Fast Back-Propagation Learning Using Steep Activation Functions and Automatic Weight Reinitialization

Tai-Hoon Cho*, Richard W. Conners*, and Philip A, Araman**

* The Spatial Data Analysis Laboratory
Virginia Polytechnic Institute & State University
Blacksburg, VA 24061

** Brooks Forest Products Center
Virginia Polytechnic Institute & State University
Blacksburg, VA 24061

Abstract --- **In this paper, several back-propagation (BP) learning speed-up algorithms that employ the "gain" parameter, i.e., steepness of the activation function, are examined. Simulations will show that increasing the gain seemingly increases the speed of convergence and that these algorithms can converge faster than the standard BP learning algorithm on some problems. However, these algorithms may also suffer from increased instability, Le., they frequently fail to converge within a finite time. One potential cause for the instability is an inappropriate choice for the initial weights. To overcome the instability resulting from this cause it is proposed that automatic weight reinitialization be used whenever the convergence speed becomes "very slow" due to a *local minimum* or *premature saturation.* On the simulations performed BP algorithms with larger Initial gain (around 2 or 3) and automatic weight reinitialization converged much faster and were more stable than algorithms employing the same gain but not, employing automatic weight reinitialization. The simulations performed involved a diverse set of problems including exclusive-or (XOR), encoder, and parity problems.**

I. INTRODUCTION

Mutilayer feedforward neural networks [1] are popular and are being used on a number of different applications. The standard back-propagation (BP) learning algorithm [1] is frequently used for training these networks. One major difficulty in using the standard BP algorithm is that training typically requires a long learning time. Consequently, a number of modified algorithms [2,3] have been proposed to speed up the learning time. However, these modified methods are usually very complex and often must be "tuned" to fit a particular application.

Among BP learning speed-up algorithms, those using the "gain" parameter are among the easiest to implement. The gain parameter controls the steepness of the activation function. It has been recently shown that a BP algorithm using a steeper activation function converges faster than the standard BP algorithm [4]. (However, it was not noticed that steeper activation function can cause more convergence failures as will be shown later.) In this paper, several learning speed-up algorithms using the "gain" parameter, i.e., steepness of the activation function, are investigated to determine what effect increased gain has on learning time. It will be shown by simulation that although these algorithms can converge faster than the standard BP learning algorithm on some problems, they can be unstable in convergence, i.e., they frequently fail to converge within a finite time. One main reason for this divergence is an inappropriate setting of the initial weights in the network. To overcome this instability, an automatic random reinitialization of the weights is proposed when convergence speed becomes "very slow" due to a *local minimum* or *premature saturation* [5]. BP learning algorithms with this weight reinitialization and larger initial gain (around 2 or 3) were found to be much faster and more stable in convergence than those without weight reinitialization. The simulations performed included such diverse benchmark problems as exclusive-or (XOR), encoder, and parity problems.

II. THE STANDARD BP LEARNING ALGORITHM

A *multilayer feedforward artificial neural network* [1] has one *output layer* and one *input layer* with one or more *hidden layers.* Each layer has a set of *units, nodes,* or *neurons.* It is usually assumed that each layer is fully connected with an adjacent layer without direct connections between layers which are not consecutive. Each connection has a *weight.*

The input of each unit in a layer (except input layer) is given by

$$net_{pi} = \sum_j w_{ij} a_{pj}$$

where net_{pi} is the net input to unit i produced by the presentation of pattern p, W_{ij} is the weight from unit j to unit i, and a_{pj} is the output value of unit j for pattern p. The output of unit i for pattern p is specified by

$$a_{pi} = f(net_{pi})$$

where f is a *semilinear activation function* which is differentiable and nondecreasing.

lSSN# 0-7803-0233-8/91 $1.00©1991 IEEE

The learning of multilayer neural networks is frequently performed using the well-known *back-propagation (BP)* learning algorithm [1]. Details of the derivation of the rule can be found in [1]. The BP algorithm tries to find a set of weights that minimizes an overall measure *E*, $E = \sum_P E_p$, where p indexes over all the patterns in the training set. E_p is defined by

$$E_p = \frac{1}{2}\sum_i (t_{pi} - o_{pi})^2,$$

where t_{pi} is the *target value (desired output)* of output unit *i* for pattern *p*, o_{pi} is the actual output of output unit *i* produced by presenting input pattern *p*, and *i* indexes over the output units.

