desired output matrix D. This is usually more difficult to achieve. In our proposed mechanism, learning only tries to obtain an

intermediate output matrix I_{real} whose column space contains the column space of the modified desired output matrix D_0. Since there are many matrices 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 set of linear equations.

Similarly, if we employ the linear programming method in the monitor, whenever the monitor obtains I_{real} that satisfies

$$R_{d_i}^k \cap \text{span}\{I_{real}\} \neq \phi \text{ for all } i = 1, 2, ..., n \ , \tag{4.3}$$

where d_i is the i-th column vector of desired matrix D, then the linear programming method can map I_{real} into D. Again, there are many matrices satisfying Eq. (4.3). Hence, learning is transformed from finding a one-to-one mapping into one that finds a one-to-many mapping. The additional overhead incurred by an extra subnet involves solving a set of feasibility problems by linear programming. The procedure of the mixed-mode learning algorithm is summarized as follows.

Procedure 4.1: Mixed-Mode Learning Algorithm

Given a training pattern set (including an input matrix P and a desired output matrix D), a configuration of a multilayered feed-forward neural network and the sigmoid function used as the activation function, the procedure consists of four steps.

Step 1 :

Train the original network for one epoch by any existing supervised learning algorithm. If the original network converges, then stop; otherwise go to Step 2.

Step 2 :

Extract matrix I_{real} from the original network, where I_{real} (see Figure 4.1) is the input matrix to the output layer.

Step 3 :

Using the linear mapping or linear programming method to obtain the weights of

connections to the output layer.

Step 4 :

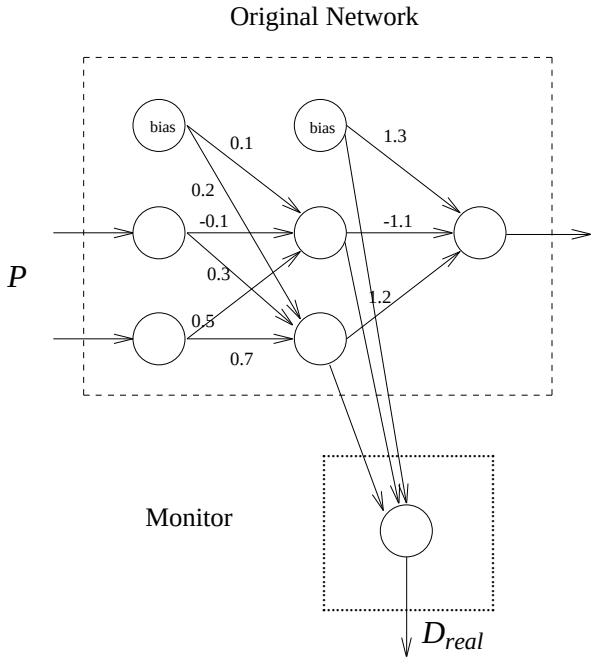
Test the criterion for learning to stop. If $\|D_{real} - D\|$ *is smaller than a prescribed threshold, then stop; otherwise go to Step 1*.

The following is a simple example to illustrate the procedure. The test problem used is the XOR problem.

Example 4.1. Consider a two-layer neural network with 2, 2, and 1 units in layers 1, 2, and 3, respectively, for solving the XOR problem. The input matrix P and desired output matrix D are

$$P = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \quad \text{and} \quad D = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} . \tag{4.4}$$

Using the back-propagation algorithm to train the original neural network, the result after one epoch of training is as follows:



The intermediate output matrix is, therefore,

$$I_{real} = \begin{bmatrix} 1.000 & 0.525 & 0.550 \\ 1.000 & 0.646 & 0.711 \\ 1.000 & 0.500 & 0.622 \\ 1.000 & 0.622 & 0.769 \end{bmatrix} . \tag{4.5}$$

If the linear mapping method is used in step 3 of our mixed-mode learning algorithm, we have to solve the following equation

$$I_{real} \cdot W^{2,3} = D_0 = \begin{bmatrix} -10 \\ 10 \\ 10 \\ -10 \end{bmatrix} , \tag{4.6}$$

and obtain the least-squared solution $W^{2,3+}$, where

$$W^{2,3+} = \begin{bmatrix} -106.7 \\ 254.5 \\ -67.08 \end{bmatrix} . \tag{4.7}$$

The actual output matrix D_{real} is calculated as follows:

$$D_{real} = sigmoid\,(\, I_{real} \cdot W^{2,3+}\,) \approx \begin{bmatrix} 0 \\ 1 \\ 0.5 \\ 0.5 \end{bmatrix} . \tag{4.8}$$

In this case, $\| D_{real} - D \|$ is large. All of the steps are then repeated until $\| D_{real} - D \|$ is smaller than a prescribed threshold.

If linear programming is used in step 3, we have to find a feasible solution for the following set of inequalities.

$$\begin{bmatrix} -1.000 & -0.525 & -0.550 & 1.000 & 0.525 & 0.550 \\ 1.000 & 0.646 & 0.711 & -1.000 & -0.646 & -0.711 \\ 1.000 & 0.500 & 0.622 & -1.000 & -0.500 & -0.622 \\ -1.000 & -0.622 & -0.769 & 1.000 & 0.622 & 0.769 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ y_0 \\ y_1 \\ y_2 \end{bmatrix} \geq \begin{bmatrix} 0.41 \\ 0.41 \\ 0.41 \\ 0.41 \end{bmatrix} , \tag{4.9}$$

and obtain

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ y_0 \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0.0 \\ 0.0 \\ 9.40 \\ 2.57 \\ 5.73 \\ 0.0 \end{bmatrix}, \tag{4.10}$$

so that

$$W^{2,3} = \begin{bmatrix} x_0 - y_0 \\ x_1 - y_1 \\ x_2 - y_2 \end{bmatrix} = \begin{bmatrix} -2.57 \\ -5.73 \\ 9.40 \end{bmatrix}. \tag{4.11}$$

The actual output matrix D_{real} is calculated as follows:

$$D_{real} = sigmoid\,(\, I_{real} \cdot W^{2,3}\,) \approx \begin{bmatrix} 0.4 \\ 0.6 \\ 0.6 \\ 0.75 \end{bmatrix}. \tag{4.12}$$

Again, $\| D_{real} - D \|$ is large, and all of the steps are repeated until $\| D_{real} - D \|$ is smaller than a prescribed threshold.

4.2. Improved Mixed-Mode Supervised Learning Algorithm

As shown in Figure 4.1, time is wasted in obtaining the redundant weight matrix W'. The objective of our improved version of the mixed-mode learning algorithm (IMM) is to use heuristics to eliminate the overhead of computing W'.

4.2.1. Description of the IMM learning algorithm

In IMM, the original network is decomposed into two subnets: a nonlinear and a linear (see Figure 4.2). The linear subnet includes the output layer and the connections to the output layer, and is equivalent to a single-layer network. Its input matrix is I_{real} (see Figure 4.2). Assuming that the linear mapping method is used, then obtaining the weights for this linear subnet is equivalent to training a single-layer neural network, and the weights can be obtained by solving a set of equations $I_{real} \cdot W = D_0$.

