Chapter 2

LEARNING RATE ADAPTATION IN STOCHASTIC GRADIENT DESCENT

V.P. Plagianakos
Department of Mathematics, University of Patras,
GR–26110, Patras, Greece.
vpp@math.upatras.gr

G.D. Magoulas
Department of Information Systems and Computing, Brunel University,
Uxbridge, UB8 3PH, United Kingdom.
George.Magoulas@brunel.ac.uk

M.N. Vrahatis
Department of Mathematics, University of Patras,
GR–26110, Patras, Greece.
vrahatis@math.upatras.gr

Abstract The efficient supervised training of artificial neural networks is commonly viewed as the minimization of an error function that depends on the weights of the network. This perspective gives some advantage to the development of effective training algorithms, because the problem of minimizing a function is well known in the field of numerical analysis. Typically, deterministic minimization methods are employed, however, in several cases, significant training speed and alleviation of the local minima problem can be achieved when stochastic minimization methods are used. In this paper a method for adapting the learning rate in stochastic gradient descent is presented. The main feature of the proposed learning rate adaptation scheme is that it exploits gradient–related information from the current as well as the two previous pattern presentations. This seems to provide some kind of stabilization in the value of the learning rate and helps the stochastic gradient descent to exhibit fast convergence and a high rate of success. Tests in various problems validate the above mentioned characteristics of the new algorithm.

Keywords: Backpropagation neural networks, batch training, on–line training, learning rate adaptation, stochastic gradient descent.

Introduction

In the neural network research field, BackPropagation Neural Networks (BPNNs) are the most popular models. The efficient supervised training of BPNNs is a subject of considerable ongoing research and numerous algorithms have been proposed to this end. A common training approach is to minimize the network learning error, which is a measure of its performance, and is usually based on the difference between the actual output vector of the network and the desired output vector (supervised learning). The rapid computation of a set of weights that minimizes this error is a rather difficult task since, in general, the number of network weights is high and the error function generates a complicated surface in the weight space, possessing multitudes of local minima and having broad flat regions adjoined to narrow steep ones that need to be searched to locate an "optimal" weight set.

Applications of supervised learning can be divided in two categories: stochastic (also called on–line) and batch (also called off–line) learning. *Batch* supervised learning is the classical approach in machine learning: a set of examples is obtained and used in order to learn a good approximating function (i.e. train the network), before the network is used in the application. On the other hand, in *on–line* learning, data gathered during the normal operation of the system are used to continuously adapt the learned function.

Batch training is consistent with the theory of unconstrained optimization, since the information from all the training set is used. It can be viewed as the minimization of the error function E; that is to find a minimizer $w^* = (w_1^*, w_2^*, \ldots, w_n^*) \in \mathbb{R}^n$, such that:

$$w^* = \min_{w \in \mathbb{R}^n} E(w),$$

where E is the batch error measure defined as the sum–of–squared–differences error function over the entire training set:

$$E(w) = \frac{1}{2} \sum_{p=1}^{P} \sum_{j=1}^{N_L} \left(y_{j,p}^L - t_{j,p}\right)^2 = \sum_{p=1}^{P} E_p. \tag{2.1}$$

Note that p is an index over input–output patterns; P denotes the total number of patterns in the training set; $y_{j,p}^L$ is the output of the j-th neuron that belongs to the L-th (output) layer; N_L is the number of neurons of the output layer; $t_{j,p}$ is the desired response at the j-th neuron of the output layer at the input pattern p, and E_p is the pattern–based error of the network. This minimization corresponds to updating the weights by epoch and to be successful it requires a sequence of weight iterates $\{w^k\}_{k=0}^{\infty}$, where k indicates epochs, which converges to a minimizer w^*.

In on–line training, the network weights are updated after the presentation of each training pattern, which may be sampled with or without repetition. This corresponds to the minimization of the instantaneous error of the network E_p (see relation (2.1)).

