Automatic Adaptation of Learning Rate for Backpropagation Neural Networks

V.P. Plagianakos, D.G. Sotiropoulos, and M.N. Vrahatis
University of Patras, Department of Mathematics, GR-265 00, Patras, Greece.
e-mail: vpp|dgs|vrahatis@math.upatras.gr.

Abstract

A method improving the convergence rate of the backpropagation algorithm is proposed. This method adapts the learning rate using the Barzilai and Borwein [IMA J.Numer. Anal., 8, 141–148, 1988] steplength update for gradient descent methods. The determined learning rate is different for each epoch and depends on the weights and gradient values of the previous one. Experimental results show that the proposed method considerably improves the convergence rates of the backpropagation algorithm and, for the chosen test problems, outperforms other well-known training methods.

1 Introduction

Artificial Feedforward Neural Networks (FNNs) have been widely used in many application areas in recent years and have shown their strength in solving hard problems in Artificial Intelligence. Although many different models of neural networks have been proposed, multilayered FNNs are the commonest. In order to train an FNN, supervised training is probably the most frequently used technique in our field. The training process is an incremental adaptation of connection weights that propagate information between simple processing units called neurons. The neurons are arranged in layers and connections between the neurons of one layer to those of the next exist.

Consider an FNN whose the l-th layer contains N_l neurons, $l = 1, \ldots, M$. The network is based on the following equations:

$$net_j^l = \sum_{i=1}^{N_{l-1}} w_{ij}^{l-1,l} y_i^{l-1}, \qquad y_j^l = f(net_j^l),$$

where $w_{ij}^{l-1,l}$ is the weights from the i-th neuron at the $(l-1)$ layer to the j-th neuron at the l-th layer, y_j^l is the output of the j-th neuron that belongs to the l-th layer, and $f(net_j^l)$ is the j-th's neuron activation function. The training process can be realized by minimizing the error function E defined by

$$E = \frac{1}{2} \sum_{p=1}^{P} \sum_{j=1}^{N_M} \left(y_{j,p}^M - t_{j,p} \right)^2 = \sum_{p=1}^{P} E_p, \quad (1)$$

where $\left(y_{j,p}^M - t_{j,p} \right)^2$ is the squared difference between the actual output value at the j-th output layer neuron for the pattern p and the target output value, and p is an index over input-output pairs. The function E also provides the error surface over the weight space.

To simplify the formulation of the above equations, let us use a unified notation for the weights. Thus, for an FNN with n weights, let w be a column weight vector with components $w_1, w_2, \ldots, w_n$, defined on the n–dimensional Euclidean space $\mathbb{R}^n$, and $w^\star$ be an optimal weight vector with components $w_1^\star, w_2^\star, \ldots, w_n^\star$. Also, let E be the batch error measure defined as the sum of squared differences error function over the entire training set, and $\nabla E(w)$ be the gradient vector of the error function E at w. The batch training of an FNN is consistent with the theory of unconstrained optimization since it uses information from all the training set, i.e.

N.E.Mastorakis (ed.), Recent Advances in Circuits and Systems, pp. 337-341.
World Scientific Publishing Co. Pte. Ltd.,ISBN 9810236441, 1998.

the true gradient, and can be viewed as the minimization of the error function E. This minimization task corresponds to the weight update by epoch and, requires a sequence of weight vectors $\{w^k\}_{k=0}^{\infty}$, where k indicates epochs. Successful training implies that $\{w^k\}_{k=0}^{\infty}$ converges to the point $w^\star$ that minimizes E.

The remaining of the paper is organized as follows: In Section 2 we propose a training algorithm with adaptive learning rate based on the Barzilai and Borwein steplength update for gradient descent methods [1]. Experiments and simulation results are presented in Section 3. The final section contains concluding remarks and a short discussion for further work.

2 The proposed algorithm

Our aim is to minimize the error function E

$$\min E(w), \qquad w \in \mathbb{R}^n, \tag{2}$$

where $E \in C^2$. The gradient $\nabla E(w)$ can be obtained by means of back-propagation of errors through the layers. Let the family of gradient training algorithms having the iterative form

$$w^{k+1} = w^k + \eta_k d_k, \qquad k = 0, 1, 2, \ldots \tag{3}$$

where w^k be the current point, d_k be a search direction, and η_k be the steplength. Various choices of the direction d_k give rise to distinct algorithms. A broad class of methods uses $d_k = -\nabla E(w^k)$ as a search direction, and the steplength η_k is given by means of either an exact line search

$$\min_{\eta_k > 0} E\left(w^k - \eta_k \nabla E(w^k)\right) \tag{4}$$

or an inexact line search satisfying Wolfe's conditions [8].