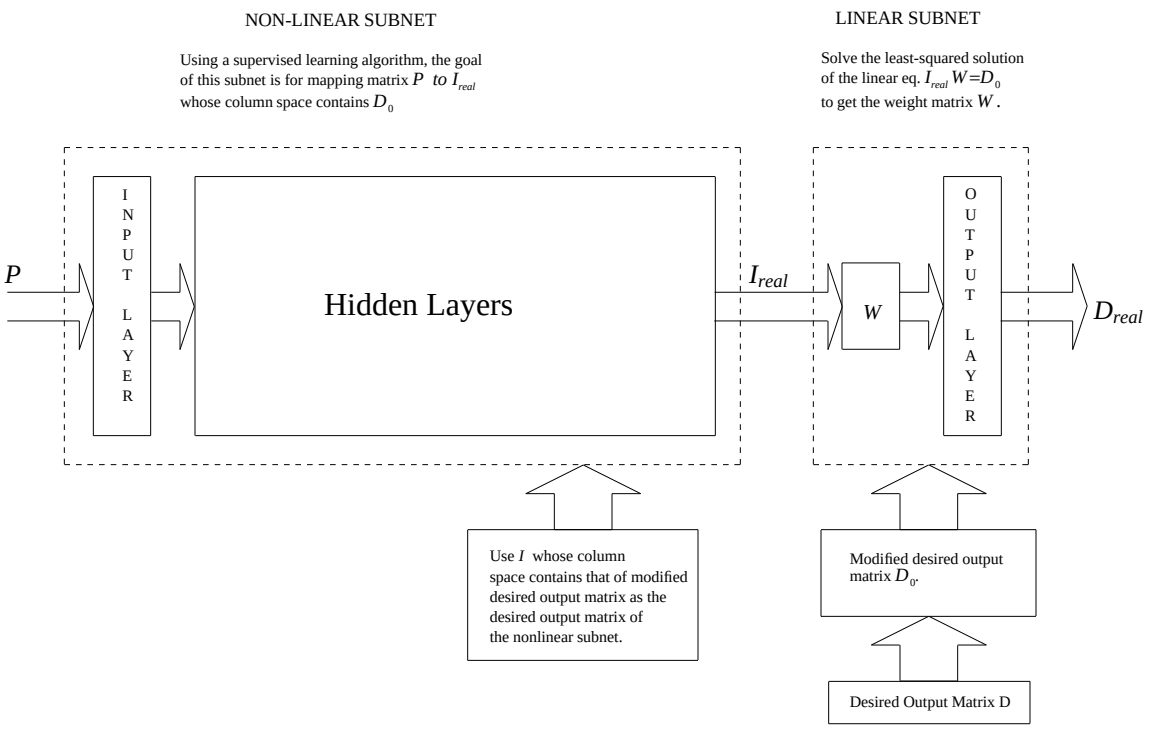


Figure 4.2 : Improved Mixed-Mode Supervised Learning Algorithm.

The nonlinear subnet includes the input layer, all hidden layers, and all connections among them. Its input is the input matrix used for training the original networks, and its output is the intermediate output matrix I_{real} of the original network. The objective of this subnet is to obtain the weights of connections such that the column space of its output matrix I_{real} contains that of the modified desired output matrix D_0. This is a nonlinear subnet because is has non-linear operations. Any existing supervised learning algorithm, such as backpropagation, can be chosen to train this nonlinear subnet.

We have to choose a desired output matrix I to train the nonlinear subnet. There are two criteria for choosing matrix I. First, the value of every element of I should be between [0.0,1.0], since the sigmoid function is used as the activation function in the hidden units.

Second, the column space of I must contain that of the modified desired output matrix D_0, since we want to obtain an I_{real} whose column space contains the column space of the modified desired output matrix D_0 (see Eq. (4.1)). Once I_{real} is obtained, we can solve the set of linear equations $I_{real} \cdot W = D_0$ to obtain the weights in the linear subnet. There are many I matrices satisfying the criterion. The heuristics we used to set matrix I are described as follows.

If the desired output matrix is binary, then the desired matrix for training the nonlinear subnet is set as follows:

$$I = \left[\begin{array}{c|c} D & G \end{array}\right] = \left[\begin{array}{c|c} D & \begin{array}{ccccc} 0 & 0 & 0 & . & . \\ 1 & 0 & 0 & . & . \\ 1 & 1 & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ 1 & 1 & 1 & . & . \end{array} \end{array}\right]_{k \times l}, \tag{4.13}$$

where l is the number of hidden units and/or input units that are connected to the output layer. Since the column space of D contains that of D_0 (see Eq. (3.10)), it is guaranteed that the column space of I defined by Eq. (4.13) contains that of D_0. Further, every element in I is in the range [0.0, 1.0]. Hence, Eq. (4.13) satisfies the two criteria discussed above.

If the desired output is continuous between 0.0 and 1.0, then I is set as follows (see Eq. (3.12)).

$$I = \left[\begin{array}{c|c} Y & G \end{array}\right] = \left[\begin{array}{c|c} Y & \begin{array}{ccccc} 0 & 0 & 0 & . & . \\ 1 & 0 & 0 & . & . \\ 1 & 0 & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ 1 & 1 & 1 & . & . \end{array} \end{array}\right]_{k \times l}, \text{ where } Y_{i,j} = \frac{(D_0)_{i,j} + 3}{6}. \tag{4.14}$$

For the continuous case, there are two features in matrix Y. First, its column space contains that of the modified desired matrix D_0. Second, its elements have values in the range between [0.0,1.0]. Again, Eq. (4.14) satisfies the two criteria for setting I.

Note that we do not aim to obtain a specific output matrix for the nonlinear subnet, although a specific desired output matrix for the nonlinear subnet is defined here. Matrix I, as defined in Eqs. (4.13) and (4.14), is used as a temporary goal for training the nonlinear subnet. Learning in the nonlinear subnet can stop whenever its outputs can be mapped to D by the methods discussed in Chapter 3.

The detailed procedure of our improved mixed-mode learning algorithm based on linear mapping is summarized as follows. Example 4.2. illustrates how the algorithm works.

Procedure 4.2: Improved Mixed-Mode Learning Algorithm

Given a training pattern set (that includes input matrix P and desired output matrix D), a configuration of a multilayered feed-forward neural network and the sigmoid function used as the activation function, the procedure consists of six steps.

Step 1: Decompose the network into two subnets:

All connections except those between the last hidden layer and the output layer belong to the nonlinear subnet and are trained by an existing supervised learning algorithm. The other connections belong to the linear subnet (see Figure 4.2).

Step 2: Initial setting:

For training the nonlinear subnet, we have to set the desired nonlinear subnet output matrix I. First, we modify the desired output matrix D used for training the original network. If the desired outputs are binary, then we modify D to D_0 as follows:

$$(D_0)_{i,j} = \begin{cases} \tau & \text{if } (D)_{i,j}=1 \\ -\tau & \text{if } (D)_{i,j}=0 \end{cases}, \tag{4.15}$$

where τ is a large positive number, and

$$I = \left[\begin{array}{c|c} & \begin{array}{ccccc} 0 & 0 & 0 & . & . \\ 1 & 0 & 0 & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ 1 & 1 & 1 & . & . \end{array} \\ D & \\ & \end{array}\right]_{k\times l}, \tag{4.16}$$

where l is the number of connections to each output unit. On the other hand, if the desired outputs are continuous between 0.0 and 1.0, we modify D as follows:

$$(D_0)_{i,j} = \begin{cases} 3 & \text{if } (D)_{i,j} \geq S^{-1}(3) \\ -3 & \text{if } (D)_{i,j} \leq S^{-1}(-3) \\ S^{-1}((D)_{i,j}) & \text{otherwise} \end{cases}, \tag{4.17}$$

and matrix I is

$$I = \left[\begin{array}{c|c} & \begin{matrix} 0 & 0 & 0 & . & . \\ 1 & 0 & 0 & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ 1 & 1 & 1 & . & . \end{matrix} \\ Y & \\ & \end{array}\right]_{k \times l} \quad \text{where } Y_{i,j} = \frac{(D_0)_{i,j} + 3}{6} \ . \tag{4.18}$$

Step 3: Train the nonlinear subnet for one epoch:

Use P as the input matrix and I as the desired output matrix to train the nonlinear subnet for one epoch by any supervised learning algorithm.