The weight update rule typically used in the BP algorithm [1] is given by

$$\Delta_p w_{ij}(n+1) = \eta(\delta_{pi} a_{pj}) + \alpha \Delta_p w_{ij}(n)$$

where $\Delta_p w_{ij}$ is the change to be made to w_{ij} following presentation of pattern *p*, *n* represents the *n* th presentation of the training set, η is called *the learning rate*, and α is called the *momentum term* that determines the effect of past weight changes on the current weight changes. When unit i is an output unit, then δ_{pi} in the above equation is given by

$$\delta_{pi} = (t_{pi} - o_{pi}) f'(net_{pi})$$

and when unit *i* is a hidden unit, then δ_{pi} is given by

$$\delta_{pi} = f'(net_{pi}) \sum_k \delta_{pk} w_{ki}.$$

Typically, the semilinear function *f* is given by the *logistic function f(x),* where

$$f(x) = \frac{1}{(1+e^{-x})}.$$

In this case

$$f'(net_{pi}) = a_{pi}(1 - a_{pi}).$$

Hence for an output unit *i*,

$$\delta_{pi} = (t_{pi} - o_{pi}) o_{pi}(1 - o_{pi}),$$

and for a hidden unit *i*,

$$\delta_{pi} = a_{pi}(1 - a_{pi}) \sum_k \delta_{pk} w_{ki}.$$

III. USING ACTIVATION FUNCTIONS WITH "GAIN"

In this section, speed-up learning algorithms are described that use an activation function with the "gain" parameter. In general, a sigmoid activation function $f_i(x)$ with gain g_i for node *i* is defined by

$$f_i(x) = \frac{1}{(1+e^{-g_i x})}.$$

Note that this activation function becomes the usual logistic function if $g_i = 1$. If this activation is used, only updating formulas for δ_{pi} are changed while others are the same as the above. Since

$$f_i'(net_{pi}) = a_{pi}(1 - a_{pi}) g_i,$$

δ_{pi} for an output unit *i* is given by

$$\delta_{pi} = (t_{pi} - o_{pi}) o_{pi}(1 - o_{pi}) g_i$$

and δ_{pi} for a hidden unit *i* is given by

$$\delta_{pi} = a_{pi}(1 - a_{pi}) g_i \sum_k \delta_{pk} w_{ki}.$$

The next problem is how to set or control the gain parameters to minimize the learning times. Here, three different schemes for controlling the gain parameters are considered. Each scheme is described below in detail.

A. SBPk -- Standard Back-Propagation with gain k

In this approach, the gains for all nodes in the network have the same constant value *k*, i.e., $g_i = k$ for each node i in the network, during the entire learning process. This is the simplest of the BP speed-up algorithms that use the gain parameter.

B. IBP -- Improved Back-propagation [6]

This approach uses one constant gain, g_0, for all nodes in the network as in SBPk until a "learning standstill" is detected. This "learning standstill" state is defined as one where connection weights cannot be corrected, even when the output layer has large error values. It was reported [6] that a learning standstill occurs when all output values on one layer are nearly 1 or 0. In the case of learning standstill, the gain g_i at each node i is reduced from its original value of g_0 so that the output values on any layer are significantly different from 0 and 1. The IBP algorithm is defined by the following.

Step 1. Check if a "serious error" has been detected in the output layer. A "serious error" is judged as

$$| t_{pi} - o_{pi} | > \alpha,$$

where α is a constant whose value is usually set to about 0.5.

Step 2. Evaluate the connection weight change

$$\sum_j | \Delta_p w_{ij} |,$$

using

$$\sum_j | \Delta_p w_{ij} | = \sum_j \eta \delta_{pi} a_{pj}.$$

Check to see if

$$\sum_j | \Delta_p w_{ij} | = \sum_j \eta \delta_{pi} a_{pj} < \zeta$$

for some preselected constant ζ. This is equivalent to checking whether

$$\sum_j | a_{pj} o_{pi} (1 - o_{pi}) | < \xi.$$

Step 3. If the conditions specified in Step 1 and Step 2 are satisfied, a neural network is considered to be in a learning standstill. In order to evade this state, g_i is reduced by half and all output values for every layer are computed from the input layer again. The above steps are repeated until the conditions are not satisfied.

