Parameter effects on convergence speed and generalization capability of backpropagation algorithm

CHUAN ZHANG TANG† and HON KEUNG KWAN†

The convergence speed of learning and the generalization capability are the two most important properties of the backpropagation (BP) algorithm. The effects of a number of modifications of BP on these two features are investigated; a new scheme of backpropagation combined with an adaptive slope of the activation function and the adaptive bias of the neuron is proposed. The new scheme has advantages in both convergence speed and generalization capability. Simulation results are provided.

1. Introduction

Feedforward structure has been an important model proposed for artificial neural networks and has found many applications. The backpropagation algorithm (Rumelhart and McClelland 1986, Widrow and Lehr 1990) is still a very popular and powerful learning scheme for feedforward networks. However, in some cases, the basic backpropagation algorithm takes an intolerable time to converge to a predetermined error level and is quite sensitive to the initial weights of a network (Lee et al. 1991, Nguyen and Widrow 1990). These drawbacks have to be solved before the feedforward neural networks can find further practical applications. This has motivated many researchers to search for improvements. A number of modification schemes have been proposed (Kruschke and Movellan 1991, Minai and Williams 1990, Schreibman and Norris 1990). Almost all of these schemes have focused on the improvement of the convergence speed of backpropagation. However, the effects of these schemes on the generalization capability, another important property of feedforward neural networks, has not been considered. The generalization capability is an important feature of artificial neural networks which should not be ignored (Whitley and Karunanithi 1991, Mehrotra et al. 1991). In this paper, we will first analyse the effects of a number of parameters on both the convergence speed and the generalization capability and then propose a new scheme for implementing the backpropagation algorithm, which shows advantages in both convergence speed and generalization capability.

2. Backpropagation and its modifications

2.1. Standard backpropagation

For the feedforward neural network shown in Fig. 1, the net input to the jth neuron at the kth layer related to the kth pattern takes the form of

\[ z_{jk}^{(h)} = \sum_{i=1}^{N_{k-1}} w_{ij}^{(h)} x_{ik}^{(h-1)} + \theta_{j}^{(h)} \quad \text{for } h = 1, 2, \ldots, L \]  

Received 28 July 1992; accepted 28 July 1992.

†Department of Electrical Engineering, University of Windsor, 401 Sunset Avenue, Windsor, Ontario, Canada N9B 3P4.

0020–7217/93 $10.00 © 1993 Taylor & Francis Ltd.
where $y_{ik}^{[h-1]}$ is the activation of the $i$th neuron at the $(h-1)$th layer, $w_{ij}^{[h]}$ is the connection weight between neuron $i$ at the $(h-1)$th layer and neuron $j$ at the $h$th layer, $\theta_j^{[h]}$ is the threshold value of the $j$th neuron at the $h$th layer, and $N_h$ is the number of neurons at the $h$th layer. For $h=1$, $y_{ik}^{[h-1]} = x_{ik}$, where $x_{ik}$ is the $i$th element of the input pattern $k$.

Applying a non-linear activation function $F(z)$ to the net input yields the output of neuron $j$ at layer $h$ as

$$y_{jk}^{[h]} = F(z_{jk}^{[h]})$$  \hspace{1cm} (2)

Thus the output of the network is

$$y_{jk}^{[L]} = F(z_{jk}^{[L]})$$  \hspace{1cm} (3)

In general this activation does not match the desired output, and the error is defined as
Parameter effects on backpropagation algorithm

\[ e_{jk} = (1/2)(t_{jk} - y_{jk}^{[L]})^2 \]  \hspace{1cm} (4)

where \( t_{jk} \) represents the \( j \)th component of a target pattern \( k \). The total output error related to a pattern \( k \) is the sum of errors at each output neuron and can be expressed as

\[ e_k = (1/2) \sum_{j=1}^{N_L} (t_{jk} - y_{jk}^{[L]})^2 \]  \hspace{1cm} (5)

The back propagation is a gradient descent algorithm which allows the updating of the weights of a feedforward network. The idea of gradient descent is to make the change in a weight proportional to the negative derivative of the error with respect to that weight. Hence, the change in weight between the neuron \( i \) of the \((h-1)\)th layer and the neuron \( j \) of the \( h \)th layer is

\[ \delta_k w_{ij}^{[h]} = -\epsilon \frac{\partial e_k}{\partial w_{ij}^{[h]}} \quad \text{for} \quad 1 \leq h \leq L \]  \hspace{1cm} (6)

where \( \epsilon \) is a learning rate parameter.