On–line training may be chosen for a learning task either because of the very large (or even redundant) training set or because we want to model a slowly time–varying system. Although batch training seems faster for small training sets and networks, on–line training is probably faster for large training sets and BPNNs, it helps escaping local minima and provides a more natural approach for learning non–stationary tasks. Given the inherent efficiency of stochastic gradient descent, various schemes have been proposed recently [1, 19, 20, 21, 23]. Unfortunately, on–line training suffers from several drawbacks such as sensitivity to learning parameters [19]. Another disadvantage is that most advanced optimization methods, such as conjugate gradient, variable metric, simulated annealing etc., rely on a fixed error surface, and thus there are difficult to use in on–line training [19].

However, on–line learning has several advantages over batch learning. On–line methods seem more robust as errors, omissions or redundant data in the training set can be corrected or ejected during the training phase. Additionally, training data can often be generated easily and in great quantities when the system is in operation, whereas it is usually scarce and precious before. Finally, on–line training is necessary in order to learn and track time varying functions and to continuously adapt in a changing environment. In a broad sense, on–line learning is essential if we want to obtain *learning* systems as opposed to merely *learned* ones, as pointed out in [24].

The paper is focused on learning rate adaptation schemes for stochastic gradient methods. The next section of the paper briefly reviews the deterministic learning rate adaptation strategies. A new on–line training algorithm with adaptive learning rate is presented in Section 2. Experimental results are reported in Section 3 to evaluate the performance of the proposed algorithm and compare it with several on–line and batch training algorithms. Finally, in Section 4, conclusions are presented.

1. DETERMINISTIC LEARNING RATE ADAPTATION

BPNN training research usually focuses on deterministic gradient–based algorithms with adaptive learning rate that aim to accelerate the learning process. The following strategies are usually suggested: (i) start with a small learning rate and increase it exponentially, if suc-

cessive epochs reduce the error, or rapidly decrease it, if a significant error increase occurs [3, 25], (ii) start with a small learning rate and increase it, if successive epochs keep gradient direction fairly constant, or rapidly decrease it, if the direction of the gradient varies greatly at each epoch [6], (iii) for each weight, an individual learning rate is given, which increases if the successive changes in the weights are in the same direction and decreases otherwise [10, 15, 17, 22], and (iv) use a closed formula to calculate a common learning rate for all the weights at each iteration [9, 12, 16] or a different learning rate for each weight [7, 13]. Note that all the above–mentioned strategies employ heuristic parameters in an attempt to enforce the decrease of the learning error at each iteration and to secure the converge of the training algorithm.

A different approach is based on Goldstein's and Armijo's work on steepest–descent and gradient methods. The method of Goldstein [8] requires the assumption that E is twice continuously differentiable on $\mathcal{S}(w^0)$, where $\mathcal{S}(w^0) = \{w : E(w) \leq E(w^0)\}$ is bounded, for some initial vector w^0. It also requires that η is chosen to satisfy the relation $\sup \|H(w)\| \leq \eta^{-1} < \infty$, where $H(w)$ denotes the Hessian of E at w, in some bounded region, where the relation $E(w) \leq E(w^0)$ holds.

The kth iteration of an algorithm model that follows this approach consists of the following steps:

1. **Choose** η_0 to satisfy $\sup \|H(w)\| \leq \eta_0^{-1} < \infty$ and δ to satisfy $0 < \delta \leq \eta_0$.
2. **Set** $\eta^k = \eta$, where η is such that $\delta \leq \eta \leq 2\eta_0 - \delta$ and **go to** the next step.
3. **Update** the weights $w^{k+1} = w^k - \eta^k \nabla E(w^k)$.

However, the manipulation of the full Hessian is too expensive in computation and storage for BPNNs with several hundred weights [5]. In [11], a technique based on appropriate perturbations of the weights has been proposed for the on–line estimation of the extreme eigenvalues and eigenvectors of the Hessian without calculating the full matrix H. According to experiments reported in [11], the largest eigenvalue of the Hessian is mainly determined by the BPNN's architecture, the initial weights and by short–term low–order statistics of the training data. This technique can be used to determine η requiring additional presentations of the training set in the early training.