Step 4: Calculate I_{real}, the actual output matrix of the nonlinear subnet.

Step 5: Calculate the weights of connections in the linear subnet:

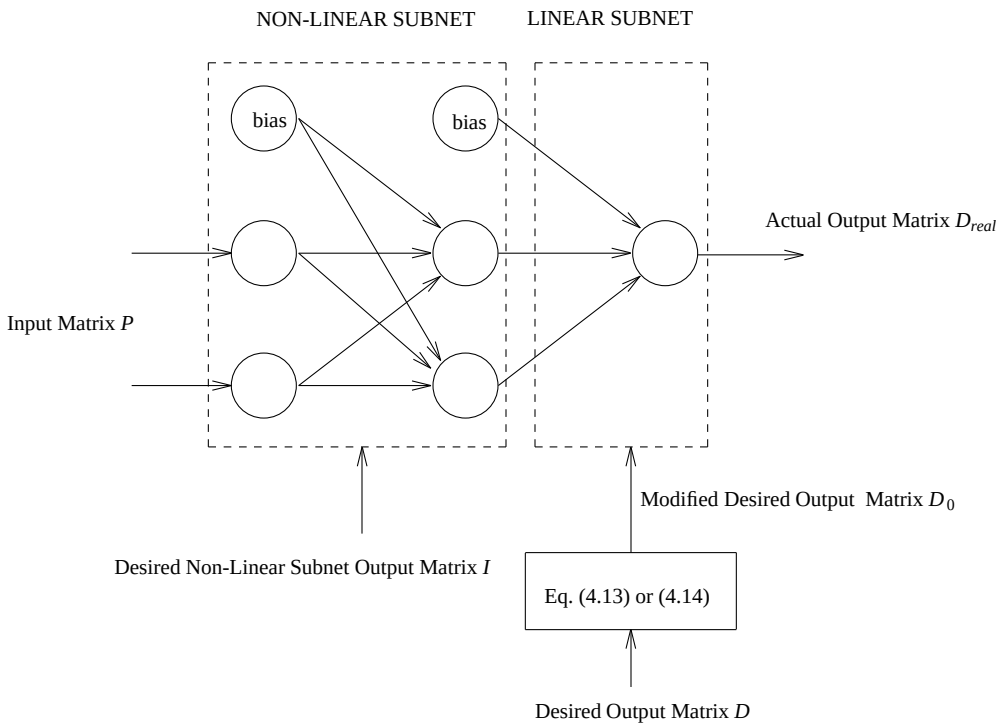
Solve the set of linear equations $I_{real} W = D_0$ to obtain the weights of connections to the output layer.

Step 6: Test the criterion for complete learning:

If the neural network can meet the criterion of complete learning, then stop; otherwise go to Step 3.

The above discussion only considers the linear mapping method. The linear programming method described in Chapter 3 can also be applied. Instead of using the linear mapping method in Step 5 of Procedure 4.2, the linear programming method can be employed here. Example 4.2 illustrates IMM when the linear mapping method is used.

Example 4.2. Consider a two-layer neural network with 2, 2, and 1 units in layers 1, 2, and 3, respectively, for solving the XOR problem. The network is first decomposed and the initial configuration is as follows:



$$P = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \quad I = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad D = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \text{ and } \quad D_0 = \begin{bmatrix} -10 \\ 10 \\ 10 \\ -10 \end{bmatrix} \tag{4.19}$$

Steps 3 and 4 of our IMM algorithm are executed. Assume that we have I_{real} as follows :

$$I_{real} = \begin{bmatrix} 1.0 & 0.6 & 0.1 \\ 1.0 & 0.4 & 0.2 \\ 1.0 & 0.2 & 0.3 \\ 1.0 & 0.8 & 0.5 \end{bmatrix} . \tag{4.20}$$

After solving the following equation to obtain $W^{2,3}$ (Step 5),

$$I_{real} \cdot W^{2,3} = \begin{bmatrix} 1.0 & 0.6 & 0.1 \\ 1.0 & 0.4 & 0.2 \\ 1.0 & 0.2 & 0.3 \\ 1.0 & 0.8 & 0.5 \end{bmatrix} \cdot W^{2,3} = D_0 \ , \tag{4.21}$$

we obtain the least-squared solution $W^{2,3+}$ as

$$W^{2,3+} = \begin{bmatrix} 18 \\ -43.3 \\ 13.33 \end{bmatrix} . \tag{4.22}$$

The actual output matrix D_{real} of the original network is then computed as

$$D_{real} = Sigmold\,(\, I_{real} \cdot W^{2,3+} \,) \approx \begin{bmatrix} 0.0 \\ 1.0 \\ 1.0 \\ 0.0 \end{bmatrix} . \tag{4.23}$$

In this case, $D_{real} \approx D$; hence, learning completes. On the contrary, if $\| D_{real} - D \|$ is very large, then return to Step 3 to continue learning.

4.2.2. Characteristics of the IMM learning algorithm

Compared with an existing learning algorithm, such as backpropagation, our improved mixed-mode learning algorithm has the following advantages.

(1) Our nonlinear subnet has a relaxed learning objective. Unlike other general supervised learning algorithms whose objective is to obtain an exact desired matrix, we do not have to obtain an exact desired output matrix for the nonlinear subnet. We merely set a target output matrix I in order to obtain learning started for the nonlinear subnet.

(2) Unlike Goggin's linear algorithm [9], our mixed-mode learning algorithm does not require a lot of hidden units. Our algorithm uses the nonlinear subnet to perturb the column space of the input matrix into any desired column space, even though the number of hidden units is very small. Goggin's algorithm basically uses two linear subnets. To train a two-layer neutral network, the algorithm chooses a proper desired hidden-layer output matrix, including its dimension and value; it then solves two sets of linear equations. If the column space of the input matrix $P_{k\times m}$ in the training set is too small, that is, $k \gg m$, then the number of hidden units must be increased to obtain a feasible mapping. This will cause overfitting, and the network memorizes the examples in the training set and learns a mapping too specific to the training set.

(3) From our experimental results, it shows that our learning algorithm can find sufficiently good learning results after very few training epochs. This makes our algorithm suitable for cases in which the network has to be retrained periodically.

(4) Our algorithm is very flexible: any existing learning algorithm, such as backpropagation or Quickprop [8] , can be applied to train the nonlinear subnet.

There is a drawback of our IMM learning algorithm. When a two-layer neural network is used, the nonlinear subnet is a single-layer neural network and is not universal [3]. In this case, we select a constant ρ such that in the first ρ training epochs, we apply our IMM learning algorithm. After ρ epochs, we switch to a regular supervised learning algorithm to train the entire network as a nonlinear network. This means that our IMM algorithm is used as a preprocessing step before a regular supervised learning algorithm is applied. Table 4.2 summarizes the usage of our IMM learning algorithm.

Table 4.2: Applying the IMM learning algorithm to multilayered neural networks.

