Accelerating Backpropagation through Dynamic Self-Adaptation

Ralf Salomon

Institute for Applied Computer Science (Sekr. FR 5-6), Technical University of Berlin, Franklinstrasse 28/29, D-10587 Berlin, Germany

J. Leo van Hemmen

Physik-Department, Technische Universität München, D-85747 Garching bei München, Germany

Abstract

Standard backpropagation and many procedures derived from it use the steepest-descent method to minimize a cost function. In this paper, we present a new genetic algorithm, dynamic self-adaptation, to accelerate steepest descent as it is used in iterative procedures. The underlying idea is to take the learning rate of the previous step, to increase and decrease it slightly, to evaluate the cost function for both new values of the learning rate, and to choose the one that gives the lower value of the cost function. In this way, the algorithm adapts itself locally to the cost function landscape. We present a convergence proof, estimate the convergence rate, and test the algorithm on several hard problems. As compared to standard backpropagation, the convergence rate can be improved by several orders of magnitude. Furthermore, dynamic self-adaptation can also be applied to several parameters simultaneously, such as the learning rate and momentum.

Keywords: Gradient method, Steepest descent, Backpropagation, Self-adaptation, Acceleration.

1 Introduction

The ability to learn is a most valuable property of a neural network. Usually, the task of the network is to map input data onto output data and its behavior is governed by a set of parameters, or weights, $\mathbf{w}$. Learning means adjusting the weights so that input data are mapped onto minima of a quality or cost function E, which is taken to be positive. Backpropagation is a learning pro-

cedure that adjusts the $\mathbf{w}$ through a steepest descent with respect to E in weight space,

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla E(\mathbf{w}_t) \qquad , \tag{1}$$

where η is the so-called *learning rate* and $\mathbf{w}_t$ is a vector representing the weights at iteration step t. Though the procedure is widespread, it has certain, problem-dependent, disadvantages. First, convergence is fast only if the parameter setting is nearly optimal. Furthermore, the convergence rate decreases rapidly as the problem size increases. Finally, convergence is guaranteed only if the learning rate η is small enough. The main problem then is to determine *a priori* what "small enough" means. In other words, for a shallow minimum the learning rate is often too small whereas for a narrow minimum it is often too large and the procedure never converges.

Since the learning rate or procedure parameters such as momentum cannot be determined a priori one needs a procedure that adjust itself to the landscape generated by the cost function E. Precisely this is performed by *dynamic self-adaptation*, an algorithm which we will analyze in detail below. A preliminary report (without proofs) has been presented by Salomon (1990). Here we present both a convergence proof and an estimate of the convergence rate for arbitrary dimensions. It is to be constantly borne in mind, though, that we treat *feed-forward*, and other, structures which give rise to a well-defined cost function. In addition, we test the algorithm on various hard problems and analyze its performance. Before doing this, however, it may be well to review presently available acceleration methods.

There are a few methods – including our own – that do not use higher-order derivatives. For example, van Ooyen and Nienhuis (1992) change the cost function whereas Rigler et al. (1991) rescale their variables. The new, and distinctive, feature of *dynamic self-adaptation* is that it adapts itself steadily to the cost-function landscape by choosing that step out of two (or finitely many, some smaller, some bigger) which *optimizes* the decrease of the cost function. Most procedures, however, root in higher-order derivatives, e.g., by varying the learning rate per weight or calculating the optimal learning rate per iteration step. We list the more recent procedures and refer the reader to Gill et al. (1981) for an excellent discussion of the older literature.

Becker and le Cun (1989) use Newton and quasi-Newton methods. Their idea is to exploit the information contained in the second derivative, the Hessian, of the cost function E. Since the calculation of the inverse Hessian, which they need, is rather time consuming Becker and le Cun use an iterative approximation of the inverse Hessian. It turns out that only a maginal increase of the convergence rate is obtained.

Jacobs (1988) as well as Silva and Almeida (1990) determine individual learning rates for the components of the vector $\mathbf{w}$ separately. These authors use a different heuristics to adapt the individual learning rates to the information obtained in earlier iteration steps. Jacobs needs three new parameters which, however, have the very same drawback as the original learning rate. In his experiments, Jacobs needed a huge amount of trial runs before he had arrived at the 'right' parameter values. In contrast, we need only perform a single run. The heuristic of Silva and Almeida needs only two extra parameters but exhibits unstable convergence behavior, as does Jacobs' procedure.

Kramer and Sangiovanni-Vincentelli (1987) exploit the well-known line search (for further details, see Luenberger (1984) and Press *et al.* (1987) and use the *false position* method in conjunction with the Polak-Ribiere formula. The idea behind this approach is to calculate an optimal learning rate for each iteration step but, due to the line search, the procedure is extremely time consuming.

Another interesting procedure is the gradient re-use algorithm of Hush and Salas (1988). They re-use the gradient as many iteration steps as the decrease of the cost function lasts and determine the new value of the learning rate depending on the re-use count. Unfortunately the increase of the convergence rate is rather moderate.

The procedure that is closest to ours but still radically different is Battiti's (1989) "bold driver". It is also a gradient descent method. One starts with a learning rate η. If the function E to be minimized decreases, one takes $\eta := \rho\,\eta$ where $\rho \approx 1.1$. And so on, until E increases. One then puts $\eta := \sigma\,\eta$ with $\sigma \approx 0.5$, iterates until a step that decreases E is found, and proceeds until E increases again (if ever). So it adapts the learning rate to the cost function landscape only every now and then. All in all, this is a nice idea but the procedure is not a truly dynamic self-adaptation. The momentum term is *not* handled this way, which is to be contrasted with the present algorithm that handles learning rate, momentum, *and whatever additional parameter* , on an equal footing. Furthermore, it adapts itself steadily to the cost function landscape. A detailed comparison will be presented in section 3.3.

