Acceleration Techniques for the Backpropagation Algorithm

Fernando M. Silva - Luís B. Almeida

INESC - Instituto de Engenharia de Sistemas e Computadores
R. Alves Redol, 9, 1100 Lisboa, Portugal
IST - Instituto Superior Tecnico
R. Rovisco Pais, 1100 Lisboa, Portugal

Abstract

Like other gradient descent techniques, backpropagation converges slowly, even for medium sized network problems. This fact results from the usually large dimension of the weight space and from the particular shape of the error surface in each iteration point. Oscillation between the sides of deep and narrow valleys, for example, is a well known case where gradient descent provides poor convergence rates.

In this work, we present an acceleration technique for the backpropagation algorithm based on individual adaptation of the learning rate parameter of each synapse. The efficiency of the method is discussed and several related issues are analyzed.

1 Introduction

Backpropagation converges slowly, even for medium sized network problems. However, it has a computationally simple structure and, moreover, it can be implemented as a local algorithm. These two properties largely justify the success of this technique as the standard training method for feed-forward, multi-layered neural networks, despite its slow convergence. In this context, the search for faster and more robust training methods must meet two requirements: simplicity, in order to reduce the total computational load (not just the number of training cycles), and locality, in order to maintain compatibility with distributed hardware architectures. In this paper, we study an acceleration technique for backpropagation obeying to these two constraints, and we discuss several related issues.

This work is structured as follows. In section 2 the conventional method of backpropagation by steepest descent is reviewed and some earlier works on acceleration techniques are reported. In section 3, a new technique for acceleration of backpropagation in batch learning is presented. In section 4, the efficiency of the method in narrow valleys is analyzed, and the role of the momentum term is discussed. In section 5 simulation results obtained for several networks are presented. In section 6, an extension of the algorithm to real-time learning procedures is suggested. Finally, in section 7, conclusions of this work are presented.

2 Overview

Let $E(\mathbf{w})$ denote the quadratic error function to be minimized by a given network, and let $\mathbf{w}$ denote the weight vector. $E(\mathbf{w})$ is defined as

$$E(\mathbf{w}) = \sum_p \sum_i (y_i^p - d_i^p)^2 \tag{1}$$

where y_i^p and d_i^p represent the actual and desired outputs for unit i and pattern p. In its basic form, the weight update rule of backpropagation algorithms can be written as

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta \frac{\partial E}{\partial \mathbf{w}} \tag{2}$$

where n identifies the update epoch and η the learning rate parameter. If the weight update is performed after each sweep over the whole training set, the training procedure is said to be batch or deterministic. If the weight update is performed for each input/output pair, using a gradient estimate, the training procedure is said to be real-time. In the following we will consider only batch training procedures, the real-time case being discussed in section 6.

As described by eq. (2), the backpropagation algorithm implements steepest descent in E, in the sense that the weight update is performed in the direction that yields a maximal error reduction. Accordingly, the weight update is largest along the components with largest derivatives. However, as noted by Sutton [Sutton86], there are good reasons to do exactly the opposite. First, if a certain component of the gradient vector has a small value, that may indicate that the error surface has a gentle slope along that direction and may be far away from the minimum, and so bigger steps in that component can be performed to reach the minimum faster. Second, if a certain component of the gradient has a large value, it may correspond to a direction orthogonal to a deep and narrow valley, and smaller steps in this direction would help to avoid oscillations and to reach the bottom of the ravine.

A first technique to avoid the problem of ravines is to add a momentum term to the weight update equation [Rumelhart86]. The introduction of the momentum term implements a low-pass filtering of the gradient vector that, while dampening oscillations that may occur across ravines, yields a gain of up to $1/(1-\alpha)$ (α being the momentum parameter) along the "dc" components of the gradient vector. To provide a significant gain along the directions where the partial derivative remains approximately constant for several iterations, one must set the value of α near one. However, this option implies also a longer memory effect in the weight update equation, and a small deviation in the gradient direction may require many iterations to be corrected, and sometimes may drive the process far away from the bottom of the ravine. In practice, the use of the momentum term implies a constant fine-tunning of the learning parameters, and an algorithm to automatically adjust their values would be desirable.