Number of Hidden Layers	Usage
No hidden layer	Use linear subnet only.
One hidden layer	Use IMM as a preprocessing step.
Two more hidden layers	Use IMM learning as defined in Procedure 4.2.

4.3. Applications of MM and IMM Learning Algorithms

In Sections 4.1 and 4.2, we have discussed two variants of the mixed-mode learning algorithm. Both variants can be applied to reduce the number of learning epochs for fixed-topology supervised learning algorithms. This reduction is attributed primarily to the overhead incurred by the monitor in each epoch. As described in Sections 4.1 and 4.2, we have applied the linear mapping and linear programming methods in the monitor. Since the overhead caused by the linear mapping method is not significant, MM and IMM with linear mapping are quite efficient and show good potential for improving the physical learning time (over existing algorithms) in a number of applications. However, MM and IMM with the linear programming method

cannot be applied to fixed-topology cases. The overhead in solving a linear programming problem in each step is significant even though the number of epochs is reduced. Table 4.3 shows the physical CPU time of an epoch for training a 21-9-3 network with 300 patterns.

Table 4.3: Comparing run times between the linear mapping and linear programming methods.

Algorithm			Sec/Epoch
Quickprop	without monitor		0.13
	with monitor	linear mapping	0.27
		linear programming	1.96

For the cascade-correlation algorithm, the number of hidden units changes as the learning algorithm proceeds. As a result, our mixed-mode learning algorithms can be applied to reduce the number of hidden units. This allows the resulting network to generalize better and reduce the application time when the trained network is used on new input patterns. Figure 4.3 shows that the number of hidden units increases monotonically with the number of epochs in the process of applying the cascade-correlation algorithm. Hence, reducing the number of epochs by the above mixed-mode learning leads to a decrease of the number of hidden units in the resulting network.

The monitor is seldom activated in the cascade-correlation case. There are two training phases in the cascade-correlation algorithm: *TRAIN_INPUT* phase for adding new hidden units, and *TRAIN_OUTPUT* phase for training the weights in the output layer. These two phases are executed alternatively. If adding a monitor to the cascade-correlation algorithm, we note that (a) 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. Since the monitor is activated infrequently, the linear programming method can be used instead of linear mapping in the mixed-mode algorithms even though its computational time is large. The reason is that the solution qualities of the linear programming method are much better than those of the linear mapping method; hence, the linear programming method has a higher

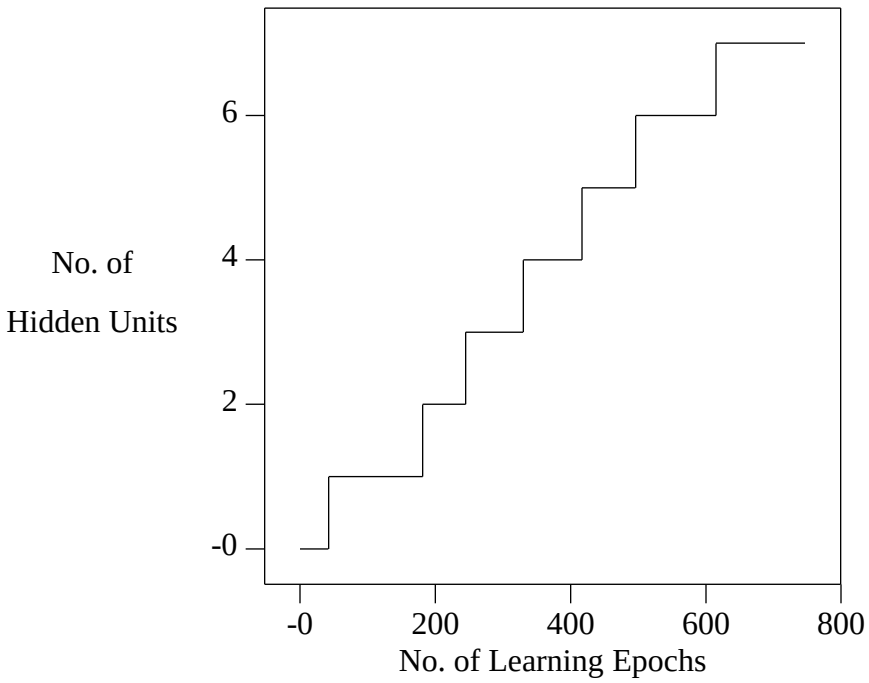


Figure 4.3 : Number of hidden units versus number of training epochs in the cascade-correlation learning algorithm.

---

probability of obtaining the solution than the linear mapping method. Further, the reduction in the number of hidden units in the cascade-correlation case is more important than saving learning time, since a trained ANN may have to be applied many times.

CHAPTER 5.

EXPERIMENTAL RESULTS

In this chapter, we present experimental results of the two mixed-mode supervised learning algorithms discussed in Chapter 4. In Section 5.1, we briefly introduce the benchmark problems used in this chapter. There are two kinds of benchmark problems tested in our experiments: those with binary outputs and those with continuous outputs between 0.0 and 1.0. In Section 5.2, we show that the mixed-mode learning algorithm with linear mapping can reduce the learning time for some existing supervised learning algorithms. We then show the convergence property of the mixed-mode learning algorithm in Section 5.3. Finally, we show that the mixed-mode learning algorithm with linear programming can reduce the number of hidden units required to converge for the cascade-correlation algorithm. We compare the performance of our learning algorithms with those of "pure" Quickprop ,"pure" backpropagation, and "pure" cascade-correlation algorithm. In addition to the number of epochs required for complete learning, we also list the CPU time incurred in each case. Table 5.1 shows the abbreviations used in this chapter.

5.1. Benchmarks Problems

In this section, we briefly introduce the benchmark problems used in this chapter. There are four benchmarks — the XOR problem, sonar problem, waveform problem, and two-spiral problem, with binary outputs — and one nonlinear function mapping problem with continuous outputs between 0.0 and 1.0.

5.1.1. The XOR problem

The task of the XOR problem is to train a network to produce the Boolean "Exclusive Or" function of two variables. This is perhaps the simplest learning problem that is not linearly separable. XOR has been a popular learning benchmark in recent literature. It is a special case of the parity function, but here we treat it as a separate benchmark in its own right. The input and desired output matrices have been listed in Eq. (4.4). For fixed-topology supervised

Table 5.1: Abbreviations used in this chapter.

Symbol	Meaning
BP	"pure" back-propagation
QP	"pure" Quickprop
CAS	"pure" cascade-correlation
IMM+BP	mixed-mode learning using BP to train the nonlinear subnet.
MM+BP	improved mixed-mode learning using BP to train the original network.
IMM+QP	mixed-mode learning using QP to train the nonlinear subnet.
MM+QP	improved mixed-mode learning using QP to train the original network.
IMM+CAS	mixed-mode learning using CAS to train the nonlinear subnet.
IMM+QP	improved mixed-mode learning using CAS to train the original network.

learning algorithms, the network configuration is a 2-2-1 neural network fully connected between adjacent layers with short-cut connections to the output layer.

5.1.2. Sonar problem

The data set of the sonar problem is provided by Gorman and Sejnowski [10] in their research on employing neural networks to classify sonar signals. The task here is to train a network to discriminate between sonar signals bounced off a metallic cylinder and those bounced off a roughly cylindrical rock. The input patterns consists of 60 numbers in the range 0.0 to 1.0. Each number represents the energy within a particular frequency band integrated over a

certain period of time. The output pattern (1,0) represents ROCK, and (0,1) represents METAL, respectively. There are 104 patterns in the training set. For fixed-topology supervised learning algorithms, the network configuration is a 60-10-2 neural network fully connected between adjacent layers with short-cut connections to the output layer.