To fully appreciate the advantage of the *dynamic* self-adaptation algorithm it is necessary to consider the *total* amount of time needed for the whole learning procedure so as to determine the convergence rate. The total amount of time is given by the product of the number of iteration steps *and* the time needed for each iteration step. Procedures such as line search or the Newton method reduce the total number of iteration steps but increase drastically the time needed for one iteration step. So the net gain is only a slight increase or even a decrease of the convergence rate. The problem which neither of these methods solves (including our own) is that of local minima. A gradient descent algorithm never does that and, hence, this problem will not be discussed here

any further.

2 Dynamic self-adaptation of the learning rate

The dynamic self-adaptation algorithm is a two-step procedure that first induces a *mutation*, compares the new value with the one corresponding to the previous situation, and then *selects* the best. Thereby it adapts itself to the local information available on the cost function. Steepest descent algorithms can be performed without normalization, as in (1), or with normalization,

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_{t+1} \frac{\nabla E(\mathbf{w}_t)}{|\nabla E(\mathbf{w}_t)|} \equiv \mathbf{w}_t - \eta_{t+1}\, \mathbf{e}_k \tag{2}$$

where $\mathbf{e}_t$ is the unit vector in the direction of $\nabla E(\mathbf{x}_t)$. The new learning rate simply is

$$\eta_{t+1} = \begin{cases} \eta_t\, \zeta & \text{if} \quad E(\mathbf{w}_t - \eta_t\, \mathbf{e}_k\, \zeta) \leq E(\mathbf{w}_t - \eta_t\, \mathbf{e}_k/\zeta) \\ \eta_t/\zeta & \text{otherwise} \end{cases} , \tag{3}$$

That is, out of the two possibilities, $\eta_t\, \zeta$ and η_t/ζ, the algorithm takes the one that gives the lower value for the cost function, i.e., the one that is best. So at *each* step the procedure adapts itself in a simple but straightforward way to the cost function landscape. Though ζ in principle can be any number, we will show that the choice $\zeta = 1.8$ is about optimal. The procedure stops as soon as $\nabla E(\mathbf{x}_t)$ vanishes. If at stage k the algorithm has used ζ, then $1/\zeta$ in (3) is the mutation. In passing we note that we could have replaced ζ and $1/\zeta$ by arbitrary numbers $\zeta_1 > 1$ and $0 < \zeta_2 < 1$; for example, $\zeta_1 = 1.8$ and $\zeta_2 = 0.5$. The above choice, however, is simpler. In an evolutionary vein, we could also have considered a finite number of steps and selected the one that is best. Since the procedure adapts itself to the *local* constraints of the landscape, there is no need for doing so and IT SUFFIces to stick to the two which we have advertized.

Without normalization, one replaces $\mathbf{e}_t$ by $\nabla E(\mathbf{e}_t)$. The normalization in (2) and (3) is convenient but not necessary. It the curvature of the cost function E that favors normalization. In neural networks, the cost function is gentle but at the same time rather *cleft* (containing many plateaus and narrow rifts) and, therefore, problems frequently arise because of large variations of the gradient.

In passing we note that in neural network applications the usual sigmoidal response function $f(x) = 1/(1 + e^{-x})$ can give rise to problems, if the η's

are too large. Then the derivative $f'(x) = f(x)[1 - f(x)]$ vanishes due to the rounding effect in real arithmetic and the procedure gets stuck.

We would like to stress that the dynamic self-adaptation algorithm is not restricted to determining the learning rate. Later on we will show that it can be applied straightforwardly to other procedure parameters such as the momentum α (Rumelhart and McClelland, (1986), p. 330). Momentum is a kind of memory that incorporates the weight change of the previous step and in this way slows down useless oscillations. More precisely, we have $\mathbf{w}_{t+1} = \mathbf{w}_t + \Delta\mathbf{w}_{t+1}$ with

$$\Delta\mathbf{w}_{t+1} = -\eta\,\nabla E(\mathbf{w}_t) + \alpha\,\Delta\mathbf{w}_t \tag{4}$$

and $0 \leq \alpha \leq 1$. Our implementation allows adaptation of both η and α by the very same procedure. It is most straightforward and, from the adaptation point of view, advantageous to adapt η and α alternately. This proposal stems from the very idea that the procedure adapts itself steadily to the cost-function landscape. We now have to adapt *two* parameters and, thus, do so alternately.

3 Convergence proof and acceleration estimates

In this section, we present the idea of a convergence proof (full details are to be found in the Appendix) and estimate the convergence rate of the dynamic self-adaptation algorithm. Plainly, it is impossible to give a convergence proof for any type of problem and any cost function. We first assume that we are already in the neighborhood of a minimum of the cost function E and approximate E by a positive-definite quadratic form. Later on we will show that convergence also holds in a more general context. Throughout this section we focus on adaptation with gradient normalization. The modification of the proof in the case without normalization is straightforward and will only be sketched.

The main result of this section is that for *each* value of the learning rate (i) convergence is guaranteed and (ii) the convergence rate is comparable to that of the gradient method proper with *optimal* parameter setting. We first treat the one-dimensional case, then turn to two dimensions, and finally present an argument for the n-dimensional case.