The widely used gradient-based training algorithm, named batch back–propagation (BP), minimizes the error function using the following steepest descent method with constant, heuristically chosen, learning rate η:

$$w^{k+1} = w^k - \eta \nabla E(w^k). \tag{5}$$

Clearly, the behaviour of any algorithm depends on the choice of the steplength no less than the choice of the search direction. It is well known that pure gradient descent methods with fixed learning rate tend to be inefficient (see [5]). This happens, for example, when the search space contains long ravines that are characterized by sharp curvature across them and a gently sloping floor [6].

The proposed algorithm is an adaptive learning rate algorithm based on the Barzilai and Borwein steplength update for the steepest descent method [1]. Barzilai and Borwein proposed a gradient method where the search direction is always the negative gradient direction, but the choice of the steplength (learning rate) is not the classical choice of the steepest descent method. The motivation for this choice is that it provides two–point approximation to the secant equation underlying quasi-Newton methods [5]. This yields the iteration:

$$w^{k+1} = w^k - \eta_k \nabla E(w^k), \tag{6}$$

where η_k for the kth epoch is given by

$$\eta_k = \frac{\langle \delta^{k-1}, \delta^{k-1} \rangle}{\langle \delta^{k-1}, \psi^{k-1} \rangle} \tag{7}$$

where $\delta^{k-1} = w^k - w^{k-1}$, $\psi^{k-1} = \nabla E(w^k) - \nabla E(w^{k-1})$ and $\langle \cdot, \cdot \rangle$ denotes the standard inner product. The key features of this method are the low storage requirements and the inexpensive computations. Moreover, it does *not* guarantee descent in the error function E. Experiments show that this property is valuable in neural network training because very often the method escapes from local minima and flat valleys where other methods are trapped. Furthermore, the method requires no line searches during the training process and less computational effort than methods which use (4) to determine their steplength.

The only implementation problem of the above method, when applied to train an FNN, is that the update formula (7) potentially gives large values for the learning rate η_k. In this case, the method may overshoot the minimum $w^\star$ or possibly diverge. To overcome this problem, we introduce a parameter μ, called maximum growth

factor. The learning rate η_k is not allowed to be greater in magnitude than μ times the previous learning rate η_{k-1}. Formally, the proposed method (BBP) is stated as

$$w^{k+1} = w^k - \lambda_k \nabla E(w^k), \tag{8}$$

where the new learning rate λ_k is given by

$$\lambda_k = \begin{cases} \eta_k \quad , & \left|\dfrac{\eta_k}{\eta_{k-1}}\right| \leq \mu \\ \mu\,\eta_{k-1}, & \text{otherwise} \end{cases}$$

Optimal convergence speed can be obtained by tuning parameter μ. However, tuning μ is quite difficult and seems to be problem dependent. In our simulations, no effort for tuning μ is necessary since the BBP method outperforms other well-known methods when compared with them.

3 Experiments and Results

A computer simulation has been developed to study the properties of the learning algorithms. The simulations have been carried out on a Pentium 133MHz PC IBM compatible using MATLAB version 5.01. The performance of the BBP algorithm has been evaluated and compared with the batch versions of BP, momentum BP (MBP) [3], and adaptive BP (ABP) [7], from Matlab Neural Network Toolbox version 2.0.4. Toolbox default values for the heuristic parameters of the above algorithms are used, unless stated otherwise. For the BBP algorithm, maximum growth factor value is fixed to $\mu = 2$. The algorithms were tested using the same initial weights, initialized by the Nguyen–Widrow method [4], and received the same sequence of input patterns. The weights of the network are updated only after the entire set of patterns to be learned has been presented.

For each of the test problems, a table summarizing the performance of the algorithms for simulations that reached solution is presented. The reported parameters are: *min* the minimum number of epochs, *mean* the mean value of epochs, *max* the maximum number of epochs, *s.d.* the standard deviation of epochs, and *succ.* the simulations succeeded out of 100 trials within the error function evaluations limit. If an algorithm fails to converge within the above limit, it is considered that it fails to train the FNN, but its epochs are not included in the statistical analysis of the algorithms. This fact clearly favours BP, MBP and ABP that require too many epochs to complete the task and/or often fail to find the minimum $w^\star$.