5.1.3. Waveform problem

This problem is adapted from the waveform recognition problem. There are three classes of waveforms generated from a random convex combination of two of three *base waves* sampled at intervals with Gaussian noise added. The three base waves are represented as $(s_j(1), s_j(2),..., s_j(21))$, $j = 1, 2, 3$, where

$$s_1(i) = \begin{cases} -cos((i-1)\pi/6) & 1 \le i \le 13 \\ -1 & 14 \le i \le 21 \end{cases} \tag{5.1}$$

$$s_2(i) = \begin{cases} -1 & 1 \le i \le 4 \\ -cos((i-5)\pi/6) & 5 \le i \le 17 \\ -1 & 18 \le i \le 21 \end{cases} \tag{5.2}$$

$$s_3(i) = \begin{cases} -1 & 1 \le i \le 8 \\ -cos((i-9)\pi/6) & 9 \le i \le 21 \end{cases} \tag{5.3}$$

A waveform $(x_1, x_{2,...,} x_{21})$ is generated from the base waves as follows:

$$x_i = \begin{cases} \mu s_1(i)+(1-\mu)s_2(i)+\xi_i & \text{Class 1} \\ \mu s_2(i)+(1-\mu)s_3(i)+\xi_i & \text{Class 2} \\ \mu s_3(i)+(1-\mu)s_1(i)+\xi_i & \text{Class 3} \end{cases} \tag{5.4}$$

where μ is a random variable distributed uniformly on the interval [0,1] and ξ_i is an independent random variable with normal distribution $N(0,0.01)$. The training set has 300 patterns with 100 patterns for each class. For fixed-topology learning algorithms, the configuration used is a 21-9-3 neural network fully connected between adjacent layers with short-cut connections to the output layer.

5.1.4. Two-spiral problem

The task of the two-spiral problem is to learn to discriminate between two sets of training points that lie on two distinct spirals in the x-y plane. These spirals coil three times around the origin and around one another. This appears to be a very difficult task for backpropagation networks and their variations. The training set has 192 patterns with two sets of points, each with 97 members (three complete revolutions at 32 points per revolution, plus endpoints).

5.1.5. Nonlinear function mapping problem

Most of the benchmark classification problems for supervised learning have binary outputs. Neural networks can also be used for function mapping. In this case, the outputs are continuous values. As an example, consider the nonlinear mapping of 8 inputs, x_i, $i = 1, 2, ..., 8$, into 3 outputs y_j, $j = 1, 2, 3$, where

$$y_1 = \frac{(x_1 x_2 + x_3 x_4 + x_5 x_6 + x_7 x_8)}{4} \tag{5.5}$$

$$y_2 = \frac{(x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8)}{8} \tag{5.6}$$

$$y_3 = (1 - y_1)^{0.5} \tag{5.7}$$

The training set includes 50 training patterns. All of the x_i's are randomly selected between [0.0, 1.0]. The desired y_i can be calculated from the above equations.

5.2. Experimental Results on Reduction of Learning Time

In this section, we show experimental results on learning times for the MM and IMM algorithms with the linear mapping method, and compare them with the performance of some existing supervised learning algorithms. First, we test three benchmarks, XOR problem, sonar problem, and WAVEFORM problem, with binary outputs. For uniformity in reporting results, the 40-20-40 criterion is applied here: that is, if an output is larger than 0.6, we consider it as a logic ONE, and if an output is smaller than 0.4, then we consider it as a logic ZERO. Quickprop and backpropagation, respectively, are plugged into two mixed-mode learning algorithms

with linear mapping. Every learning algorithm had thirty trials with different initial weights. To avoid over-training the neural networks, training stops when a network can classify correctly 95% of the patterns in the training set. The statistical results are shown in Tables 5.2 and 5.3, respectively. For the XOR problem, results show that our mixed-mode supervised learning algorithms, regardless of whether they are IMM or MM, can almost converge after only one training epoch. The reason is that the rank of the column space of the intermediate output matrix I_{real} (a 5-by-4 matrix) is usually equal to four, and span$\{I_{real}\}$ contains the column space of D_0. In this case, the set of linear equations $I_{real} \cdot W = D_0$ will have an exact solution and training completes. I_{real}, in the sonar problem, is a 104-by-71 matrix. Since the possibility of this I_{real} to contain the column space of D_0 is small, the network usually cannot converge after only one epoch of training as in the XOR problem. However, our results show that MM and IMM still have better learning performances than those of BP and QP. Note that IMM is slightly faster than MM, since IMM does not waste time in computing the redundant weight matrix in the output layer.

The waveform problem is more difficult than the sonar and the XOR problems. This is true because the rank of I_{real} is relatively small, and it is hard to find an exact solution for the set of linear equations $I_{real} \cdot W = D_0$. When solving the waveform problem, we found that the maximal learning times of IMM and MM are larger than those of BP and Quickprop, although the maximum learning epochs of IMM and MM are smaller than those of BP and Quickprop. This happens because there exists overhead for solving the set of linear equations in the mixed-mode learning. However, the average learning times of of IMM and MM are still smaller than those of BP and Quickprop. Figure 5.1 compares the fraction of 30 converged nets as a function of learning time for networks trained to solve the waveform problem shown in Table 5.3. It illustrates that most of the networks trained by the mixed-mode algorithm converge faster that those trained by QP, although the maximum learning times of MM and IMM are larger than that of QP.

To ensure that the trained neural networks work properly, when training stops, we test the resulting network by a test set whose patterns are different from those in training set. The average correct rate of classifying patterns of the test set in the sonar and waveform problems is shown in Table 5.4.

Table 5.2: Performances of QP, and MM and IMM based on QP.

Problem	Performance				
	measures		QP	MM+QP	IMM+QP
XOR	Epochs	avg	44.93	1.17	1.19
		max	304	3	3
		min	21	1	1
		std dev	50.48	0.461	0.41
	Time (sec)	avg	0.031	8.e-4	7.e-4
		max	0.18	2.e-3	2.e-3
		min	0.01	7.e-4	6.e-4
		std dev	0.03	3.e-4	3.e-4
SONAR	Epochs	avg	366.3	74.27	66.51
		max	818	564	464
		min	131	1	1
		std dev	189.9	116.2	102.88
	Time (sec)	avg	47.89	19.46	13.96
		max	106.4	147.9	97.44
		min	17.05	0.26	0.21
		std dev	24.73	30.47	21.60
WAVE	Epochs	avg	221.5	66.20	68.9
		max	442	409	374
		min	105	8	7
		std dev	77.21	79.74	76.95
	Time (sec)	avg	28.87	17.69	15.16
		max	59.07	107.9	82.28
		min	28.87	2.29	1.54
		std dev	11.48	21.07	16.93

Table 5.3: Performances of BP, and MM and IMM based on BP.