Step 4. When the conditions are no longer satisfied, connection weights are updated and the gain parameter is again set to g_0.

C. BPG -- Back-propagation with adaptive Gain [7]

In this method, the gains are updated in a manner analogous to the way weights are updated using the standard BP algorithm. That is, in the standard BP algorithm weight changes are based on $\partial E/\partial w_{ij}$. In the BPG algorithm gain changes are based on $\partial E/\partial g_i$. The resulting gain update rule is given by

$$\Delta g_i = \eta_g \delta_{pi} net_{pi}/g_i,$$

where η_g is the step size of the gains.

IV. AUTOMATIC WEIGHT REINITIALIZATION

In the BP learning algorithms, the initial weights are usually randomly chosen. These initial weights may significantly affect the convergence speed of the learning algorithm. Some initial weights lead to very slow convergence to a solution or, in the worst case, to divergence. It will be shown later that, in general, the BP algorithm using the modified activation function with $g_i > 1$ converges to a solution much faster than when the standard logistic activation function ($g_i = 1$) is used. However, an algorithm using the standard logistic activation function typically can converge to a solution more frequently than an algorithm using an activation function with $g_i > 1$.

To help avoid such divergence it is proposed that random weight reinitialization be used whenever convergence becomes slow. To determine when convergence to a solution is slow, the sum of the error E is checked after some number, N, of epochs (50 in the current implementation) to determine how quickly E is being reduced. Let E_k denote the E value after k epochs and E_{k+N} denote the E value after $k+N$ epochs. If $0 < E_k - E_{k+N} < T$, where T *is* some preselected threshold, then the convergence is considered slow and the weights are randomly reinitialized.

V. EXPERIMENTAL RESULTS

Two types of experiments were performed. The first type of experiment attempts to demonstrate how increased gain can affect both learning speed and stability, The second type of experiment attempts to show how weight reinitialization can affect both leaning speed and stability.

For these experiments, a standard set of diverse benchmark problems were used: exclusive-or (XOR), encoder, and parity problems [1]. For the XOR problem, a 2-2-1 network was used. As a notational convention, let a *K-L-M* neural network denote a 3-layer neural network with *K* input nodes, *L* hidden nodes, and *M* output nodes. Two encoder problems, 4-4 encoder and 8-8 encoder, were used in this experiment. The function of the *N-N* encoder is to map *N* orthogonal input patterns into *N* orthogonal output patterns. In this experiment, both encoders implemented the identity mapping. The *K-L-M* structure used to create the two encoders is $N\text{-}\log_2 N\text{-}N$ where in one instance $N = 4$ and on the other $N = 8$. In the N-bit parity problem, the output value should be 1 if the input pattern contains an odd number of 1's and 0 otherwise. This is a very difficult problem because the most similar patterns (those which differ by a single bit) require a very different response. An *N-N-1* architecture was used for the N-bit parity problem. Experiments were performed on 3-bit, 4-bit, and 5-bit parity problems.

During learning, weights are updated after one pattern is presented. This process is repeated until for each pattern, the difference between every output value and its target value is less than 0.1. (Actual target values used are 0.1 and 0.9 instead of 0 and 1, respectively.) Since the initial values of weights affect the convergence of a learning algorithm, it is reasonable to judge each algorithm by the statistics obtained from multiple runs. In our experiment, 20 sets of different initial weights were randomly generated. These weights were used to run each of the two algorithms on the XOR, encoder, and parity problems. The initial value of the gain parameter g_i on the BP algorithms was also varied on each problem. The learning time was measured by the average of the number of epochs required to reach a solution. For this average, only runs that reached the solution within 5000 epochs were included. If the solution was not reached in 5000 epochs, divergence was assumed. Also, a *failure rate* of the algorithm was measured by the ratio of the number of *unsuccessful* runs to the number of total runs. An unsuccessful run is one in which after 5000 epochs a solution still has not been found. In all these experiments, the momentum term was set to 0.9 for both weights and gains. The step size of the gains, η_g, was set equal to the learning rate η. One fixed value (0.005) for the threshold T was used throughout all the experiments.