Armijo's method, [2], suggests that the value of the learning rate η is related to the value of the Lipschitz constant L, which depends on the morphology of the error surface. In this case, the weights are updated using the formula:

$$w^{k+1} = w^k - \frac{1}{2L} \nabla E(w^k), \tag{2.2}$$

and convergence to the point w^* which minimizes E is obtained (see [2] for conditions under which convergence occurs and a convergence proof). In [12] a local estimation of the Lipschitz constant has been proposed in a learning rate adaptation strategy, which provides increased rate of convergence, and guarantees the stability of the learning process.

2. STOCHASTIC LEARNING RATE ADAPTATION

Despite the abundance of methods for learning from examples, there are only few that can be used effectively for on–line learning. For example, the classic batch training algorithms cannot straightforwardly handle nonstationary data. Even when some of them are used in on–line training there exists the problem of "catastrophic interference", in which training on new examples interferes excessively with previously learned examples, leading to saturation and slow convergence [24].

Methods suited to on–line learning are those that can handle nonstationary (time–varying) data, while at the same time, require relatively little additional memory and computation, in order to process one additional example. Examples of such methods are the variants of the stochastic gradient descent proposed in [1]. The first method, named ALAP_1, uses at each iteration a common learning rate for all the weights:

$$\eta_i^k = \eta_i^{k-1} + \gamma \Big\langle \nabla E_{p-1}(w^{k-1}), \nabla E_p(w^k) \Big\rangle, \tag{2.3}$$

where $\eta_i^0 = c$ for all network weights (c is a small positive constant), and $\langle \cdot, \cdot \rangle$ stands for the usual inner product in $\mathbb{R}^n$.

The other two methods introduced in [1] (named ALAP_2 and ALAP_3), use a different learning rate for each weight. This feature makes these on–line training algorithms able to realize variants of the gradient procedure, which move along a direction that does not necessarily coincide with the gradient direction, to accelerate the minimization process. The ALAP_2 uses the learning rate update formula:

$$\eta_i^k = \eta_i^{k-1} \left[1 + \gamma \, \partial_i E_{p-1}(w^{k-1}) \; \partial_i E_p(w^k) \right]. \tag{2.4}$$

The ALAP_3 is a normalized version of ALAP_2 update rule, given by:

$$\eta_i^k = \eta_i^{k-1} \left[1 + \gamma \, \frac{\partial_i E_{p-1}(w^{k-1}) \; \partial_i E_p(w^k)}{u_i^k} \right], \tag{2.5}$$

where u_i^k is an exponential average of the square of $\partial_i E_p(w^k)$, obtained through:

$$u_i^k = \mu u_i^{k-1} + (1 - \mu) \left[\partial_i E_p(w^k) \right]^2,$$

where μ and γ are positive constants ($\mu = 0.9$ and $\gamma = 0.01$ seem appropriate for most cases [1]).

We propose a new relation for adapting a common learning rate for all weights in the context of stochastic gradient descent:

$$\eta^{k+1} = \eta^k + \gamma_1 \Big\langle \nabla E_{p-1}(w^{k-1}), \nabla E_p(w^k) \Big\rangle + \gamma_2 \Big\langle \nabla E_{p-2}(w^{k-2}), \nabla E_{p-1}(w^{k-1}) \Big\rangle. \tag{2.6}$$

The main feature of the proposed learning rate adaptation scheme is that it exploits gradient–related information from the current as well as the two previous pattern presentations. This seems to provide some kind of stabilization in the values of the learning rate and helps the stochastic gradient descent to exhibit fast convergence and a high rate of success.