In the neighborhood of a minimum $\mathbf{x}^*$ with $f(\mathbf{x}^*) = 0$ we can write

$$f(\mathbf{x}) = f(\mathbf{x}^*) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^*)^T\, H(\mathbf{x}^*)\,(\mathbf{x} - \mathbf{x}^*)\ . \tag{5}$$

where $H(\mathbf{x}^*) = \left(\frac{\partial^2}{\partial x_i \partial x_j}\right)$ is the Hessian of f at the minimum $\mathbf{x}^*$. In (5) we

have neglected all terms of higher order than two. After a principal-axis transformation the Hessian assumes the simple form $\mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ with $\lambda_i > 0$. There is no harm in performing a uniform translation so that $\mathbf{x}^* = 0$. We then end up with

$$f(\mathbf{x}) = \frac{1}{2}\,\lambda_1\,x_1^2 + \frac{1}{2}\,\lambda_2\,x_2^2 + \cdots + \frac{1}{2}\,\lambda_n\,x_n^2 \quad . \tag{6}$$

From now on, we use the following symbols:

- $f(\mathbf{x})$ and $f^{'}(\mathbf{x})$ are the function under consideration and its first derivative.
- $\mathbf{x}_0$ is a randomly chosen starting point.
- $\mathbf{x}_t$ is the actual value of $\mathbf{x}$ after iteration step t.
- η_t is the value of the learning rate η in iteration step t; its initial value is η_0.
- g_t is the accuracy of $\mathbf{x}_t$, defined through $|\mathbf{x}_{t'} - \mathbf{x}^*| = |\mathbf{x}_{t'}| \leq g_t$ for $t^{'} \geq t$; here $\mathbf{x}^* = 0$.

3.1 One-dimensional case

We study the parahabolf $f(x) = \frac{1}{2}\lambda x^2$ with $\lambda > 0$. For the sake of convenience we assume $x_t > 0$; this does not necessarily imply $x_{t+1} > 0$. Before turning to dynamic self-adaptation we briefly consider the steepest-descent method proper.

A Steepest descent with normalization

For a steepest descent with normalization we obtain

$$x_{t+1} = x_t - \eta\frac{f^{'}(x_t)}{|f^{'}(x_t)|} = x_t - \eta\,\mathrm{sgn}(x_t) \quad . \tag{7}$$

Due to the normalization the step size is η. For a constant learning rate η we then get that the number of iteration steps n amounts to $|x_0|\,/\eta$ with final accuracy $g_n = \eta/2$.

B Steepest descent without normalization

Since $f^{'}(x) = \lambda\,x$ we find

$$x_{t+1} = x_t - \eta\,f^{'}(x_t) = x_t(1 - \lambda\,\eta) = x_0(1 - \lambda\ \eta)^{t+1} \quad . \tag{8}$$

Table 1
A typical example of convergence for $f(x) = \frac{1}{2}x^2$, $x_0 = 54$, $\eta_0 = 2$, and $\zeta = 2$.

t	x_t	η_t	u_t	$x_t - \eta_t \frac{1}{\zeta}\,\mathrm{sgn}(x_t)$	$x_t - \eta_t\,\zeta\,\mathrm{sgn}(x_t)$	η_{t+1}/η_t	x_{t+1}	η_{t+1}
0	54	2	2.5	53	50	ζ	50	4
1	50	4	5.0	48	42	ζ	42	8
2	42	8	10.0	38	26	ζ	26	16
3	26	16	20.0	18	-6	ζ	-6	32
4	-6	32	40.0	10	58	$1/\zeta$	10	16
5	10	16	20.0	2	42	$1/\zeta$	2	8
6	2	8	10.0	-2	-14	$1/\zeta$	-2	4
7	-2	4	5.0	0	6	$1/\zeta$	0	2

To get convergence we need $|1 - \lambda\,\eta| < 1$. If so, the accuracy at time t is

$$g_t = \left| x_0\,(1 - \lambda\,\eta)^t \right| \qquad . \tag{9}$$

C Dynamic self-adaptation with normalization

Because $f'(x) = \lambda x$, the algorithm assigns to x_{t+1} the value

$$x_{t+1} = x_t - \eta_{t+1}\frac{f'(x_t)}{|f'(x_t)|} = x_t - \eta_{t+1}\mathrm{sgn}(x_t) \tag{10}$$

with

$$\eta_{t+1} = \begin{cases} \eta_t\,\zeta & \text{if} \qquad f(x_t - \eta_t\,\mathrm{sgn}(x_t)\,\zeta) \le f(x_t - \eta_t\,\mathrm{sgn}(x_t)/\zeta)\,, \\ \eta_t/\zeta & \text{otherwise} \end{cases} \tag{11}$$

Throughout what follows we take $\zeta > 1$. Before proceeding it is worthwhile to consider a typical example with $f(x) = \frac{1}{2}x^2$, $x_0 = 54$, $\eta_0 = 2$ and $\zeta = 2$. The resulting iteration steps are presented in Table 1 and Figure 1. The behavior is typical in that, for any t, there exists a so-called *reversal interval* $U_t = [-u_t, u_t]$ such that η_t increases as long as x_t is outside this interval and that it *decreases monotonically* as soon as x_t is in this interval. We will show

$$U_t = [-u_t, u_t] \quad \text{with} \quad u_t = \eta_t\,\frac{\zeta^2 + 1}{2\,\zeta} \qquad . \tag{12}$$