Problem	Performance				
	measures		BP	MM+BP	IMM+BP
XOR	Epochs	avg	182.3	1	1
		max	508	1	1
		min	73	1	1
		std dev	81.92	0.0	0.0
	Time (sec)	avg	0.15	8.e-4	7.e-4
		max	0.9	8.e-4	7.e-4
		min	0.04	8.e-4	7.e-4
		std dev	0.184	0.0	0.0
SONAR	Epochs	avg	245.9	32.13	31.53
		max	750	218	241
		min	94	1	1
		std dev	145.7	44.04	38.76
	Time (sec)	avg	42.73	11.30	10.38
		max	129.4	75.73	74.31
		min	16.26	0.513	0.432
		std dev	25.07	15.26	15.27
WAVE	Epochs	avg	116.5	47.9	48.86
		max	177	135	144
		min	95	9	13
		std dev	17.02	37.56	34.53
	Time (sec)	avg	22.68	18.70	18.08
		max	34.07	52.29	53.28
		min	18.61	3.67	4.81
		std dev	3.23	14.49	12.77

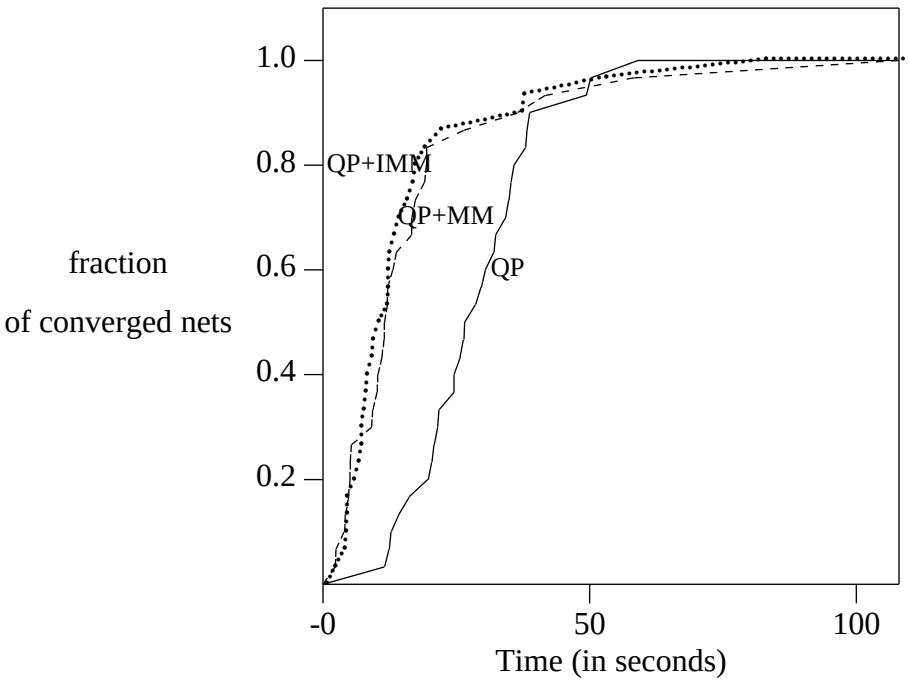


Figure 5.1 : Cumulative distribution of fraction of converged nets for nets trained by QP, QP+IMM, and QP+MM for solving the waveform problem.

Table 5.4 shows that neural networks trained by our mixed-mode supervised learning algorithms have almost the same correct rate as those trained by BP or QP. This implies that ANNs learned by our mixed-mode learning algorithm have equal generalization power as those learned by BP or QP.

Table 5.4 : Average correct rate in testing the ANN for the sonar and waveform problem.

Algorithm		BP	MM+BP	IMM+BP	QP	MM+QP	IMM+QP
Correct	SONAR	90.8%	90.2%	89.3%	89.5%	87.3%	89.8%
	WAVE	93.8%	91.2%	88.3%	91.5%	90.8%	88.7%

In addition to the problems with binary outputs, we have also tested the function mapping problem with continuous outputs. Learning stops when $\|y - y_{real}\| / \|y\| \leq 0.9$. Again, each algorithm had 30 trials. Tables 5.5 and 5.6 show the experimental results. These results also show that the mixed-mode learning algorithm performs better than the BP and QP.

After comparing MM and IMM with some fixed-topology learning algorithms, the cascade-correlation algorithm is then plugged into our mixed-mode learning algorithm with linear mapping. Again, each learning algorithm had thirty trials with different initial weights. The statistical results are shown in Table 5.7. We see that either IMM or MM does not improve the learning time significantly and, in some cases, results in increased learning time. This happens because the number of hidden units connected to the output layer is small, resulting in an I_{real} with a small number of columns. Note that in the XOR problem, the results of MM+CAS and IMM+CAS are the same. This is true because every network converges in the first epoch after adding the first hidden unit.

5.3. Convergence Property of the Mixed-Mode Supervised Learning

A nice characteristic of our mixed-mode learning algorithm is that it can converge very quickly to a good solution. We use a simple classification problem to show this feature (Figure 5.2). The training set has 200 patterns that are randomly selected. The neural network configuration used here is 2-7-7-1. Backpropagation is used to train the original network. The error traces in the learning phase for the two learning algorithms are shown in Figure 5.3.

After 10 epochs of training, we tested the two neural networks trained by backpropagation and IMM with the linear mapping method. The results are shown in Figure 5.4. We see that the network trained by IMM can classify much better than that trained by backpropagation. Although the difference in speedup for complete learning between our mixed-mode algorithm and backpropagation is around 1.5 to 5.0, our algorithm can result in a network with fewer errors in a small number of training epochs. This feature is very useful when a neural network has to be retrained periodically.

Table 5.5 : Performances of MM+QP, IMM+QP and QP for solving the nonlinear mapping problem.

Problem	Performance				
	results		QP	MM+QP	IMM+QP
Mapping	Epochs	avg	317.8	27.76	22.65
		max	483	98	81
		min	144	21	20
		std dev	15.34	17.32	16.31
	Time (sec)	avg	9.74	2.82	1.23
		max	15.89	6.14	5.10
		min	4.54	1.34	0.97
		std dev	3.21	2.02	1.25

Table 5.6 : Performances of MM+BP, IMM+BP and BP for solving the nonlinear mapping problem.

Problem	Performance				
	results		BP	MM+BP	IMM+BP
Mapping	Epochs	avg	429.4	34.7	30.2
		max	690	121	108
		min	206	25	19
		std dev	20.13	27.53	24.16
	Time (sec)	avg	13.92	2.26	1.60
		max	22.38	7.87	5.7
		min	6.68	1.62	1.01
		std dev	4.13	2.15	1.34

Table 5.7: Performances of MM+CAS, IMM+CAS and CAS.

Problem	Performance				
	measures		CAS	MM+CAS	IMM+CAS
XOR	Epochs	avg	54.14	43.0	43.0
		max	67	50	50
		min	47	37	37
		std dev	3.08	3.276	3.276
	Time (sec)	avg	0.024	0.01	0.01
		max	0.03	0.02	0.02
		min	0.01	1.e-3	1.e-3
		std dev	7e-3	6e-3	6e-3
SONAR	Epochs	avg	389.6	277.7	283.4
		max	554	491	503
		min	298	218	207
		std dev	69.63	63.74	70.31
	Time (sec)	avg	6.16	5.94	6.11
		max	9.97	10.81	11.21
		min	4.36	4.47	4.52
		std dev	1.38	1.46	1.72
WAVE	Epochs	avg	491.7	532.5	731.2
		max	618	783	1002
		min	406	319	353
		std dev	52.68	143.99	181.3
	Time (sec)	avg	11.88	14.61	19.98
		max	14.60	20.33	27.3
		min	9.55	9.20	9.53
		std dev	1.37	3.69	4.31

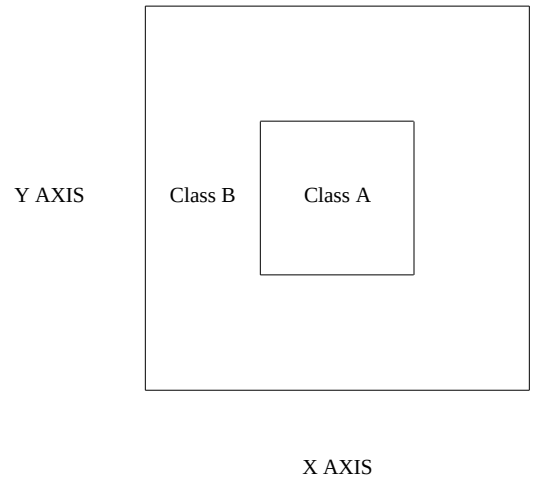


Figure 5.2. A simple classification problem.