A high level description of the proposed algorithm is given in Algorithm 1. In this algorithm model, η is the learning rate, γ_1 and γ_2 are

Stochastic Descent with Adaptive Learning Rate	
0:	Initialize the weights w^0, η^0, γ_1, and γ_2.
1:	**Repeat**
2:	Set $k = k + 1$
3:	Randomly choose a pattern p from the training set.
4:	Using this pattern, calculate $E_p(w^k)$ and then $\nabla E_p(w^k)$.
5:	Calculate the new weights using: $w^{k+1} = w^k - \eta^k \nabla E_p(w^k)$
6:	Calculate the new learning rate using: $\eta^{k+1} = \eta^k + \gamma_1 \langle \nabla E_{p-1}(w^{k-1}), \nabla E_p(w^k) \rangle + \gamma_2 \langle \nabla E_{p-2}(w^{k-2}), \nabla E_{p-1}(w^{k-1}) \rangle$
7:	**Until** the *termination condition* is met.
8:	**Return** the final weights w^{k+1}.

Algorithm 1: The Proposed Algorithm in Pseudocode.

the meta–learning rates. As the termination condition the classification error or an upper limit to the error function evaluations can be used.

3. EXPERIMENTAL TESTS

Simulations have been conducted to study the performance of the proposed training method. To this end, Algorithm–1 has been evaluated and compared with stochastic as well as batch training methods. More specifically, in the simulations reported below, we have compared

Algorithm–1 with three stochastic learning rate adaptation methods proposed by Almeida *et al.* in [1] ($ALAP_1$, $ALAP_2$ and $ALAP_3$). Moreover, for the comparisons we have also tested the on–line Back Propagation (On–line BP) [18], the batch Back Propagation (Batch BP) [18], and the batch adaptive BP with adaptive momentum (Batch ABP) [25].

The algorithms were tested using the same initial weights, initialized by the Nguyen–Widrow method [14], and received the same sequence of input patterns. For each test problem described below, we present a table summarizing the performance of the algorithms for simulations that reached solution. The training phase was considered successful when the network exhibited zero misclassifications on the training set. The reported parameters are: Min the minimum number of pattern presentations, $Mean$ the mean value of pattern presentations, Max the maximum number of pattern presentations, and *Succ.* the number of simulations succeeded out of 100 runs. If an algorithm fails to converge within a predetermined error function evaluation limit, it is considered that it fails to train the BPNN, and its pattern presentations are not included in the statistical analysis of its results.

The values of the meta–learning parameters γ_1 and γ_2 were chosen as $\gamma_1 \ll \gamma_2 = 1$. It seems that the choice of γ_1 and γ_2 is not critical for successful training. However, one may achieve faster convergence, if the meta–learning rates are fine–tuned, which is not the case in our experiments. On the other hand, much effort has been made to properly tune the heuristic learning parameters of the Batch BP and Batch ABP, but there is no guarantee that our final choice is optimal. However, our experience with simulations indicates that the behavior of the algorithms described in the examples to follow is characteristic.

3.1 THE XOR PROBLEM

The first test case is the eXclusive–OR (XOR) Boolean function problem, which has historically been considered as a good test of a neural network model and training 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, the neural network requires the use of extra hidden units to learn the task. Moreover, XOR learning 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 BPNN (six weights, three biases) has been used to learn the XOR mapping. The network is based on neurons with logistic activations. The error function evaluation limit was 4000, i.e. only 4000

pattern presentations were allowed. Comparative results are shown in Table 2.1. From those results it is evident that the proposed algorithm clearly outperforms the $ALAP_1$, $ALAP_2$ and $ALAP_3$ algorithms, the on–line and the batch versions of the BP algorithm, but the ABP method exhibits a slightly higher rate of success. This was expected since, in general, the batch algorithms are very good with problems that have small training sets and/or small network topologies, but are slower than on–line methods.