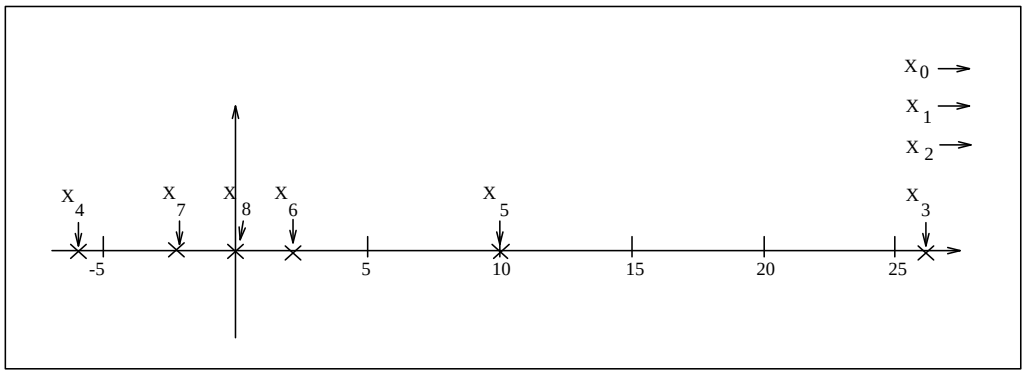

Fig. 1. Plot of the convergence described in Table 1.

In passing we note that the algorithm itself does not need an explicit computation of the reversal interval. It just exists and is a useful notion to formulate the convergence proof. Once x_t is inside the reversal interval it will remain so forever provided $\zeta < \zeta_c$ where $\zeta_c \approx 1.839$. Furthermore, convergence is guaranteed for any $\zeta > 1$. It turns out, however, that the convergence rate is *optimal* for $\zeta = \zeta_c$. The convergence proof itself is to be found in the Appendix. There we also analyze the process of convergence in detail and show how to optimize ζ so as to arrive at $\zeta_c = 1.839$.

D Dynamic self-adaptation without normalization

Without normalization the algorithm is given by

$$x_{t+1} = x_t - \eta_{t+1} f'(x_t) \tag{13}$$

with

$$\eta_{t+1} = \begin{cases} \eta_t \, \zeta & \text{if} \quad f(x_t - \eta_t \, \zeta \, f'(x_t)) \leq f(x_t - \eta_t \, \frac{1}{\zeta} \, f'(x_t)) \\ \eta_t \, \frac{1}{\zeta} & \text{otherwise} \end{cases} , \tag{14}$$

For $f(x) = \frac{1}{2} \lambda \, x^2$ and $f'(x) = \lambda \, x$ the procedure performs two test steps and chooses either

$$x_{t+1} = x_t (1 - \lambda \, \eta_t \, \zeta) \tag{15}$$

or

$$x_{t+1} = x_t \, (1 - \lambda \, \eta_t / \zeta) \tag{16}$$

Because the algorithm with normalization is favorable to hard problems we refrain from giving details about the procedure without normalization; full details are provided by Salomon (1991). The main idea of the proof is introducing a reverse learning rate $\eta_u = \frac{2}{\lambda}\frac{\zeta}{\zeta^2+1}$ in analogy to the reversal interval.

E Convergence rate

For the performance quality the convergence rate is of utmost importance. In all cases we assume a conservative choice of the learning rate, which means $\eta_0 \ll x_0$. For the steepest-descent method proper there is a little problem in that the λ in $f(x) = \frac{1}{2}\lambda x^2$ is not known a priori. For dynamic self-adaptation this information is not needed. We list the performance quality of the various procedures.

A. Steepest descent with normalization: Before crossing the origin we have $x_t = x_0 - \eta t$, so convergence is very slow. This simple procedure gives at best the accuracy $g_t = \eta/2$.

B. Steepest descent without normalization: Provided $\eta < 2/\lambda$, we get an exponential convergence of the accuracy given by $g_t = x_0(1-\eta\lambda)^t$. Plainly, in real-life problems the situation for this procedure is much worse since quadratic approximation does not hold farther away from the minimum. Nor do we know λ.

C. Dynamic self-adaptation with normalization: For $\zeta \leq \zeta_c$ we now have two phases, a start phase where we get an exponential increase of the learning rate (Theorem 2) and a final phase where we obtain an exponential decrease of the learning rate (Theorem 3). Only the latter is time consuming. The proof of Theorem 2 shows that start phase lasts approximately $\tau < \log(x_0/\eta_0)/\log\zeta$. In the final phase we get that the accuracy equals at least

$$g_t = \eta_0 \frac{1}{\zeta^{t-2\tau}} \cdot \frac{\zeta^2+1}{2\zeta} \quad . \tag{17}$$

Taking $\zeta = 1.3$ or $\zeta_c = 1.839$ we obtain a convergence rate comparable to the one of the steepest descent method without normalization but with *optimal* learning rate. For $\zeta > \zeta_c$ one has to replace $t - 2\tau$ by $(t-2\tau)/\zeta$ in (17) as a consequence of Theorem 5.

D. Dynamic self-adaptation without normalization: This procedure has the very same two phases as the previous one. The optimal learning rate, however, oscillates around $\frac{1}{\lambda}\frac{2\zeta}{\zeta^2+1}$. During the start phase we find

$$\tau = \log\left(\frac{2}{\lambda\,\eta_0}\,\frac{\zeta}{\zeta^2+1}\right)/\log\zeta \tag{18}$$

