PERGAMON

Nonlinear Analysis 47 (2001) 3425–3430

Nonlinear Analysis

www.elsevier.nl/locate/na

Adaptive Stepsize Algorithms for On–line Training of Neural Networks

G.D. Magoulas [a,c], V.P. Plagianakos [b,c], M.N. Vrahatis [b,c]

[a] *Department of Information Systems and Computing, Brunel University, Uxbridge UB8 3PH, United Kingdom*

[b] *Department of Mathematics, University of Patras, GR-26110 Patras, Greece*

[c] *University of Patras Artificial Intelligence Research Center (UPAIRC)*

Abstract

In this paper a method for adapting the stepsize in on–line network training is presented. The proposed technique derives from the stochastic gradient descent proposed by Almeida *et al.* [On–line Learning in Neural Networks, 111–134, Cambridge University Press, 1998]. The new aspect of our approach consists in taking into consideration previously computed pieces of information regarding the adaptation of the stepsize. The proposed algorithm has been implemented, tested and compared against other on–line methods in three problems. The results shown that it behaves predictably and reliably, and possesses a satisfactory average performance.

1 Introduction

Learning in multilayer perceptrons (MLPs) is usually achieved by minimizing the network error using a *stochastic*, also called *on–line*, or a *batch*, also called *off–line* training algorithm. Batch training is considered as the classical machine learning approach: a set of examples is used for learning a good approximating function, i.e. train the MLP, before the network is used in the application. In this case, the aim is to find a minimizer $w^* = (w_1^*, w_2^*, \ldots, w_n^*) \in \mathbb{R}^n$, such that:

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

0362-546X/01/$ - see front matter
PII: S0362-546X(01)00458-8

where E is the batch error measure of an MLP, whose l-th layer ($l = 1, \ldots, M$) contains N_l neurons

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

In Relation (1), the error function is based on the squared difference between the actual output value at the j-th output layer neuron for pattern p, $y_{j,p}^{M}$, and the target output value, $t_{j,p}$; p is an index over input–output pairs. The rapid computation of such a minimizer is a rather difficult task since, in general, the dimension of parameter space is high and the function E generates a complicated surface in this 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.

On the other hand, in on–line training, the MLP parameters are updated after the presentation of each training example, which may be sampled with or without repetition. On–line training may be the appropriate choice for learning a task either because of the very large (or even redundant) training set, or because of the slowly time–varying nature of the task. Although batch training seems faster for small–size training sets and networks, on–line training is probably more efficient for large training sets and MLPs. It helps escaping local minima and provides a more natural approach for learning non–stationary tasks. On–line methods seem to be more robust than batch methods 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 they are usually scarce and precious before. Lastly, on–line training is necessary in order to learn and track time varying functions and continuously adapt in a changing environment.

Given the inherent efficiency of stochastic gradient descent, various schemes have been proposed recently [1,4–7]. However, these schemes suffer from several drawbacks such as sensitivity to learning parameters [4]. Note that in this framework it is not possible to use advanced optimization methods, such as conjugate gradient, variable metric, simulated annealing etc., as these methods rely on a fixed error surface [4].

This paper is organized as follows: a new on–line training algorithm is presented in the next section. 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. In Section 4, conclusions are presented.

2 Memory–based adaptation of the stepsize

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

ON–LINE TRAINING WITH ADAPTIVE STEPSIZE

0: Initialize the weights w^0, η^0, and K.

1: **Repeat** for each input pattern p

2: Calculate $E(w^p)$ and then $\nabla E(w^p)$.

3: Update the weights:

$$w^{p+1} = w^p - \eta^p \nabla E(w^p)$$

4: Calculate the stepsize to be used with the next pattern $p+1$:

$$\eta^{p+1} = \eta^p + K\langle \nabla E(w^{p-1}), \nabla E(w^p) \rangle$$

5: **Until** the *termination condition* is met.

6: **Return** the final weights w^{p+1}.

Algorithm 1: The proposed algorithm in pseudocode.

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 [8]. 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.

A high–level description of the stochastic gradient descent equipped with the proposed adaptation strategy is given in Algorithm 1, where η is the stepsize, K is the meta–stepsize and $\langle \cdot, \cdot \rangle$ stands for the usual inner product in $\mathbb{R}^n$.

The memory–based calculation of the stepsize, in Step 4, takes into consideration previously computed pieces of information to adapt the stepsize for the next pattern presentation. This seems to provide some kind of stabilization in the calculated values of the stepsize, and helps the stochastic gradient descent to exhibit fast convergence and high success rate. Note that the classification error or an upper limit to the error function evaluations can be used as the termination condition in Step 5.

The key features of this method are the low storage requirements and the inexpensive computations. Moreover, in order to calculate the stepsize to be used at the next iteration, this on–line algorithm uses information from the current, as well as the previous iteration.