A possible solution to this problem was recently proposed by Chan and Fallside [Chan87]. These authors proposed an adaptation rule for the learning rate parameter, based on the cosine of the angle defined by the gradient vector in two successive iterations. In that algorithm, the momentum term is also adapted, in order to avoid an

excessive weight of the filter memory over the gradient component in the weight update equation. However, this technique implies a non-local task (the angle computation) and, sometimes, it drives the learning rate parameter to a very small value.

A different approach to this problem was initially proposed by Kesten [Kesten58] on a different context, and by Sutton [Sutton86] for neural networks. The proposed solution is to let each synapse have its own learning rate parameter and, in some way, increase or decrease its value according to the number of sign changes observed in the partial derivative of the error function relative to the corresponding weight. It is clear that, by doing so, the weight update is no longer performed along the gradient direction, and the resulting procedure is not strictly a steepest descent one. However, the dot product of the gradient vector with the weight update vector remains negative, and so the derivative of the error function along the update direction is always negative. A specific implementation of the technique proposed by Sutton was recently studied by Jacobs [Jacobs88]. In the next section, we will introduce another algorithm that implements the same basic concept.

3 The Adaptive Backpropagation Algorithm

Consider that in a given learning process the sign of a certain component of the gradient remains equal for several iterations. This fact suggests that the error surface has a smooth variation along this axis and, therefore, the learning rate for this particular component should be increased. On the other hand, if the sign of some component changes in several consecutive iterations, the learning rate parameter should be decreased to avoid oscillation.

A simple and effective way of implementing this basic idea is as follows. Write the weight update equation for the synapse linking units i and j at epoch n as

$$w_{ij}(n) = w_{ij}(n-1) + \eta_{ij}(n)\nabla_{ij}E(n) \tag{3}$$

where $\eta_{ij}(n)$ is the specific learning rate parameter of the ij-th synapse at epoch n and $\nabla_{ij}E(n)$ is the ij-th component of the gradient vector at time n. At each epoch, adapt the learning rate parameter according to

$$\eta_{ij}(n) = \begin{cases} u\eta_{ij}(n-1) & \text{if} \quad \nabla_{ij}E(n)\nabla_{ij}E(n-1) > 0 \\ d\eta_{ij}(n-1) & \text{if} \quad \nabla_{ij}E(n)\nabla_{ij}E(n-1) < 0 \end{cases} \tag{4}$$

where u and d are positive constants with values slightly above and below unity, respectively.

One of the significant features of this adaptation rule is its exponential nature. As it will be shown in the examples of section 5, the convergence process can increase or decrease each learning rate parameter by several orders of magnitude in a few iterations. This property enables the process to find "optimal" values for the learning rate parameters in a few learning cycles, producing a fast reduction of the total output error. In previous reported works, as in [Jacobs88], a linear increase and exponential decrease of the learning rate parameters was proposed. Although this linear adaptation rule is also able to produce excellent results, it implies a hard work for tuning the initial learning

rate parameters (in Jacobs' work, five initial parameters must be adjusted to obtain optimal results). Mainly due to its exponential nature, the multiplicative technique has a low sensitivity to initial values. Note that in a wide range of tests performed with this technique, we found that a value of u somewhere between 1.1 and 1.3 was able to produce good results. For the d parameter, a value slightly below $1/u$ enables the adaptive process to give a small preference to learning rate decrease, yielding a somewhat more stable convergence process.

The application of this algorithm to several neural network problems has shown that the fast increase of some learning rate parameters could sometimes drive the learning process to an unstable behavior. This phenomenon can be avoided by testing the total output error after each iteration, and rejecting the new weight vector if an error increase is observed. However, the adaptation of the learning rate parameters is still performed as usual, using as a reference the gradient evaluated in the rejected iteration point. With this technique, a valid iteration point can generally be obtained after a few training cycles (typically, one or two epochs). If this simple strategy does not work after a few trials, it is always possible to simply reduce all the learning rate parameters by a fixed factor and repeat the process.

4 Diagonal Valleys

In the conventional momemtum technique, the weight update vector results from the linear filtering of the gradient vector. This means that, for a given shape of the error surface, the performance of the method is invariant to any rotation of the weight space. When independent learning rate parameters are used, the weight update results from a non-isotropic operation, and the performance of the method can become dependent on the relative positions of the error surface and the coordinate axes.