Algorithm	min	mean	max	s.d.	succ.
BP	34	197.76	745	167.93	58
MBP	36	212.18	822	173.40	60
ABP	20	104.92	648	142.11	63
BBP	8	107.02	957	186.34	71

Table 1: Results of simulations for the XOR problem

3.1 The Exclusive-OR Problem

The first test problem we will consider is the exclusive-OR (XOR) Boolean function problem, which has historically been considered a good test of a network model and learning algorithm. The XOR function maps two binary inputs to a single binary output. This simple Boolean function is not linearly separable (i.e. it cannot be solved by a simple mapping directly from the inputs to the output), and thus requires the use of extra hidden units to learn the task. Moreover, it is sensitive to initial weights as well as to learning rate variations and presents a multitude of local minima with certain weight vectors. A 2-2-1 FNN (six weights, three biases) has been used to train the XOR problem. For the BP and MBP algorithms the learning rate is chosen to be 0.1 instead of the default value 0.01 to accelerate their convergence, since – for this problem – they converge slowly with the default value. The termination criterion is $E \leq 0.01$ within 1000 epochs. The results of the simulation are shown in Table 1. In Figure 1 the learning rate variation of the BBP method is plotted versus the epochs for one typical trial.

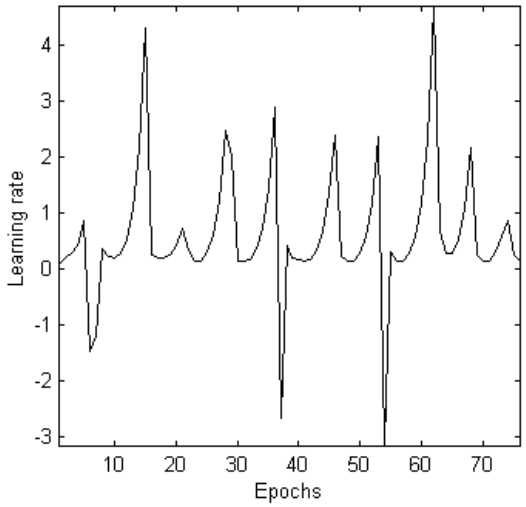

Figure 1: Behaviour of the learning rate of the BBP method over epoch for the XOR problem

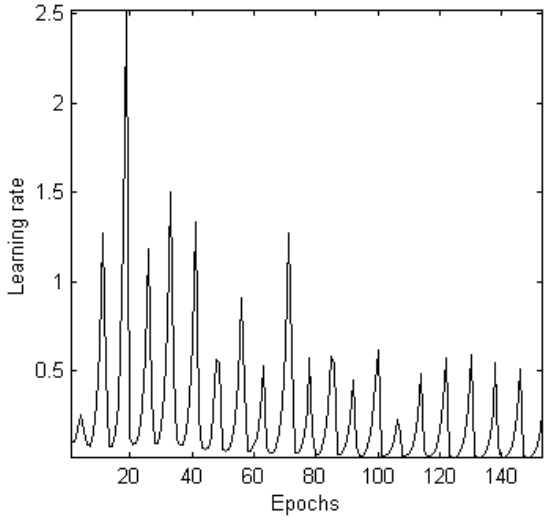

Figure 2: Behaviour of the learning rate of the BBP method over epoch for the 3-Bit parity problem

Algorithm	min	mean	max	s.d.	succ.
BP	–	–	–	–	0
MBP	246	485.44	973	195.42	48
ABP	465	599.19	924	103.91	45
BBP	114	266.17	865	194.08	60

Table 2: Results of simulations for 3 Bit Parity problem

Algorithm	min	mean	max	s.d.	succ.
BP	1132	1533.49	1963	195.84	80
MBP	1263	1496.67	1697	218.91	3
ABP	1342	1756.94	1965	165.67	35
BBP	270	470.12	716	90.44	100

Table 3: Results of simulations for the font problem

3.2 3-Bit Parity

The parity problem can be considered as a generalized XOR problem but it is more difficult. The task is to train a neural network to produce the sum, mod 2, of 3 binary inputs – otherwise known as computing the "odd parity" function. We use a 3-2-1 ANN (eight weights, three biases) to train the 3-Bit Parity problem. The results of the computer simulation are summarized in Table 2.

Despite the effort we made to choose its learning rate, BP diverged in all the simulations. On the other hand, the MBP and ABP algorithms performed much better, with MBP slightly outperforming ABP. The BBP algorithm 60 times successfully trained the FNN mentioned above and exhibited the best performance. Additionally, the average epochs needed (approximately 266) were the best compared with the other algorithms. In Figure 2 the learning rate variation for the BBP method is plotted.

3.3 The Font Learning Problem

In this simulation we consider the font learning problem. We present to the network 26 matrices with the capital letters of the English alphabet. Each letter has been defined in terms of binary values on a grid of size 5×7. For this problem a 35-30-26 FNN (1830 weights, 56 biases) has been used. The performance of the methods has been measured and the Table 3 shows the results. In order to help the methods with fixed learning rate (i.e. BP and MBP), the weights have been initialized following the Nguyen–Widrow method but afterwards the weights and biases of the output layer have been multiplied by 0.01. BBP method has also performed very well without this scaling and exhibited a success rate of 84%, when the other methods failed to converge. Addition-

ally, the error function evaluations limit has been increased to 2000, since (as indicated by the minimum number of epochs needed to train the FNN) all the methods – except the proposed one, which had maximum number of epochs 716 – failed to complete the task within the standard limit we used in the previous test problems.