Algorithm	Min	Mean	Max	Succ.
Batch BP	176	1693.9	3840	17%
Batch ABP	144	1430.4	3708	49%
On–line BP	72	724.2	2972	43%
$ALAP_1$	56	736.1	3900	38%
$ALAP_2$	40	816.9	3960	37%
$ALAP_3$	52	1000.5	3636	43%
Algorithm–1	44	680.2	3388	48%

Table 2.1 Results for the XOR problem

3.2 THE NUMERIC FONT LEARNING PROBLEM

In the second experiment, a network with 64 input, 6 hidden and 10 output nodes (444 weights, 16 biases) is trained to recognize 8×8 pixel machine printed numerals from 0 to 9 in helvetica italic [12]. The network is based on neurons of the logistic activation model. The termination condition for all algorithms tested is to exhibit zero misclassifications on the training set, within 1000 error function evaluations. Detailed results regarding the training performance of the algorithms are presented in Table 2.2.

The on–line BP method exhibited very high success rate, but the $ALAP_1$, $ALAP_2$ and $ALAP_3$ methods were faster. On the other hand, the proposed method and the On–line BP algorithms had almost perfect success rate (99%). Moreover, Algorithm–1 exhibited fast convergence since it needed on average only 436 pattern presentations in order to train the network.

Algorithm	Min	Mean	Max	Succ.
Batch BP	210	500.8	980	90%
Batch ABP	420	789.2	990	51%
On-line BP	230	507.7	950	99%
$ALAP_1$	130	475.5	990	90%
$ALAP_2$	190	433.6	860	90%
$ALAP_3$	210	486.3	990	96%
Algorithm–1	170	436.3	870	99%

Table 2.2 Results for the numeric font learning problem

3.3 THE ALPHABETIC FONT LEARNING PROBLEM

For this problem, 26 matrices with the capital letters of the English alphabet are presented to a 35–30–26 BPNN (1830 weights, 56 biases). Each letter has been defined in terms of binary values on a grid of size 5×7. The BPNN was based on neurons with logistic activations. The results are exhibited in Table 2.3. Once again, the proposed method exhibited a very high success rate (96%) and was faster than all the other methods considered. On average it needed only 749 pattern presentations in order to complete the task.

Algorithm	Min	Mean	Max	Succ.
Batch BP	4498	21375.9	41860	79%
Batch ABP	3588	3815.7	4212	98%
On–line BP	1404	1861.1	2418	87%
$ALAP_1$	494	1519.4	2548	72%
$ALAP_2$	338	756.6	1846	94%
$ALAP_3$	338	754.5	2418	79%
Algorithm–1	364	749.6	1872	96%

Table 2.3 Results for the alphabetic font learning problem

4. CONCLUSIONS

In this paper, a new on–line learning algorithm for neural network training has been proposed. The algorithm is a variant of the stochastic gradient descent that uses a common adaptive learning rate for all weights. Such algorithms are able to train large networks capable to adapt to data unknown at the time of the first training, and are better suited for tasks with large, redundant or slowly time varying training sets.

The simulation results suggest that the proposed algorithm provides fast and stable learning, when compared with other on–line as well as batch training methods, and, therefore, a greater possibility of good performance. Further work must be done to optimize the algorithm's performance and extensive testing on bigger and more complex real–life learning tasks is necessary to fully evaluate its performance.

References

[1] L.B. Almeida, T. Langlois, J.D. Amaral, A. Plankhov (1998). Parameter adaptation in Stochastic Optimization, In: *On–line Learning in Neural Networks*, D. Saad, ed., 111–134, Cambridge University Press.

[2] L. Armijo (1966). Minimization of functions having Lipschitz–continuous first partial derivatives. *Pacific Journal of Mathematics*, **16**, 1–3.

[3] R. Battiti (1989). Accelerated backpropagation learning: two optimization methods. *Complex Systems*, **3**, 331–342.

[4] R. Battiti (1992). First- and second-order methods for learning: between steepest descent and Newton's method. *Neural Computation*, **4**, 141–166.