Consider, for instance, that the convergence process reaches a deep and narrow valley. One would like to have a small value of the learning rate parameters for directions orthogonal to the valley, and a large value for the direction parallel to it. The proposed learning rate adaptation rule produces this optimal result if the valley is parallel to any coordinate axis. However, when the angle defined by the principal direction of the valley and the coordinate axis increases, the learning rate parameters must find a compromise solution in order to follow the direction of the valley. Although the proposed method is able to provide this compromise in a few iterations, it is clear that its efficiency will tend to lower as the angle increases.

In order to gain some further insight into this question, we present in this section a short comparative study of the performance of both techniques in deep and narrow valleys. To enable a more accurate control of the error surface shape, this study was carried out using the bi-dimensional quadratic function

$$E(x,y) = (ax_r)^2 + y_r^2 \tag{5}$$

By varying the value of a in equation (5) is possible to control the shape of the error surface (a value of a lower than 1 produces a valley along the x_r direction). A simple rotation transform between (x_r,y_r) and (x,y) was used to generate different angles ϕ of

Rotation ϕ (degrees)	A		B					
	BP	Adp. BP	BP			Adp. BP		
			$\alpha = .5$	$\alpha = .9$		$\alpha = .5$	$\alpha = .9$	
	$\eta = .5$	$\eta = .5$	$\eta = .5$	$\eta = .5$	$\eta = .05$	$\eta = .5$	$\eta = .5$	$\eta = .05$
0	10367	44	5180	1007	10339	38	35	48
10		507				344	467	495
20		1904				1137	1188	1204
30		3035				2017	1182	1198
45		3853				2675	1197	1209

Table 1 - Performance evaluation: Conventional vs. adaptative BP. A - without momentum term. B - with momemtum term. Each entry shows the number of iterations needed to reach solution.

the valley relative to the x-axis.

In the first test, the basic backpropagation algorithm and the adaptive technique were both used to find the minimum of $E(x, y)$. The parameter a was set to 0.02. A value of $\eta = .5$ was used for conventional backpropagation (this value is nearly optimal for this case). In adaptive backpropagation, the η's were initialized to 0.5, and adapted at each iteration according to eq. (4), with $u = 1.2$ and $d = .7$. The results of this test, for several values of ϕ, are shown in table 1, part A. Each entry shows the number of iterations needed to obtain $E(x, y) < 0.001$, starting at $(x_r, y_r) = (100, 2)$ (the results for conventional backpropagation are independent of ϕ, so only one value is shown).

The results show that the adaptive method decreases the number of iterations needed to reach the solution in all cases. However, as the angle increases, a significant loss of efficiency is observed. In fact, the poor performance of the method in this case results from the continuous adaptation of the two learning rate parameters, trying to "track" the valley direction. A possible solution to minimize this problem is to introduce a momentum pre-filtering of the gradient vector in the adaptive method, in order to smooth this process and amplify the gradient component along the valley direction. With this option, the weight adaptation rule becomes

$$\Delta w_{ij}(n) = \eta_{ij}(n) v_{ij}(n) \tag{6}$$

where $v_{ij}(n)$ is given by

$$v_{ij}(n) = \nabla_{ij} E(n) + \alpha v_{ij}(n-1) \tag{7}$$

where α is the momentum parameter. The learning rate parameters are still adapted as in (4).

The same test performed with the basic algorithm was repeated with the addition of the momentum term both for conventional and adaptive backpropagation. The test was performed with the same initial learning parameters used before, and setting $\alpha = 0.5$ and $\alpha = 0.9$. The test was also performed for $\alpha = 0.9$ with the initial learning rate parameter decreased by one order of magnitude ($\eta = 0.05$).