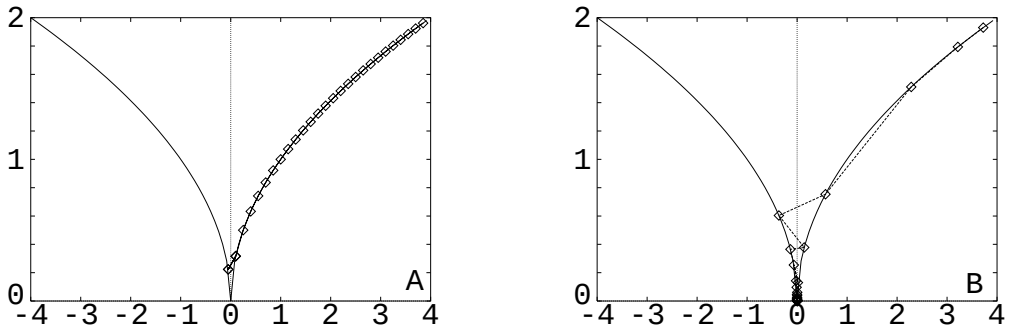

Fig. 2. Comparison of (A) the gradient method and (B) dynamic self-adaptation. Both are taken with normalization and applied to the function $f(x) = \sqrt{|x|}$ during 30 iteration steps starting at $x_0 = 4$. We have taken $\eta = 0.15$ in **A** and $\eta_0 = 0.15$ with $\zeta = \zeta_c$ in **B**. The minimum at $x = 0$ has a square-root singularity. The last 27 steps of steepest descent with normalization in **A** show a typical behavior: The procedure first approaches the minimum but will never reach it as it oscillates forever between the lowest two points. On the other hand, dynamic self-adaptation approaches the minimum monotonically and gives function values below 0.01 already after 20 steps.

whereas during the final phase we obtain for the accuracy at least

$$g_t = x_0 \left(1 - 2\,\frac{\zeta}{\zeta^2 + 1}\right)^{t-\tau} \tag{19}$$

which is *independent* of the actual value of λ. If, for instance, ζ equals 1.3, then we have $g_t = x_0\, 0.03^{t-\tau}$.

F Hard test: Square-root singularity

It may be worthwhile to consider a function that is not differentiable and cleft at its minimum, such as $f(x) = \sqrt{|x|}$. Figure 2 depicts the performance of the steepest-descent method proper and the dynamic self-adaptation algorithm, both with normalization. The convergence of the latter procedure towards zero is fast whereas that of the former needs no further comment. The steepest-descent method without normalization is even worse. In short, dynamic self-adaptation also handles non-quadratic minima fast and efficiently.

3.2 Two-dimensional case

In two dimensions, a quadratic approximation takes the form

$$f(\mathbf{x}) = \frac{1}{2}\,\lambda_1\, x_1^2 + \frac{1}{2}\,\lambda_2\, x_2^2 \tag{20}$$

where $0 < \lambda_1 \leq \lambda_2$. Through the *uniform* transformation $\sqrt{\lambda_1}\mathbf{x} \rightarrow \mathbf{x}$ we can rewrite this

$$f(\mathbf{x}) = \frac{1}{2}\, x_1^2 + \frac{1}{2}\,\lambda\, x_2^2 \tag{21}$$

with $\lambda = \lambda_2/\lambda_1 \geq 1$. The case $\lambda = 1$ has rotational invariance and, therefore, is equivalent to the one-dimensional case. The new aspect of two dimensions is the fact that the angle φ between the gradient (current direction of the steepest-descent process) and the radius (optimal direction) is nonzero. Through a minimization of $\cos\varphi$ on the ellipse $x_1^2 + \lambda\, x_2^2 = 2\,c$ we find $\cos\varphi_{\max} = 2\sqrt{\lambda}/(\lambda+1)$. For the case $\lambda \gg 1$, which is a worst case and also called 'the cigar', we then obtain that $\varphi_{\max}$ is about $\pi/2$ and, thus, a very slow convergence due to oscillations results. Only for staring points $\mathbf{x_0} = (x_{01}, x_{02})$ with one of the x_{0i} vanishing, i.e., for a motion along one of the principal axes, do we end up with $\varphi = 0$ and, thus, with the one-dimensional case. We, therefore, turn to the oscillations and, as in the previous section, consider four generic cases. We will stick to the worst case with $\lambda \gg 1$.

A Steepest descent with normalization

For $\lambda \gg 1$, all four procedures first optimize x_2, leaving x_1 about constant, until $\lambda\, x_2$ becomes of the same order of magnitude as x_1. For a *gradient* descent algorithm this is natural. When the gradient makes an angle of about $\pm\pi/4$ with the x_1-axis, oscillations set in. Turning to the cigar ($\lambda = 100$) in figure 3, we note that for steepest descent with normalization but *constant* learning rate η the system moves very slowly along the x_1-axis with x_2 jumping between $\pm\eta/2$. As it proceeds the 'orbit' become steeper and steeper. Let us define $\Delta\, x_1 = x_{t+1,1} - x_{t,1}$. We have

$$\Delta\, x_1 = -\eta \frac{(\nabla f)_1}{\|\nabla f\|} = -\frac{\eta\, x_1}{\sqrt{x_1^2 + (\lambda\eta/2)^2}} \approx -\frac{2\, x_1}{\lambda} \tag{22}$$

as $x_1 \rightarrow 0$. Thus we find

$$x_{t,1} \approx x_{0,1}\, e^{-\frac{2\,t}{\lambda}} \tag{23}$$

and $x_{t,1}$ approaches zero very slowly for large λ. During this process $|x_{t,2}|$ sticks to $\eta/2$ and the final error ($t \rightarrow \infty$) in f is $\lambda\,\eta^2/8$. The convergence rate in the 1-direction is given by equation (23).