Tables I - V show the effects of the initial value of gain g_i on the performance of the BP learning algorithms with gain parameter. Each entry of these table represents average number of epochs of runs that reached the solution within 5000 epochs. The number of convergence failure out of a total 20 runs is shown inside the parentheses. It is noted that the BP learning algorithm using the activation function with steeper slant (i.e., larger values of g_i of about 2 or 3) generally converge faster to a solution. However, it is also notable that more failures in convergence to solution occurred when the steeper activation function is used. Table VI shows the effect of automatic weights reinitialization (initial gain=2) on the BP learning algorithms for $\eta = 0.1$. SBP1 represents standard BP algorithms with $g_i = 1$ SBP2_A, and IBP_A, BPG_A represents SBP2, IBP, and BPG algorithms with initial gain of 2 and automatic weights initializations, respectively. This table shows that automatic reinitialization of weights in the BP algorithms greatly improves the success rate in convergence to solution while still maintaining much faster speed of learning.

VI. CONCLUSION

In this paper, several learning speed-up algorithms using the "gain" parameter, i.e., steepness of the activation function, were investigated to determine the effect of increase gain on learning time. It was shown by simulation that although these algorithms can converge faster than the standard BP learning algorithm on some problems, they can be unstable in convergence, i.e., they frequently fail to converge within a finite time. One main reason for this divergence is inappropriate setting of initial weights in the network. To overcome this instability, an automatic random reinitialization of the weights has been proposed when convergence speed becomes “very slow”. BP learning algorithms with this weight reinitialization and larger initial gain (around 2 or 3) were found to be much faster and more stable in convergence than those without weight reinitialization.

REFERENCES

[1] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, “Learning intend representations by error propagation,” in *Parallel Distributed Processing. Exploration of the Microstructure of Cognition, vol.1: Foundations*, D.E. Rumelhart and J.L. McClelland, Eds. Cambridge, MA: MIT Press, 1986.

[2] S.E. Fahlman, “An empirical study of learning speed in back-propagation,” Tech, Rep. CMU-CS-88-162, Comp. Sci. Dept., Carnegie-Mellon Univ., June 1988.

[3] R.A. Jacobs, “Increased rates of convergence through learning rate adaption,” *Neural Networks*, vol.1, pp.295-307, 1988.

[4] Y. Izui and A. Pentland, “Speeding up back propagation,” *Proc. Int. Joint Conf. Neural Networks* (Washington, D.C.), vol.I, Jan. 1990, pp.639-642.

[5] Y. Lee, S-H. Oh, M.W. Kim, “The effect of initial weights on premature saturation in back-propagation learning,” *Proc. Int. J. Conf. Neural Networks*, Seattle, WA, July 1991, vol.I, pp.765-770.

[6] K. Yamada, H. Kami, J. Tsukumo, and T. Temma, “Handwritten numeral recognition by multi-layered neural network with improved learning algorithm,” *Proc. Int. J. Conf. Neural Networks*, Washington, DC, June 18-22, 1989, vol.II, pp.259-266.

[7] J.K. Kruschke and J.R. Movellan, "Benefits of gains: speeded learning and minimal hidden layers in back-propagation networks,” *IEEE Trans. System, Man, and Cybernetics*, vol.21, pp.273-280, 1991.

TABLE I
Average Epochs for XOR

Initial Gain	SBPk	IBP	BPG
η = 0.05			
1	1659 (3)	1678 (3)	1004 (3)
2	317 (6)	317 (6)	274 (3)
3	310 (6)	303 (5)	327 (6)
4	323 (7)	317 (6)	116 (10)
η = 0.1			
1	859 (2)	871 (2)	499 (4)
2	164 (6)	202 (5)	193 (3)
3	227 (4)	222 (3)	104 (7)
4	94 (7)	94 (6)	119 (9)
η = 0.25			
1	356 (3)	365 (3)	512 (5)
2	71 (5)	73 (4)	90 (7)
3	81 (4)	88 (3)	382 (7)
4	421 (7)	297 (4)	228 (12)