The results of this test, listed in table 1, part B, show that, for large values of ϕ, conventional backpropagation with a large momentum can provide slightly better results than the adaptive technique. However, several points favor the new method. First, as suggested by Sutton in [Sutton86], there are reasons to believe that in many practical problems valleys will often tend to be parallel to the axes, and in this case the adaptive algorithm decreases the number of iterations needed to converge, by some orders of magnitude. Second, the adaptive technique is much less sensitive to the initial value of η, as can be seen by comparing the columns corresponding to $\alpha = 0.9$, $\eta = 0.5$, and $\alpha = 0.9$, $\eta = 0.05$, in table 1. Finally, one should also note that the straight valley used in this example is a near optimal shape for conventional backpropagation with momentum, and this is not often the case found in real neural network problems. These ideas are confirmed by the results of several experiments that are reported in the next section.

5 Simulation Results

Several experiments were conducted in order to evaluate the performance of the adaptive algorithm. In all tests reported, a symmetric arc-tangent activation function scaled between -1 and 1 was used. The target output for binary values was -0.8 and +0.8. Each test was performed 20 times, with randomly generated initial weights in the range [-1,1]. The result of each test is given by the mean and standard deviation of the number of iterations needed to reach solution. In all tests performed, the u and d parameters of the adaptive algorithm were set to 1.2 and 0.7, respectively. As mentioned before, the adaptive algorithm rejects the new weight vector when an increase in the output error is observed, and that iteration is only used as a reference for learning rate adaptation. These "false" iterations are however included in the counting of the number of iterations.

The first experiment was a very simplified version of a digit recognition task. The network had 25 input units, 9 hidden units and 4 output units, and tried to recognize the digits 0, 1, 2 and 3. Each input pattern was a 5×5 pixel binary image of a digit. The training set was formed by 60 patterns, 15 for each digit. The recognition of each digit was indicated by the activation of a single output unit. A root mean squared error lower than 0.03 (averaged over the four outputs and the whole training set) was used as convergence criterion. The main goal of this first test was to check the sensitivity of the algorithm to different sets of initial parameters. The learning parameters and the results of each test are listed in table 5 (the value of η in the adaptive algorithm refers to initial values). The learning rate parameters were chosen after some preliminary tests using the conventional backpropagation algorithm, in order to select values that resulted in a good performance of this method.

The results show that the adaptive algorithm performed much better than conventional backpropagation in all cases. Similar results were obtained for all sets of initial learning rate parameters when using the adaptive method, although significant changes are observed for conventional backpropagation, showing the very low sensitivity of the adaptive method to initial values of the learning rate parameter. Note, however, that our experience with this algorithm suggests that it is generally preferable to choose a small

η	α	BP		Adaptive BP	
		Mean	Std. Dev.	Mean	Std. Dev.
0.05	0.0	1298.2	317.1	88.4	16.5
	0.5	733.3	169.4	54.4	6.0
	0.9	836.7	375.0	75.4	19.4
0.005	0.0	10748.3	2661.8	97.5	15.3
	0.5	6219.8	1937.7	64.2	5.9
	0.9	1806.6	425.0	84.5	12.1

Table 2 - Simulation results for digit recognition. In tables 2-4, each entry shows the mean and standard deviation of the number of iterations needed to reach solution.

initial value for η, and let the adaptive process increase the learning rate parameters to the "correct" values. This option can avoid initial "false" iterations.

In order to assess the importance of using independent learning rate parameters, this test was repeated, for one specific weight initialization, with the conventional backpropagation algorithm and on-line (manual) adjustment of the learning rate and momentum parameters. The best result obtained, after a few trials, was 187 iterations. With fixed values $\eta = 0.05$ and $\alpha = 0.9$, and the same initial conditions, conventional backpropagation reached solution after 857 iterations, while the adaptive technique performed the same task in 57 iterations. Despite the informal nature of this test, it gives an idea of the capabilities of the proposed technique.

The second test performed was a standard 10-5-10 encoder problem. The network had 10 input units, 5 hidden units and 10 output units. Each input pattern was a binary valued vector, where only one element was active at a time. The network tried to map the input pattern into the output units. This test was performed in two steps. In the first one, a simple binary error was used as convergence criterion, i.e., only the sign of each output unit was tested in order to stop the process (however, conventional analog error propagation was used for learning). In the second step, the process was carried on until a root mean squared error lower than 0.1 was obtained. This test was performed using the conventional and adaptive algorithms, and also the cosine adaptation technique proposed by Chan and Fallside [Chan87]. The initial parameters and results for each test are listed in table 3. For the adaptive and cosine techniques, η refers to initial values. For the cosine adaptation technique, α refers to the momemtum factor of this algorithm (note that this is not exactly the conventional momentum factor, but it has a similar role and must also be set in the range [0,1]).