In general, the derivative part of the above equation can be expressed as

\[ \frac{\partial e_k}{\partial w_{ij}^{[h]}} = \frac{\partial e_k}{\partial z_{jk}^{[h]}} \frac{\partial z_{jk}^{[h]}}{\partial w_{ij}^{[h]}} \]

\[ = \frac{\partial e_k}{\partial z_{jk}^{[h]}} \frac{\partial}{\partial w_{ij}^{[h]}} \left[ \sum_{i=1}^{N_{h-1}} w_{ij}^{[h]} y_{ik}^{[h-1]} + \theta_{ij}^{[h]} \right] \]

\[ = -\delta_{jk}^{[h]} y_{ik}^{[h-1]} \]  \hspace{1cm} (7)

where

\[ \delta_{jk}^{[h]} = -\frac{\partial e_k}{\partial z_{jk}^{[h]}} \]  \hspace{1cm} (8)

With these derivations, we have

\[ \delta_k w_{ij}^{[h]} = \epsilon \delta_{jk}^{[h]} y_{ik}^{[h-1]} \]  \hspace{1cm} (9)

Considering the weight-update at the \( L \)th or the output layer, we have

\[ \delta_{jk}^{[L]} = -\frac{\partial e_k}{\partial y_{jk}^{[L]}} \frac{\partial y_{jk}^{[L]}}{\partial z_{jk}^{[L]}} \]

\[ = (t_{jk} - y_{jk}^{[L]}) F'(z_{jk}^{[L]}) \]  \hspace{1cm} (10)

where \( F'(z_{jk}^{[L]}) \) represents the partial derivative of \( F(z_{jk}^{[L]}) \) with respect to \( z_{jk}^{[L]} \). Hence, the weight-update at the output layer is

\[ \delta_k w_{ij}^{[L]} = \epsilon \delta_{jk}^{[L]} y_{ik}^{[L-1]} \]

\[ = \epsilon (t_{jk} - y_{jk}^{[L]}) F'(z_{jk}^{[L]}) y_{ik}^{[L-1]} \]  \hspace{1cm} (11)

For the \((L-1)\)th layer,
\[ \delta_{jk}^{[L-1]} = -\frac{\partial e_k}{\partial z_{jk}^{[L-1]}} = - \sum_{i=1}^{N_k} \frac{\partial e_k}{\partial z_{ik}^{[L]}} \frac{\partial z_{ik}^{[L]}}{\partial z_{jk}^{[L-1]}} \]
\[ = - \sum_{i=1}^{N_k} \frac{\partial e_k}{\partial z_{ik}^{[L]}} \frac{\partial z_{ik}^{[L]}}{\partial y_{jk}^{[L-1]}} \frac{\partial y_{jk}^{[L-1]}}{\partial z_{jk}^{[L-1]}} \]
\[ = - F'(z_{jk}^{[L-1]}) \sum_{i=1}^{N_k} \frac{\partial e_k}{\partial z_{ik}^{[L]}} \frac{\partial y_{jk}^{[L-1]}}{\partial y_{ik}^{[L-1]}} \frac{\partial y_{ik}^{[L-1]}}{\partial y_{ji}^{[L-1]}} \frac{\partial y_{ji}^{[L-1]}}{\partial y_{ij}^{[L-1]}} \]
\[ = F'(z_{jk}^{[L-1]}) \sum_{i=1}^{N_k} \frac{\partial e_k}{\partial y_{ik}^{[L]}} \frac{\partial y_{ik}^{[L]} + \theta_i^{[L]}}{\partial y_{ji}^{[L-1]}} \]
\[ = F'(z_{jk}^{[L-1]}) \sum_{i=1}^{N_k} \delta_{ik}^{[L]} w_{ji}^{[L]} \] (12)

Similarly, we can obtain
\[ \delta_{jk}^{[h]} = F'(z_{jk}^{[h]}) \sum_{i=1}^{N_k} \delta_{ik}^{[h+1]} w_{ji}^{[h+1]} \] for \( 1 \leq h \leq L - 1 \) (13)