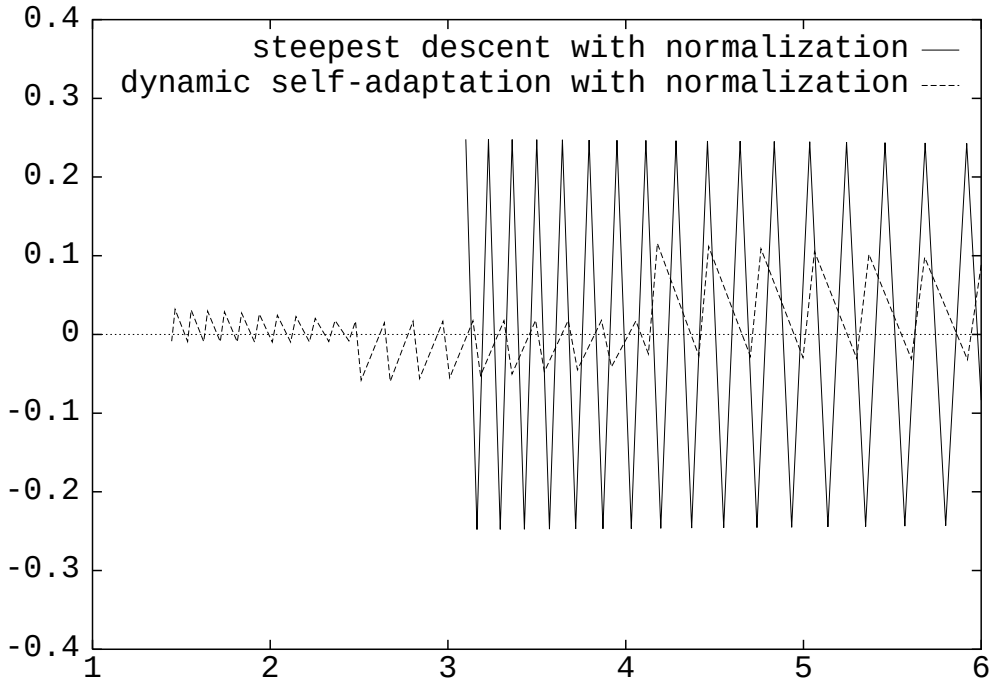

Fig. 3. The cigar $f(\mathbf{x}) = x_1^2 + \lambda\, x_2^2$ with $\lambda \gg 1$ is a notoriously difficult case leading to oscillations. Here we show the last steps out of 100 altogether for $\lambda = 100$ and starting point $\mathbf{x}_0 = (20, 5)$. The solid line represents the steepest descent method proper ($\eta = 0.5$) and so does the dashed line for dynamic self-adaptation ($\eta_0 = 0.5$), both with normalization. At the end points, the function values $f(\mathbf{x}_{100})$ are 7.88 and 1.05, respectively. Note the different scales of the horizontal and vertical axis.

B Steepest descent without normalization

Since

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \nabla f(\mathbf{x}_t) \Leftrightarrow \begin{pmatrix} x_1(t+1) \\ x_2(t+1) \end{pmatrix} = \begin{pmatrix} x_1(t)\,(1-\eta) \\ x_2(t)\,(1-\lambda\,\eta) \end{pmatrix} \tag{24}$$

we cannot but conclude $|1 - \lambda\,\eta| < 1$ if we want convergence to the minimum at $(0, 0)$. That is, $0 < \eta < 2/\lambda$ is a necessary and sufficient condition for convergence. For large λ, then, convergence is extremely slow in the 1-direction. More precisely, we get asymptotically

$$\|\mathbf{x}_{t+1}\| \sim \|\mathbf{x}_0\|(1-\eta)^t \geq \|\mathbf{x}_0\| \left(1 - \frac{2}{\lambda}\right)^t \sim \|\mathbf{x}_0\| e^{-\frac{2\,t}{\lambda}} \tag{25}$$

as the 1-direction becomes dominant. Note that we must know λ beforehand (we usually do not), if we want to get convergence. If so, the error approaches

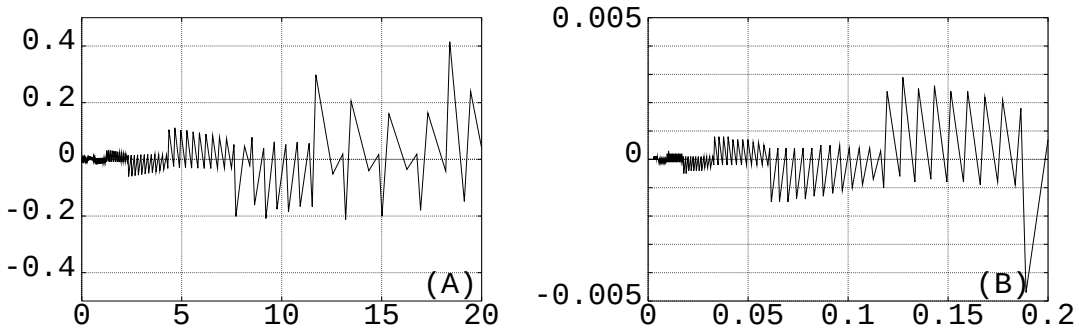

Fig. 4. Self-similarity of dynamic self-adaptation: Though (B) is a a strongly enlarged part of (A), the qualitative behavior in both is the same. During each 'run' of nearly 20 iterations the system oscillates between a momentary η/ζ (steeper) and η until the center of the orbit coincides with one of the lines $x_2 = \pm\lambda^{-1}\, x_1$. Then η is reduced by $1/\zeta$ and a new run starts.

zero – though quite slowly.