3.4 Continuous function approximation

The forth test problem we consider is the approximation of the continuous trigonometric function, $f(x) = \sin(x)\cos(2x)$. In this problem we consider the performance of the methods to form a look-up table for the function mentioned above. In this case a 1-15-1 FNN (thirty weights, sixteen biases) is trained to approximate the function $f(x)$, where the input values are scattered in the interval $[0, 2\pi]$ and the network is trained until the sum of the squares of the errors (over 20 input/output sets) becomes less than the error goal 0.1. The network is based on hidden neurons of logistic activations with biases and on a linear output neuron with bias. Comparative results are exhibited in Table 4. Once again, the BBP method exhibited the best performance and had a very high success rate 74%.

Algorithm	min	mean	max	s.d.	succ.
BP	566	724.81	969	128.87	16
MBP	561	718.25	963	131.66	16
ABP	280	577.26	970	200.01	27
BBP	63	309.82	966	230.28	74

Table 4: Results of simulations for function approximation

3.5 Restarting the training

Fahlman [2] introduces the idea of restarting the training of a network which has taken "too long" to learn, but does not specify how long that should be. Heuristic rules have been investigated by many researchers in order to determine where the best point to restart the learning is. One of the advantages of the proposed method is that gives a secure mechanism to decide when to restart. We are able to restart the process of training, if the inner product $\langle \delta^{k-1}, \psi^{k-1} \rangle$ in the denominator in (7) tends to zero.

Problem	min	mean	max	s.d.	succ.
Prob. 3.1	9	98.17	404	84.43	100
Prob. 3.4	22	205.09	834	164.26	99

Table 5: Results of the R-BBP for the XOR and function approximation problems

We have tested this technique and the results for the XOR and the function approximation problems are shown in Table 5. The performance of the method with restarts (R-BBP) is similar to BBP method for the rest test problems. When using this technique, there is no need to use the heuristic parameter μ to control the learning rate increase and, in fact, the method performs better without this constrain.

4 Concluding Remarks

A simple technique for the adaptation of the learning rate for batch training FNNs is proposed. It is shown that this adaptation improves the convergence speed in several classical test problems. To obtain optimal convergence speed, only one heuristic parameter is required to be tuned. One of the main advantages of the proposed algorithm lies in the fact that for many test problems no choice of parameters is needed. For example, in all the test problems considered in this work, this parameter has a fixed value and no extra effort for fine-tuning this parameter was made.

Our experimental results clearly show that the proposed method outperforms the classical training algorithms (BP, MBP and ABP). It runs much faster, has improved the average number of epochs needed, and has better convergence rates.

Further work must be done to eliminate the heuristic parameter μ and optimize the algorithm's performance. Future work also includes testing on bigger and more complex real-life learning tasks.

Acknowledgements

We would like to thank Prof. J. Barzilai for sending us a reprint of his work and E.C. Stavropoulos for his valuable comments and helpful suggestions on earlier drafts of this paper.

References

[1] J. Barzilai and J.M. Borwein, *Two point step size gradient methods*, IMA J. Numer. Anal., **8**, 141–148, (1988).

[2] S. Fahlman, *An empirical study of learning speed in back-propagation networks*, Technical Report CMU-CS-88-162, Carnegie Mellon University, Pittsburgh, PA 15213, September 1988.

[3] R.A. Jacobs, *Increased rates of convergence through learning rate adaptation*, Neural Networks, **1**, 295–307, (1988).

[4] D. Nguyen and B. Widrow, *Improving the learning speed of 2-layer neural network by choosing initial values of the adaptive weights*, IEEE First International Joint Conference on Neural Networks, **3**, 21–26, (1990).

[5] E. Polak, *Optimization: Algorithms and Consistent Approximations*, Springer-Verlag, (1997).

[6] D.E. Rumelhart and J.L. McClelland(eds), *Parallel distributed processing: explorations in the microstructure of cognition*, Vol. 1: Foundations, MIT Press, (1986).

[7] T.P. Vogl, J.K. Mangis, J.K. Rigler, W.T. Zink and D.L. Alkon, *Accelerating the convergence of the back-propagation method*, Biological Cybernetics, **59**, 257–263,(1988).

[8] Ph. Wolfe, *Convergence conditions for ascend methods*, SIAM Review, **11**, 226–235, (1969).