In summary, we have
\[ \delta_{jk}^{[h]} = F'(z_{jk}^{[h]}) \sum_{i=1}^{N_k} \delta_{ik}^{[h+1]} w_{ji}^{[h+1]} \] for \( 1 \leq h \leq L - 1 \)
\[ \delta_k w_{ij}^{[h]} = \varepsilon \delta_{jk}^{[h]} y_{jk}^{[h-1]} \] (14)

Although the changes in weights are calculated for each pattern, we update the weights only when all the training patterns have been presented, i.e., at the end of each epoch rather than immediately after the presentation of each pattern. Thus we have
\[ w_{ij}^{[h]}(t+1) = w_{ij}^{[h]}(t) + \sum_{k=1}^{K} \delta_k w_{ij}^{[h]} \] (15)

where \( K \) represents the total number of patterns to be trained.

This concludes the basic backpropagation algorithm. The main drawback of the BP algorithm is that the convergence speed of the learning procedure is very slow, especially when the training set contains many complicated patterns. Several modifications have been proposed to speed up the backpropagation.

2.2. Backpropagation with adaptive biases

The idea of gradient descent can easily be extended to the bias of each neuron so that the biases can be adjusted in a similar manner to updating the weights. We define the change in the bias at the \( j \)th neuron of the \( h \)th layer during the training of a \( k \)th pattern as
\[ \delta_k \theta_j^{[h]} = - \varepsilon_r \frac{\partial e_k}{\partial \theta_j^{[h]}} \] for \( 1 \leq h \leq L \) (16)

Applying the chain rule, we have
Parameter effects on backpropagation algorithm

\[
\frac{\partial e_k}{\partial \theta_{ij}^{(h)}} = \frac{\partial e_k}{\partial z_{jk}^{(h)}} \frac{\partial z_{jk}^{(h)}}{\partial \theta_{ij}^{(h)}}
\]

\[
= \frac{\partial e_k}{\partial z_{jk}^{(h)}} \left[ \sum_{i=1}^{N_h} w_{ij}^{(h)} y_{ik}^{(h-1)} \right] + \theta_{ij}^{(h)}
\]

\[
= \frac{\partial e_k}{\partial z_{jk}^{(h)}}
\]

\[
= -\delta_{jk}^{(h)}
\]  

Then,

\[
\delta_k \theta_{ij}^{(h)} = \varepsilon_o \delta_{jk}^{(h)}
\]

Thus

\[
\theta_{ij}^{(h)}(t+1) = \theta_{ij}^{(h)}(t) + \varepsilon_o \sum_{k=1}^{K} \delta_{jk}^{(h)}
\]

where \( \varepsilon_o \) is the step size for bias adjustment.

2.3. Backpropagation with adaptive slope of non-linear function

The most widely used activation function is the so-called sigmoid function which can be expressed in the following form for bipolar pixel elements,

\[
F(z) = \frac{1 - e^{-az}}{1 + e^{-az}}
\]

Obviously, by adjusting \( \alpha \), we can adjust the slope of the sigmoid function. We define the change in \( \alpha \) at the \( j \)-th neuron of the \( h \)-th layer during the training of a \( k \)-th pattern as

\[
\delta_k \alpha_j^{(h)} = -\frac{\partial e_k}{\partial \alpha_j^{(h)}} \quad \text{for } 1 \leq h \leq L
\]

Hence,

\[
\alpha_j^{(h)}(t+1) = \alpha_j^{(h)}(t) + \varepsilon \sum_{k=1}^{K} \delta_k \alpha_j^{(h)}(t) + \mu \alpha_j^{(h)}(t) - \alpha_j^{(h)}(t-1)
\]

where \( \mu \) is the momentum for \( \alpha \).

For \( h = L \),

\[
\delta_k \alpha_j^{(L)} = -\frac{\partial e_k}{\partial y_j^{(L)}} \frac{\partial y_j^{(L)}}{\partial \alpha_j^{(L)}}
\]

\[
= (t_{jk} - y_{jk}^{(L)}) F'_a(z_{jk}^{(L)}, \alpha_j^{(L)})
\]

where \( F'_a(z, \alpha) \) is the partial derivative of the non-linear function with respect to \( \alpha \).