TABLE II
Average Epochs for Encoder 4-4

Initial Gain	SBPk	IBP	BPG
η = 0.05			
1	715 (0)	715 (0)	211 (0)
2	180 (0)	194 (0)	121 (0)
3	110 (0)	108 (0)	95 (0)
4	71 (1)	77 (1)	72 (1)
η = 0.1			
1	360 (0)	360 (0)	116 (0)
2	94 (0)	100 (0)	69 (0)
3	57 (0)	62 (0)	56 (0)
4	108 (1)	80 (0)	74 (1)
η = 0.25			
1	148 (0)	148 (0)	60 (0)
2	43 (0)	46 (0)	43 (0)
3	119 (0)	42 (0)	55 (0)
4	272 (14)	131 (6)	714 (8)

TABLE III
Average Epochs for Encoder 8-8

Initial Gain	SBPk	IBP	BPG
η = 0.05			
1	1526 (0)	1526 (0)	291 (0)
2	400 (0)	392 (0)	210 (0)
3	221 (0)	188 (0)	224 (0)
4	555 (0)	176 (0)	435 (0)
η = 0.1			
1	826 (0)	826 (0)	158 (0)
2	208 (0)	195 (0)	124 (0)
3	318 (0)	97 (0)	186 (1)
4	1151 (6)	110 (0)	1205 (4)
η = 0.25			
1	332 (0)	332 (0)	101 (0)
2	221 (0)	97 (0)	152 (0)
3	1184 (13)	240 (5)	581 (13)
4	- (20)	99 (18)	- (20)

TABLE IV
Average Epochs for Parity 3

Initial Gain	SBPk	IBP	BPG
η = 0.05			
1	1753 (2)	1753 (2)	1217 (4)
2	680 (0)	574 (0)	277 (2)
3	619 (0)	478 (0)	518 (2)
4	633 (2)	641 (1)	107 (7)
η = 0.1			
1	1132 (0)	1169 (1)	951 (4)
2	348 (0)	286 (0)	158 (2)
3	386 (0)	293 (0)	198 (4)
4	392 (9)	163 (6)	488 (6)
η = 0.25			
1	709 (0)	557 (1)	1391 (3)
2	405 (0)	376 (1)	292 (2)
3	292 (9)	63 (9)	330 (7)
4	25 (19)	455 (12)	57 (11)

TABLE V
Average Epochs for Parity 4

Initial Gain	SBPk	IBP	BPG
	η = 0.05		
1	2927 (9)	2941 (9)	2381 (11)
2	1576 (9)	1804 (8)	413 (6)
3	1266 (6)	1246 (6)	389 (9)
4	1852 (10)	1016 (10)	842 (9)
	η = 0.1		
1	1613 (12)	1883 (10)	1870 (14)
2	1669 (6)	1376 (5)	674 (7)
3	841 (13)	1299 (11)	283 (11)
4	386 (18)	191 (18)	444 (15)
	η = 0.25		
1	618 (17)	2127 (3)	998 (17)
2	1410 (l0)	267 (13)	607 (12)
3	- (20)	208 (18)	4039 (19)
4	- (20)	- (20)	- (20)

TABLE VI
The Effect of Steep Activation Function
and Automatic Weights Reinitialization

	SBP1	SBP2_A	IBP_A	BPG_A
	η =0.1			
XOR	859 (2)	253 (0)	250 (0)	186 (0)
Encoder 4-4	360 (0)	94 (0)	100 (0)	69 (0)
Encoder 8-8	826 (0)	208 (0)	195 (0)	124 (0)
3-bit Parity	1132 (0)	272 (0)	267 (0)	191 (0)
4-bit Parity	1613 (12)	931 (1)	915 (0)	547 (0)
5-bit Parity	3888 (18)	1837 (3)	1722 (1)	1800 (3)

Conference Proceedings

1991 IEEE International Conference on Systems, Man, and Cybernetics

"Decision Aiding for Complex Systems"

October 13-16, 1991

Omni Charlottesville Hotel and the University of Virginia Charlottesville, Virginia

IEEE

Volume 3

91CH3067 - 6

SCHOOL OF
ENGINEERING
& APPLIED SCIENCE