3 Simulation results

A set of computer simulations has been developed to study the performance of the proposed stepsize adaptation scheme. To this end, the stochastic gradient descent with adaptive stepsize (Algorithm–1) has been compared with other on–line and batch training methods. More specifically, we have compared Algorithm–1 with three stochastic adaptive stepsize methods proposed by Almeida *et al.* in [1]. The first of these methods, i.e. ALAP_1, uses at each iteration a common adaptive stepsize for all weights, while the other two, i.e ALAP_2 and ALAP_3, use a different adaptive stepsize along each weight direction. This feature makes ALAP_2 and ALAP_3 able to actually implement variants of the stochastic gradient procedure, and, thus, follow a search direction that does not necessarily coincide with the gradient direction, but, hopefully, provides accelerated learning. The ALAP_2 method is called the *unnormalized* update rule, while ALAP_3 is the *normalized* update rule. Additionally, for comparison purposes we have also tested the on–line Back–Propagation (On–line BP), the batch Back–Propagation (Batch BP), and the batch adaptive BP with adaptive stepsize and momentum (Batch ABP) [9]. The algorithms were tested using the same initial weights, initialized by the Nguyen–Widrow method [3], and received the same sequence of input patterns. The meta–learning parameter received the fixed value $K = 1$. It seems that the choice of K is not critical for successful training. However, one may achieve faster convergence, if the value of the meta–learning parameter is fine–tuned. As a last remark, the training phase was considered successful when the network exhibited zero misclassifications on the training set.

For each test problem, described below, a table is presented that summarizes the performance of the algorithms for simulations that reached solution out of 100 runs. 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. If an algorithm fails to converge within a predetermined error function evaluation limit, it is considered that it fails to train the MLP, and its pattern presentations are not included in the statistical analysis of that algorithm.

The first test problem we will consider is the eXclusive–OR (XOR) Boolean function problem. This simple problem is not linearly separable (i.e. it cannot be solved by a simple mapping directly from the inputs to the output), and thus an MLP requires the use of extra hidden units to learn the task. Moreover, training is sensitive to initial weights as well as to stepsize variations and the batch error surface of the problem presents a multitude of local minima with certain weight vectors. A 2–2–1 MLP (six weights, three biases) with logistic activations has been used to learn the XOR function. The error function evaluation limit was 4000, i.e. only 4000 pattern presentations were allowed.

From the results of Table 1, it is evident that the proposed algorithm clearly outperforms the ALAP_1, ALAP_2 and ALAP_3 algorithms, the on–line and the batch

Table 1
Comparative results for the XOR problem

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%

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.

In the second experiment, a network with 64 input, 6 hidden and 10 output nodes (444 weights, 16 biases) was trained to recognize 8×8 pixel machine printed numerals from 0 to 9 in helvetica italic [2]. The network was based on neurons of the logistic activation model. The termination condition for all algorithms tested was 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. 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 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.

Table 2
Results for the numeric font learning problem

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%

In the third experiment, 26 matrices with the capital letters of the English alphabet are presented to a 35–30–26 MLP (1830 weights, 56 biases) that used logistic activations. Each letter has been defined in terms of binary values on a grid of size 5×7. The results are exhibited in Table 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.

4 Conclusions

In this paper, a new learning algorithm for neural network on–line training has been proposed. Such algorithms are able to train large networks using on–line

Table 3
Comparative results for the alphabetic font learning problem

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%

data, and are better suited for tasks with large, redundant or slowly time–varying training sets. Numerical evidence 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 is needed to optimize the algorithm's performance and test it on bigger and more complex real–life learning tasks.

Acknowledgement

The authors wish to thank the referees for useful suggestions.

References

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

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

[3] D. Nguyen and B. Widrow, Improving the learning speed of 2–layer neural network by choosing initial values of the adaptive weights, In: Proc. IEEE 1^{st} Int. J. Conf. on Neural Networks, 21–26, 1990.

[4] D. Saad, On–line Learning in Neural Networks, Cambridge University Press, 1998.

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

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

[7] R.S. Sutton, Adapting Bias by Gradient Descent: an Incremental Version of Delta–Bar–Delta, In: Proc. 10^{th} Nat. Conf. on Artificial Intell., MIT Press, 171–176, 1992.

[8] R.S. Sutton and S.D. Whitehead, Online Learning with Random Representations, In: Proc. 10^{th} Int. Conf. on Machine Learning, Morgan Kaufmann, 314–321, 1993.

[9] T.P. Vogl, J.K. Mangis, J.K. Rigler, W.T. Zink and D.L. Alkon, Accelerating the Convergence of the Back–propagation Method, Biol. Cybern., 59 (1988) 257.