For \( h < L \),

Copyright © 2001. All Rights Reserved.
\[
\delta_k z^{[h]}_j = - \frac{\partial e_k}{\partial a^{[h]}_j} = - \sum_{i=1}^{N_h+1} \frac{\partial e_k}{\partial z^{[h+1]}_k} \frac{\partial z^{[h+1]}_k}{\partial a^{[h]}_j} \\
= - \sum_{i=1}^{N_h+1} \frac{\partial e_k}{\partial y^{[h]}_j} \frac{\partial y^{[h]}_j}{\partial z^{[h+1]}_k} \frac{\partial z^{[h+1]}_k}{\partial a^{[h]}_j} \\
= - F'(z^{[h]}_j, a^{[h]}_j) \sum_{i=1}^{N_h+1} \frac{\partial e_k}{\partial y^{[h]}_j} \frac{\partial}{\partial a^{[h]}_j} \left[ \sum_{i=1}^{N_h} w^{[h+1]}_i y^{[h]}_i \right] \\
= F'(z^{[h]}_j, a^{[h]}_j) \sum_{i=1}^{N_h+1} \delta^{[h+1]}_k w^{[h+1]}_{ji} \\
\]

(24)

where \(\delta^{[h+1]}_k\) has the same meaning as defined in (8).

To prevent the slopes taking a very small or a negative value, a small positive number \(\alpha_{\min}\) is chosen such that

\[
a^{[h]}_j = \max \{a^{[h]}_j, \alpha_{\min}\} \\
\]

(25)

3. Comparison of performance

The convergence speed of the learning process can be improved by the previous two modifications on the standard backpropagation algorithm and by those proposed elsewhere (Kruschke and Movellan 1991, Minai and Wiliams 1990, Schreibman and Norris 1990). However, the degree of improvement can be greatly affected by other factors, such as, the initial weights of the network, the complexity of the network (i.e., the number of layers and the number of neurons at each layer) and the characteristics of the training patterns, (i.e., the kind of problem being dealt with). But the effects of these modifications on the generalization capability of the network have not yet been investigated. Since the generalization capability is another important feature of artificial neural networks, modifications should not jeopardize the generalization capability to any extent for convergence speed gain in training.

Based on our simulations, we found that in most cases the backpropagation algorithm with adaptive biases shows an advantage in the improvement of the generalization capability while contributing little to the improvement in learning speed. On the other hand, the backpropagation algorithm with adaptive slopes for the activation functions can speed up the learning process considerably and also result in a good generalization. Computer simulations were conducted in order to compare the performance of standard backpropagation and those of any proposed modifications.

The architecture of the network involved in the simulation is shown in Fig. 1. The network consists of an input layer, 0, an output layer, \(L\), and \(L-1\) hidden layers, i.e. layer 1 to layer \(L-1\). The input layer consists of 16 linear neurons, and each of the other \(L\) layers has 16 non-linear neurons with a bipolar sigmoid activation function of the form defined in (20). The structure of each neuron is shown in Fig. 2.
Four training sets, each consisted of six pairs of patterns of 16 bipolar elements, were constructed for our simulations. The patterns of the first two training sets are orthogonal and those of the other two training sets are non-orthogonal. In the simulations, two types of mapping schemes: auto-associative memory and pattern associative memory, have been used. For each combination of training set and mapping scheme, we first train the network by using the training set, and after the learning process converges to a predetermined error level, we test the generalization capability of the network by feeding it with corrupted training patterns. Although the bipolar training patterns are given in terms of +1 and −1, we use +0·9 and −0·9 in the simulations to avoid unwanted saturations. Since there are 16 elements in each pattern, we inverse two elements randomly during the test process, i.e., 12·5% of the inputs are corrupted. There are 120 noisy versions for each pattern and therefore a total of 720 for each training set.

The following parameters are used in the simulations presented below.

(1) Standard backpropagation
   Learning rate for weights change: 0·1

(2) Backpropagation with adaptive biases
   Learning rate for weights change: 0·1
   Step size for biases change: 0·1
(3) Backpropagation with adaptive slopes of activation functions
   Learning rate for weights change: 0.7
   Step size for slopes change: 0.15
   Momentum for slopes change: 0.05

   All the initial weights and biases are randomly selected from \([-0.1, 0.1]\) with a
   uniformly distributed function.