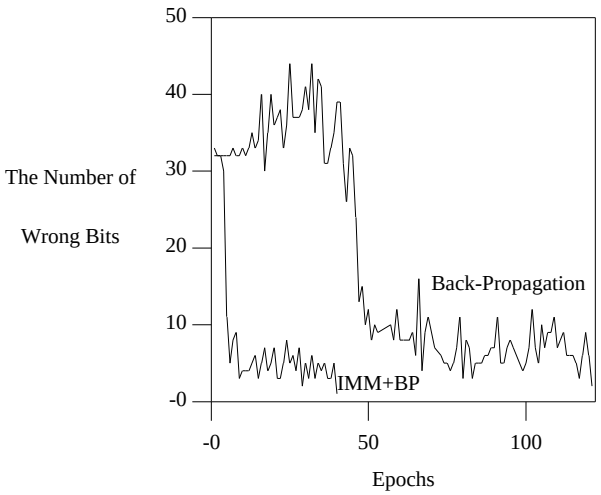
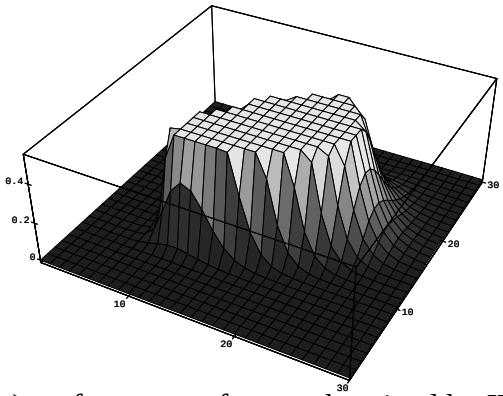


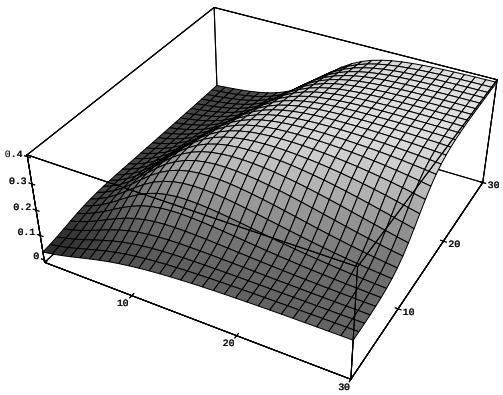
Figure 5.3. Convergence of networks trained by BP and IMM+BP.

5.4 Experimental Results on Reduction of Hidden Units

In this section, we show that MM with linear programming can reduce the number of hidden units required for convergence in the cascade-correlation algorithm. We compare the performance of the original cascade-correlation algorithm with that of MM using linear



(a) performance of network trained by IMM+BP



(b) performance of network trained by BP

Figure 5.4. Test results after 10 epochs of learning.

---

programming. The two-spiral problem is tested here. Again, each learning algorithm had thirty trials with different initial weights. Figure 5.5 shows a plot of the number of hidden units versus learning time. Each point in this figure is normalized with respect to the performance of the "pure" cascade-correlation algorithms, that is,

$$\left(\frac{\text{No. of hidden units of a net trained by MM}}{\text{No. of hidden units of a net trained by CAS}}, \frac{\text{learning time of net trained by MM}}{\text{learning time of net trained by CAS}} \right) \quad (5.8)$$

For the convenience of reading our experimental data, we assume that the joint distribution of the normalized number of hidden units and the normalized learning time is a bivariate normal

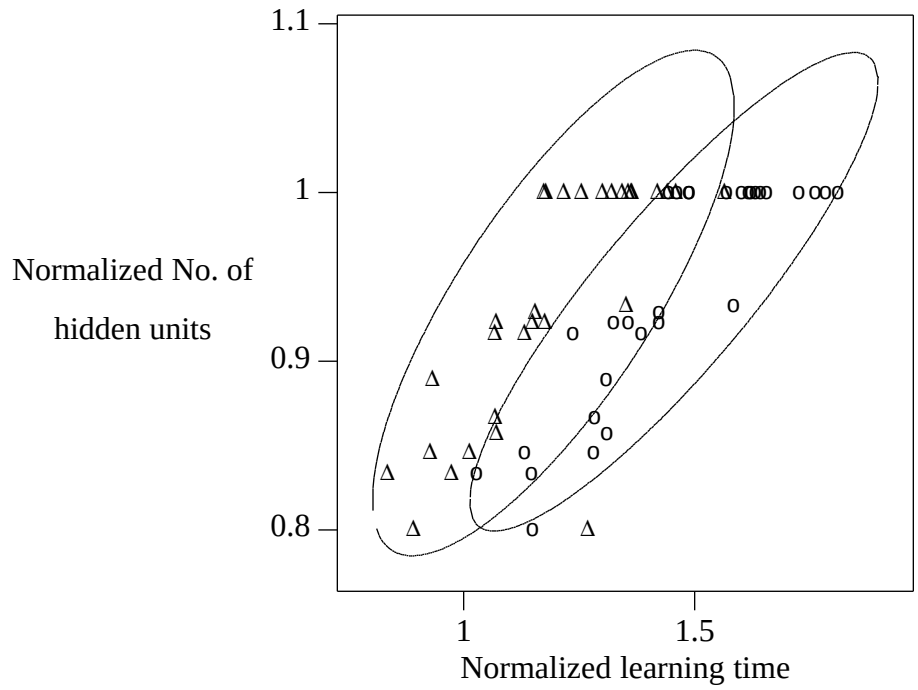


Figure 5.5 : Performances of CAS and MM+CAS for solving the two-spiral problem.