C Dynamic self-adaptation with normalization

Concentrating on the worst possible case where λ is extremely large we now get that *at first* the 2-coordinate is reduced since in general $|x_2(t)| < u_t$, and η_t is adapted accordingly, until we reach the 1-axis. During the *second* stage, we get a phenomenon that we have already hinted at and that is typical for all dimensions exceeding one: The orbit oscillates in a narrow valley. As can be seen in Figure 4, the oscillations occur in a 'cone' bounded by $x_2 = \pm\lambda^{-1}\, x_1$. As the system reaches one of the boundaries, the learning rate is reduced by $1/\zeta$. We remind the reader that on the boundaries the angle between the gradient and the x_1-axis is $\pm\pi/4$. Inside the cone the gradient has an angle *less than* $\pi/4$ with the x_1-axis and the procedure uses the learning rates η and η/ζ alternately where η is the momentary value. During a run, these values are kept constant until the orbit 'on the average' has reached one of the boundaries. Then the learning rate is reduced once again, the forward angle is now less than $\pi/4$, and the whole process is repeated. One notices that, during a run, x_1 and $\lambda\, x_2$ are of the same order of magnitude while the steep orbit ends approximately on the boundary so that, averaged over the run, $\eta/\zeta \approx \lambda^{-1}\, x_1, \Delta\, x_1 \approx -(\zeta/\eta)x_1$, and, this time on the average, we find

$$x_{t,1} \approx x_{0,1}\, e^{-\frac{\zeta\, t}{\lambda}} \tag{26}$$

Since $x_{t,2} \approx \lambda^{-1}\, x_{t,1}$ the error in f can be estimated by $e^{-2\,\zeta\, t/\lambda}$ and *approaches zero* as $t \to \infty$.

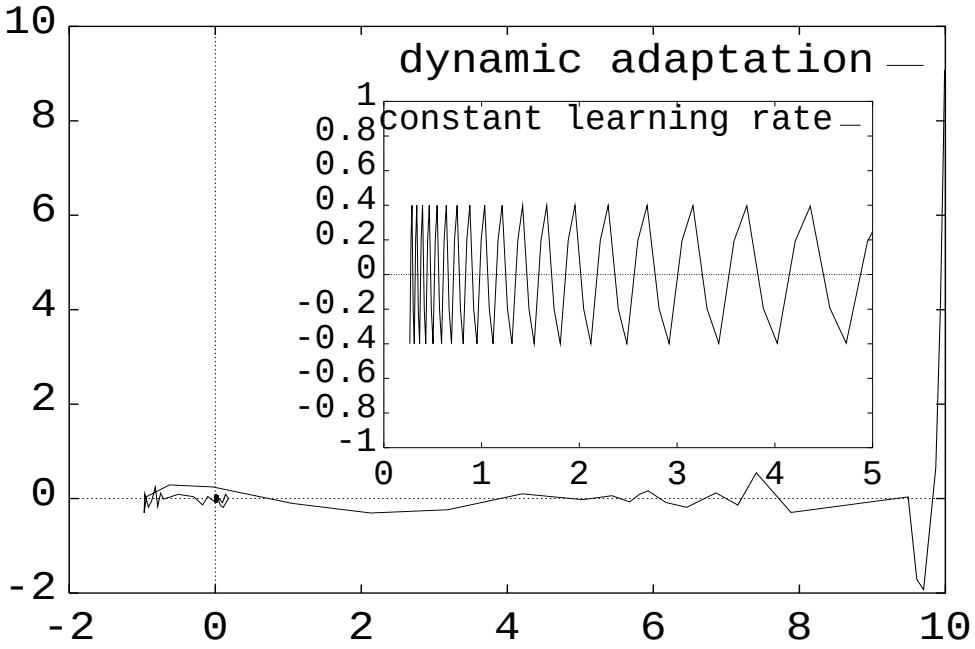

Fig. 5. Performance of dynamic self-adaptation of both the learning rate η and the momentum α for the 'cigar' $f(\vec{x}) = \frac{1}{2}x_1^2 + \frac{1}{2}(100)\,x_2^2$ with $\zeta = 1.839$. Note the fast and smooth convergence towards the origin which is to be contrasted with the oscillations in Figure 3 and the inset, that shows the behavior for *constant* α and η. In the figure, 100 iterations are shown altogether with a final accuracy of 0.09 whereas the inset exhibits 100 iterations and leads to a final accuracy of 8.04, which is not very accurate yet.

D Dynamic self-adaptation without normalization

For a quadratic cost function, dynamic self-adaptation without normalization is slightly faster than that with normalization. The reason is that one also takes into account the derivative which provides us with information concerning the distance to the origin, i.e., the minimum. Convergence itself is treated in a similar way to the previous case (C) with normalization. In general, however, when the cost function landscape is not quadratic, the algorithm with normalization is to be preferred as it is more robust.

3.3 Momentum

Having treated the four canonical cases A – D we turn to the *momentum* α as defined in Eq. (4). This equation makes explicit that the addition of momentum should slow down the oscillations and can be interpreted as an increase of 'inertia'. Figure 5 shows that as compared to Figures 3 and 4 the oscillations have been reduced drastically in case C, i.e., with normalization and adaptation of both η and α. The inset reveals that case A, with *constant* $\eta = 0.5$ and