The results show that both the cosine and adaptive techniques yield significant speed gains over conventional back-propagation. For conventional backpropagation without momentum, only 3 trials out of 20 were able to reach the RMS criterion in less than 20000 iterations, so statistics of this test were not computed. Note that while the cosine and adaptive techniques provide similar results with the binary error criterion,

Stopping Crit. →			Bin		RMS < 0.1	
Method	η	α	Mean	Std. Dev.	Mean	Std. Dev.
BP	0.5	0.0	130.0	93.0	> 16000	-
BP	0.1	0.8	115.8	64.62	5181.6	3790.3
Adp.	0.005	0.0	73.0	19.9	116.8	38.7
Adp.	0.005	0.8	48.9	8.7	79.1	18.2
Cosine	0.005	0.0	98.7	33.5	280.3	175.0
Cosine	0.005	0.8	48.4	15.2	129.6	73.3

Table 3: Simulation results for 10-5-10 encoder

Method	η	α	Mean	Std. Dev.
BP	0.05	0.0	5513.5	1994.3
BP	0.05	0.9	2580.9	2843.5
Adp.	0.005	0.0	1828.0	1119.9
Adp.	0.005	0.9	421.4	325.3
Cosine	0.005	0.0	7069.5	4991.0
Cosine	0.005	0.9	2419.9	1909.6

Table 4: Simulation results for 3-to-8 decoder

the former yielded better results for the RMS criterion.

The last test was proposed by Jacobs in [Jacobs88]. The network implements a binary 3-to-8 decoder, but a hidden layer with a single unit is used. The network had 3 input units, a first hidden layer with a single unit, a second hidden layer with 8 units and an output layer with 8 units. In this problem, only the binary error was used as convergence criterion.

The results for this test, listed in table 4, show that the adaptive technique again yields the best performance in all cases, but some further conclusions are possible when comparing the results of this test with previous ones. First, the inclusion of the momentum term in the adaptive technique seems to have a much more important role here than in previous problems. This fact suggests the presence of narrow diagonal valleys in the error surface, where the adaptive technique without momentum yields worse results. Second, the cosine adaptation algorithm provides here similar or worse results than conventional backpropagation. It was observed that, in this case, the cosine technique provides small adaptation rates, which may be a possible explanation for this unexpected result. One should also note that the results for the adaptive technique are similar to those reported by Jacobs in [Jacobs88] with "optimal" initial parameters[1].

A very significant feature of the adaptive technique is its low sensitivity to initial learning rate parameters (note that in all problems reported the same initial learning

[1]We used here a different activation function, but this does not seem to be of much importance