3.1. Orthogonal patterns

3.1.1. Auto-associative memory. For the auto-associator (AA), the six orthogonal
   training pattern pairs of each of the two sets are:

   **Set 1:**
   \[
   \begin{align*}
   \mathbf{X}_1 &= \mathbf{Y}_1 = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{X}_2 &= \mathbf{Y}_2 = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{X}_3 &= \mathbf{Y}_3 = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{X}_4 &= \mathbf{Y}_4 = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{X}_5 &= \mathbf{Y}_5 = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{X}_6 &= \mathbf{Y}_6 = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \end{align*}
   \]

   **Set 2:**
   \[
   \begin{align*}
   \mathbf{X}_1 &= \mathbf{Y}_1 = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{X}_2 &= \mathbf{Y}_2 = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{X}_3 &= \mathbf{Y}_3 = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{X}_4 &= \mathbf{Y}_4 = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{X}_5 &= \mathbf{Y}_5 = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{X}_6 &= \mathbf{Y}_6 = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \end{align*}
   \]

   Where \(\mathbf{X}_i\) and \(\mathbf{Y}_i\) (for \(i = 1\) to \(6\)) represent the \(i\)th input and the \(i\)th output training
   pattern, respectively.

3.1.2. Pattern associative memory. For the pattern associator (PA), the six orthogonal
   training pattern pairs of each of the two sets have been selected as

   **Set 1:**
   \[
   \begin{align*}
   \mathbf{X}_1 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{Y}_1 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{X}_2 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{Y}_2 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{X}_3 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{Y}_3 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{X}_4 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{Y}_4 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{X}_5 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{Y}_5 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{X}_6 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{Y}_6 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \end{align*}
   \]

   **Set 2:**
   \[
   \begin{align*}
   \mathbf{X}_1 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{Y}_1 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{X}_2 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{Y}_2 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{X}_3 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{Y}_3 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{X}_4 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{Y}_4 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{X}_5 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{Y}_5 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{X}_6 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \\
   \mathbf{Y}_6 &= [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]
   \end{align*}
   \]
\[ X_3=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ Y_3=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_4=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ Y_4=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_5=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ Y_5=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_6=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ Y_6=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]

3.2. Non-orthogonal patterns

3.2.1. Auto-associative memory. For the auto-associator, the training pattern pairs of each of the two sets are listed below:

**Set 1:**
\[ X_1=Y_1=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_2=Y_2=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_3=Y_3=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_4=Y_4=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_5=Y_5=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_6=Y_6=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]

**Set 2:**
\[ X_1=Y_1=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_2=Y_2=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_3=Y_3=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_4=Y_4=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_5=Y_5=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_6=Y_6=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]

3.2.2. Pattern associative memory. For the pattern associator, the two sets of six non-orthogonal pattern pairs are:

**Set 1:**
\[ X_1=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ Y_1=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_2=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ Y_2=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_3=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ Y_3=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_4=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ Y_4=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_5=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ Y_5=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_6=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ Y_6=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]

**Set 2:**
\[ X_1=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ Y_1=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_2=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ Y_2=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_3=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ Y_3=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_4=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ Y_4=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_5=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ Y_5=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ X_6=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
\[ Y_6=[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1] \]
Table 1. Ratios of average learning times (in epochs) and recall accuracy (in number of correct recalls) for modified BP to standard BP: (orthogonal patterns).

<table>
<thead>
<tr>
<th>Number of layer, L</th>
<th>BP with adaptive biases</th>
<th></th>
<th></th>
<th>BP with adaptive slopes</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Learning</td>
<td>Recall</td>
<td></td>
<td>Learning</td>
<td>Recall</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AA  PA</td>
<td>AA  PA</td>
<td></td>
<td>AA  PA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 1</td>
<td>2             1.49  1.10</td>
<td>1.04  1.03</td>
<td>0.21  0.18</td>
<td>1.00  0.98</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3             0.34  1.32</td>
<td>1.01  1.06</td>
<td>0.31  0.53</td>
<td>1.07  1.10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 2</td>
<td>2             0.97  0.63</td>
<td>1.04  1.02</td>
<td>0.43  0.21</td>
<td>0.99  0.98</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3             0.51  0.24</td>
<td>0.95  0.96</td>
<td>1.09  2.54</td>
<td>1.07  1.01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Ratios of average learning times (in epochs) and recall accuracy (in number of correct recalls) for modified BP to standard BP: (non-orthogonal patterns).