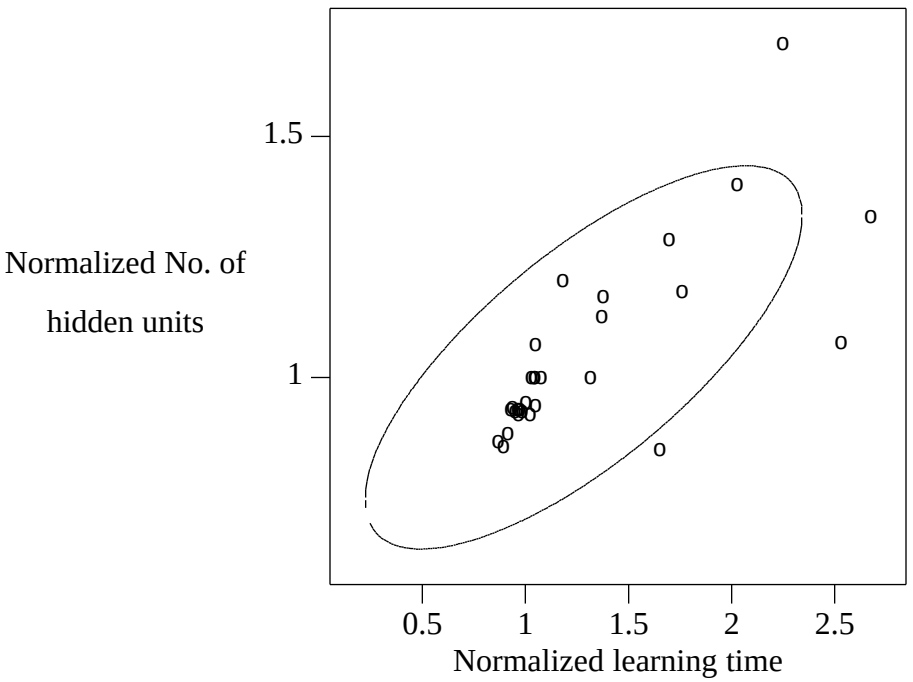


Figure 5.6 : Performances of CAS and IMM+CAS for solving the two-spiral problem.

distribution. Based on the experimental data, we then plot the 90% constant probability density contour of this bivariate normal distribution [11]. If the assumption of the bivariate normal distribution is true, then the probability of experimental data inside the ellipse is 90%. The results show that, in most trials, MM+CAS can converge with less hidden units, although the learning time is longer. The reduction in the number of hidden units is important, since a trained ANN may have to be applied many times in the future.

During the experiments, we found that it is not necessary to activate the monitor early in the process, since it is difficult for the monitor to find an I_{real} in the initial stage of learning to be mapped to the desired output matrix D. In Figure 5.5, points indicated by circles are obtained when the monitor is always activated. Points indicated by triangles are obtained in cases for which the monitor is activated when the error in the original network is smaller than 20%. We see that the latter case can result in savings in learning time. Note that the number of units of networks trained by MM is never larger than those trained by the cascade-correlation algorithm, since learning completes when either CAS completes or MM finds a suitable mapping.

Next, we plug the cascade-correlation algorithm into IMM with linear programming. Figure 5.6 shows the results. We can see that the result is not good as for MM, since the heuristics using in the linear programming method are not well-tuned.

CHAPTER 6

CONCLUSIONS

In this thesis, we have studied two new supervised learning algorithms. They use a monitor to extract outputs from neurons in hidden layers that are connected to the output layer and employ either the linear mapping method or the linear programming method to transform the intermediate outputs into desired outputs. Our learning methods transform supervised learning from a one-to-one mapping to a one-to-many mapping. That is equivalent to relaxing the objective of learning and making it easier. From our experimental results, we conclude as follows.

(1) The error in our learning method converges very quickly after a few training epochs. This feature is very useful when ANNs have to be retrained periodically in real-time applications.

(2) Our proposed learning method is flexible and can adapt to many existing supervised learning algorithms. For fixed-topology supervised learning algorithms, such as backpropagation and Quickprop, we can combine MM or IMM with the linear mapping method to improve their learning times. For the cascade-correlation algorithm, we can combine MM with linear programming to reduce the number of hidden units requireded for convergence.

[1] L. W. Chan and F. Fallside, ‘‘An Adaptive Training Algorithm for Backpropagation Networks,’’ *Computer Speech and Language*, vol. 2, pp. 205-218, Academic Press, London, 1987.

[2] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms* , MIT Press, 1990 .

[3] G. Cybenko, ‘‘Approximation by Superpositions of a Sigmoidal Function,’’ *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303-314, Springer International, New York, 1989.

[4] C. Darken and J. Moody, ‘‘Notes on Learning Rate Schedules for Stochastic Optimization,’’ *Neural Information Processing System*, pp. 832-838, 1991.

[5] Harry A. C. Eaton and Tracy L. Olivier, ‘‘Learning Coefficient Dependence on Training Set Size,’’ *Neural Networks*, pp. 283-288, Pergamon Press, Elmsford, NY, 1992.

[6] 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.

[7] S. E. Fahlman and Christian Lebiere, The Cascade-Correlation Learning Architecture (CMU-CS-90-100), School of Computer Science, Carnegie Mellon Univ., Pittsburgh, PA, Feb. 1990.

[8] S. E. Fahlmann, *An Empirical Study of Learning Speed in Back-Propagation Networks*, Carnegie Mellon University, Pittsburgh, PA, 1988.

[9] S. D. D. Goggin, K. E. Gustafson, and K. M. Johonson, ‘‘An Asymptotic Singular Value Decomposition Analysis of Nonlinear Multilayer Neural Networks,’’ *Int’l Joint Conf. on Neural Networks*, Seattle, WA, July 1991.

[10] R. P. Gorman and T. J. Sejnowski, ‘‘Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets,’’ *Neural Networks*, vol. 1, pp. 75-89, Pergamon Press, Elmsford, NY, 1988.

[11] A. Ieumwananonthachai, A. Aizawa, S. R. Schwartz, B. W. Wah, and J. C. Yan, ‘‘Intelligent Process Mapping Through Systematic Improvement of Heuristics,’’ *J. of Parallel and Distributed Computing*, vol. 15, pp. 118-142, Academic Press, June 1992.

[12] J. K. Kruschke, ‘‘Creating Local and Distrtibuted Bottlenecks in Hidden Layers of Back-Propagation Networks,’’ *Proc. Connectionist Models Summer School*, pp. 120-126, Morgan Kaufmann, Palo Alto, CA, 1988.

[13] R. P. Lippmann, ‘‘An Introduction to Computing with Neural Nets,’’ *Accoustics, Speech and Signal Processing Magazine*, vol. 4, no. 2, pp. 4-22, IEEE, April 1987.

[14] P. Mehra and B. W. Wah (ed.), *Artificial Neural Networks: Concepts and Theory*, IEEE Computer Society Press, ISBN 0-8186-8997-8, Los Alamitos, CA, 1992.

[15] M. C. Mozer and P. Smolensky, ‘‘Using Relevance to Reduce Network Size Automatically,’’ *Connection Science*, vol. 1, no. 1, pp. 3-16, Carfax Pub. Co., Oxfordshire, United Kingdom, 1989.

[16] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, in *Numerical Recipes in C*, Cambridge University Press, 1988.

[17] M. Riedmiller and H. Braun, ‘‘RPROP - A Fast Adaptive Learning,’’ Technical Report, University of Karlsruhe, Karlsruhe, Germany, 1992.

[18] 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.

[19] J. Sietsma and R. J. F. Dow, ‘‘Neural Net Pruning--Why and How?,’’ *Int’l Joint Conf. on Neural Networks*, vol. 1, pp. 325-333, IEEE, 1988.

[20] F. M. Silva and L. B. Almeida, ‘‘Speeding up Backpropagation,’’ *Advanced Neural Computers*, pp. 151-158, North-Holland , New York , 1990.

[21] G. Smith, *Back Propagation with Dynamic Topology and Simple Activation Functions*, Tech. Rep. TR 90-12, School of Information Science and Technology, The Flinders Univ. of South Australia, Adelaide, 1990.

[22] T. Tollenaere, ‘‘SuperSAB : Fast Adaptive Backpropagation with Good Scaling Properties,’’ *Neural Networks*, vol. 3, pp. 561-573, Pergamon Press, Elmsford, NY, 1990.