[5] S. Becker and Y. Le Cun, (1988). Improving the convergence of the back–propagation learning with second order methods. In: D.S. Touretzky, G.E. Hinton and T.J. Sejnowski (Eds.), *Proceedings of the 1988 Connectionist Models Summer School*, 29–37. San Mateo: Morgan Koufmann.

[6] L.W. Chan and F. Fallside (1987). An adaptive training algorithm for back–propagation networks. *Computers Speech and Language*, **2**, 205–218.

[7] S.E. Fahlman (1988). Faster–learning variations on back–propagation: an empirical study. In: D.S. Touretzky, G.E. Hinton and T.J. Sejnowski (Eds.), *Proceedings of the 1988 Connectionist Models Summer School*, 38–51. San Mateo: Morgan Koufmann.

[8] A.A. Goldstein (1962). Cauchy's method of minimization. *Numerische Mathematik*, **4**, 146–150.

[9] H.C. Hsin, C.C. Li, M. Sun, and R.J. Sclabassi (1995). An adaptive training algorithm for back–propagation neural networks. *IEEE Transactions on System, Man and Cybernetics*, **25**, 512–514.

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

[11] Y. Le Cun, P.Y. Simard, and B.A. Pearlmutter (1993). Automatic learning rate maximization by on–line estimation of the Hessian's eigenvectors. In: S.J. Hanson, J.D. Cowan, and C.L. Giles (Eds.), *Advances in Neural Information Processing Systems 5*, 156–163. San Mateo: Morgan Koufmann.

[12] G.D. Magoulas, M.N. Vrahatis and G.S. Androulakis (1997). Effective back–propagation with variable stepsize, *Neural Networks*, **10**, 69–82.

[13] G.D. Magoulas, M.N. Vrahatis, and G.S. Androulakis (1999). Improving the convergence of the back–propagation algorithm using learning rate adaptation methods. *Neural Computation*, **11**, 1769–1796.

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

[15] M. Pfister and R. Rojas (1993). Speeding–up backpropagation – A comparison of orthogonal techniques. In *Proceedings of the Joint Conference on Neural Networks*. 517–523. Nagoya, Japan.

[16] V.P. Plagianakos, M.N. Vrahatis, and G.D. Magoulas (1999). Nonmonotone Methods for Backpropagation Training with Adaptive Learning Rate. In: *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'99)*. Washington D.C.

[17] M. Riedmiller and H. Braun (1993). A direct adaptive method for faster backpropagation learning: the Rprop algorithm. In: *Proceedings of the IEEE International Conference on Neural Networks*, 586–591. San Francisco.

[18] D.E. Rumelhart and J.L. McClelland (1986). Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations, MIT Press.

[19] D. Saad (1998). *On–line Learning in Neural Networks*, Cambridge University Press.

[20] N.N. Schraudolph (1998). Online Local Gain Adaptation for Multi–layer perceptrons, Technical Report, IDSIA–09–98, IDSIA, Lugano, Switzerland.

[21] N.N. Schraudolph (1999). Local Gain Adaptation in Stochastic Gradient Descend, Technical Report, IDSIA–09–99, IDSIA, Lugano, Switzerland.

[22] F. Silva and L. Almeida (1990). Acceleration techniques for the back–propagation algorithm. *Lecture Notes in Computer Science*, **412**, 110–119. Berlin: Springer–Verlag.

[23] R.S. Sutton (1992). Adapting Bias by Gradient Descent: an Incremental Version of Delta–Bar–Delta, In: *Proc. of the Tenth National Conference on Artificial Intelligence*, MIT Press, 171–176.

[24] R.S. Sutton and S.D. Whitehead (1993). Online Learning with Random Representations, In: *Proc. of the Tenth International Conference on Machine Learning*, Morgan Kaufmann, 314–321.

[25] T.P. Vogl, J.K. Mangis, J.K. Rigler, W.T. Zink and D.L. Alkon (1988). Accelerating the Convergence of the Back–propagation Method, Biological Cybernetics, **59**, 257–263.