<table>
<thead>
<tr>
<th>Number of layer, L</th>
<th>BP with adaptive biases</th>
<th></th>
<th></th>
<th>BP with adaptive slopes</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Learning</td>
<td>Recall</td>
<td></td>
<td>Learning</td>
<td>Recall</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AA  PA</td>
<td>AA  PA</td>
<td></td>
<td>AA  PA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 1</td>
<td>2             0.75  4.40</td>
<td>1.15  1.08</td>
<td>0.19  1.45</td>
<td>0.98  1.01</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3             1.18  1.87</td>
<td>0.93  0.97</td>
<td>0.66  0.18</td>
<td>1.04  1.09</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 2</td>
<td>2             0.20  2.40</td>
<td>1.06  1.06</td>
<td>0.26  1.07</td>
<td>1.00  0.96</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3             0.66  1.59</td>
<td>1.07  1.03</td>
<td>0.52  0.09</td>
<td>1.07  1.10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\mathbf{X}_4=[-1 \ -1 \ 1 \ -1 \ -1 \ -1 \ -1 \ 1 \ 1 \ 1 \ -1 \ -1] \\
\mathbf{Y}_4=[-1 \ -1 \ 1 \ -1 \ -1 \ 1 \ -1 \ 1 \ -1 \ -1 \ 1 \ -1] \\
\mathbf{X}_5=[-1 \ -1 \ 1 \ -1 \ -1 \ 1 \ 1 \ -1 \ -1 \ 1 \ -1 \ -1] \\
\mathbf{Y}_5=[-1 \ -1 \ 1 \ -1 \ -1 \ 1 \ 1 \ -1 \ -1 \ 1 \ -1 \ -1] \\
\mathbf{X}_6=[1 \ -1 \ -1 \ 1 \ -1 \ -1 \ 1 \ -1 \ -1 \ 1 \ -1 \ -1] \\
\mathbf{Y}_6=[1 \ -1 \ -1 \ 1 \ -1 \ -1 \ 1 \ -1 \ -1 \ 1 \ -1 \ -1]
\]

In each of above mentioned four conditions, simulations of five different initializations of weights and biases were used to obtain final results. Tables 1 and 2 show the ratios of average learning times and recall accuracy for modified backpropagation to standard backpropagation. Although backpropagation with adaptive biases has almost no benefits for convergence speed, it can yield a significant improvement in generalization; while the backpropagation with adaptive slopes can speed up learning significantly and also improve generalization. It is interesting to see that the backpropagation with adaptive biases and the backpropagation with adaptive slopes can compensate each other in convergence speed as well as in generalization capability.
4. Backpropagation with both adaptive biases and adaptive slopes (BPABAS)

When seeking a good algorithm, our objective is to obtain as high a generalization capability as possible at a tolerable convergence speed for the learning process. Based on the analyses and simulation results obtained in §3, it appears that by combining the techniques of both adaptive biases and adaptive slopes of activation functions with the standard backpropagation, we may obtain improvements in both convergence speed and generalization capability with little increase in complexity of the algorithm. The performance of this algorithm turned out to be satisfactory. The implementation of this proposed algorithm, called BPABAS can be stated as:

**Step 1.** Initialize the network with random weights and random biases. Set initial values of all $\alpha$ equal to 1.

**Step 2.** Calculate outputs of the network using (1) and (2).

**Step 3.** Calculate output error using (5). If $\sum_{k=1}^{K} e_k$ is less than the predetermined level, stop; otherwise, proceed to Step 4.

**Step 4.** Update biases by (19).

**Step 5.** Update all $\alpha$ by (22).

**Step 6.** Update weights by (15).

**Step 7.** Go to Step 2.

Simulations have been conducted using BPABAS under the same conditions as given in §3. They are as follows.

- Learning rate for weights change: 0.1
- Step size for biases change: 0.1
- Step size for slopes change: 0.15
- Momentum for slopes change: 0.05
- Initial weights and biases: $[-0.1, +0.1]$ uniformly distributed

Table 3 shows the results, it can be seen that, in nearly all cases, the proposed BPABAS algorithm shows advantages in both convergence speed and recall accuracy over the standard BP algorithm.
5. Conclusions

We have analysed and compared the effects of various parameters of the multi-layer feedforward neural networks on their convergence speed and generalization capability. Based on these analyses, an enhanced version of the backpropagation algorithm, namely, BPABAS, has been proposed. The proposed algorithm has been shown to be beneficial in speeding up the learning process while increasing the generalization capability. Simulation results confirmed these observations.

Acknowledgments

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada and in part by Micronet.

References