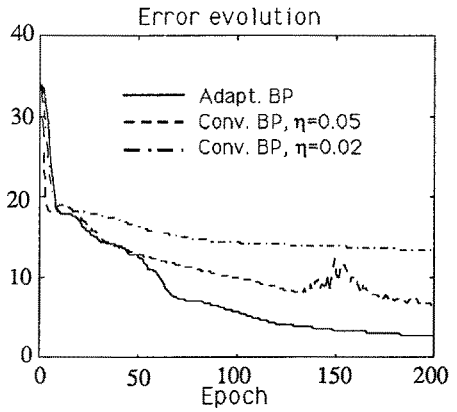

Fig 1a - Error evolution: Adaptive vs. conventional BP

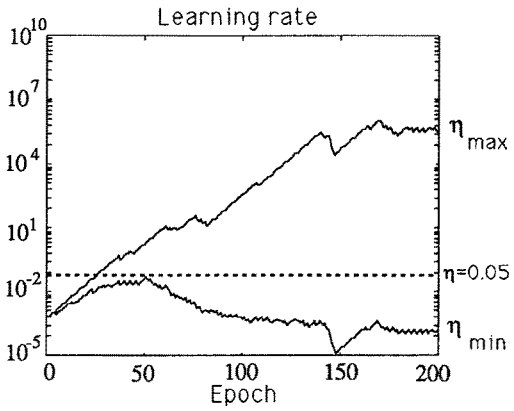

Fig 1b - Learning rate evolution in adaptive BP

rate parameters can be used). Another interesting feature of the adaptive technique is the large dynamic range that is observed in the learning rate parameters. Figure 1a shows the error evolution for a particular run of the 3-to-8 encoder problem, with adaptive and conventional backpropagation, and $\alpha = 0.9$ (for conventional backpropagation, error evolution is shown for $\eta = 0.05$ and $\eta = 0.01$; note that the former, while more stable, yields a slower convergence). Figure 1b shows the evolution of the maximum and minimum learning rate parameters for each epoch for the same run of this problem with the adaptive technique. A logarithmic scale was used for the y-axis in order to obtain a convenient display of the dynamic range. Note that, after the initial adaptation phase, the increase of the dynamic range of learning rate parameters corresponds to a considerable improvement of the adaptive technique over conventional backpropagation.

6 Real-Time Learning

In real-time learning procedures, the weight update is performed after the presentation of each input/output pair. When the input training set is of stochastic nature and obeys certain conditions, the learning process can be treated as a stochastic approximation procedure, and convergence to a local minimum occurs if the learning rate parameter approaches 0 in a non-summable sequence. To our knowledge, there is no such equivalent result when dealing with deterministic training sets. However, real-time training procedures are often used in this situation, yielding excellent results.

The main concept behind real-time training is that, after the presentation of the whole training set, the overall weight update is still performed approximately in the gradient direction. This principle suggests a simple extension of the adaptive backpropagation algorithm to real-time learning procedures: the adaptation of the learning rate parameters is performed as in (4), but $\Delta E(n)$ and $\Delta E(n-1)$ are now gradient

estimates obtained from the accumulated "gradients" on a sweep over all training patterns.

While adaptive backpropagation can be guaranteed to converge to a local minimum when used in the batch mode, its behavior in the real-time mode is less understood. In fact, when a fixed learning rate parameter is used in a stochastic approximation procedure, the mean output error will generally decrease to a certain value, where a stationary random oscillation around a local minimum is reached. After this point, the reduction of the mean output error is only possible by gradually decreasing the learning rate parameter. While it is our belief that the adaptive technique can improve the initial phase of a real-time learning procedure, its behavior when an oscillation around a minimum is reached is not well known yet. We should mention, however, that real-time adaptive backpropagation as proposed above was already used by us, with good results, in a large task of noise reduction in speech signals.

7 Conclusions

A simple and local technique for the adaptation of the learning rate parameters for batch training procedures was introduced. It was shown that this technique is able to improve convergence speed in real neural network problems, while keeping a low sensitivity to initial learning rate parameters. An heuristic extension of the technique to real-time learning procedures was suggested, but further work must be done in order have a complete evaluation of its performance. Future work will also include a performance comparison with "quickprop", an acceleration technique proposed by Fahlman [Fahlman88] which only recently came to our knowledge.

References

[Sutton86] Richard S. Sutton. "Two Problems With Backpropagation and other Steepest-Descent Learning Procedures for Networks", *Proceedings of the 8th. Annual Conf. of the Cognitive Science Society*, 1986, pp 823-831.

[Rumelhart86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning Internal Representations by Error Propagation", in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Cambridge, MA: MIT Press, 1986.

[Chan87] L-W. Chan and F. Fallside. "An Adaptive Training Algorithm for Back Propagation Networks", *Computer Speech & Language*, vol. 2., Sep/Dec. 1987, pp 205-218.

[Kesten58] H. Kesten. "Accelerated Stochastic Approximation", *Annals of Mathematical Statistics*, 29, 1957, pp 41-59.

[Jacobs88] R. Jacobs. "Increased Rates of Convergence Trough Learning Rate Adaptation", *Neural Networks*, Vol 1, N. 4, 1988.

[Fahlman88] S. Fahlman. "Faster-Learning Variations on Back-Propagation: An Empirical Study", *Proc. of the 1988 Connectionist Models Summer School*, Carnegie Mellon, 1988.