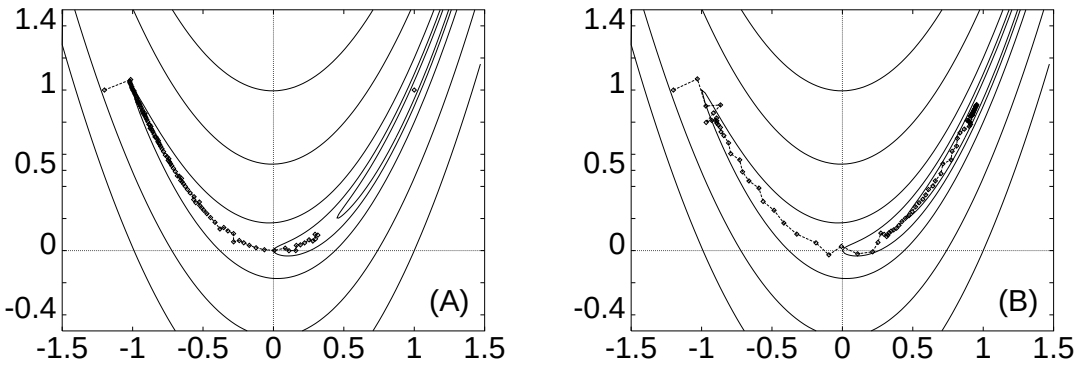

Fig. 6. Finding the minimum of the Rosenbrock function as defined by Eq. (27). It has a unique minimum at (1,1), at the base of the narrow banana-shaped valley (upper right-hand corner). For ease of comparison with other optimization procedures, we have followed Gill et al. (1981) in plotting the function. (**A**) "Bold driver" has $\alpha = 0.2$ fixed and gets stuck; the initial value of η was 0.001. (**B**) Starting with $\eta = 0.1$ and $\alpha = 0.1$ while $\zeta = 1.839$, dynamic self-adaptation of *both* parameters leads to a steady and smooth convergence to the minimum. In both cases only the first 100 iterations have been plotted. As shown in Figure 7, the rate of convergence is exponential.

$\alpha = 0.5$, the oscillations by no means have been extinguished. In short, here the momentum does not the work as expected and we are left with a nonzero final error.

Though the effect of momentum in Figure 5 looks quite spectacular, a closer examination reveals that once the procedure has nearly reached the origin (the minimum) inertia takes over and the 'cigar' $f(\vec{x}) = \frac{1}{2}\,x_1^2 + 50\,x_2^2$ sticks to a final error of about 0.2 due to a nonzero x_2. Without normalization but with adaptation of α and η convergence is fast; e.g., after $t = 60$ steps $f(\vec{x}_t)$ is less than 0.0001.

It may be interesting to compare the "bold driver" (Battiti 1989), which keeps α constant, with dynamic self-adaptation of both η and α. To this end we have studied the Rosenbrock function (Rosenbrock 1960)

$$f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2\,. \tag{27}$$

It has a unique minimum at (1,1), at the base of a narrow, banana-shaped, valley; see Figure 6. An extensive discussion of the performance of various optimization procedures when applied to this function has been given by Gill et al. (1981). In the present case, the performance of "bold driver" (see Figure 6a) is extremely bad: It effectively gets stuck. This already happens for fairly small values of α. On the other hand, dynamic self-adaptation of both η and α gives an excellent performance, as is shown in Figure 6b. In Figure 7 we exhibit an exponential decrease of the remaining error as the number of iterations

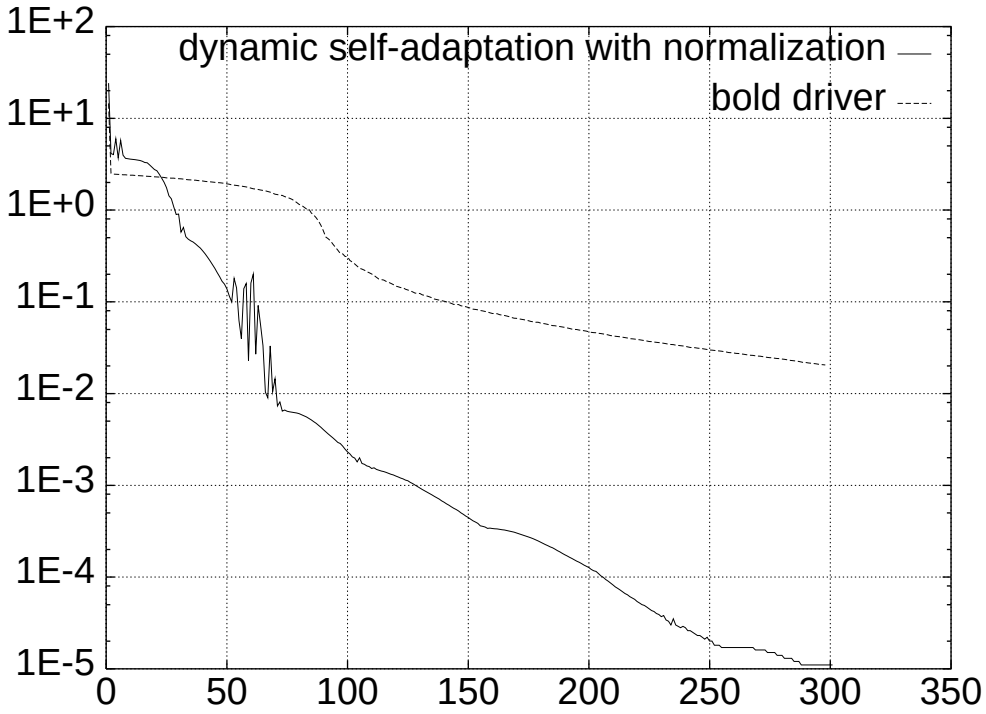

Fig. 7. Speed of convergence in case of Rosenbrock's function as shown in Fig. 6 for the "bold driver" procedure and dynamic self-adaptation. The difference between the actual value and the minimum (= 0) at (1,1) has been plotted as a function of the number of iterations. The vertical scale is a logarithmic one so that the convergence rate is exponential for dynamic self-adaptation (solid line) whereas the "bold driver" effectively gets stuck (horizontal dashed line).

increases.

3.4 n-dimensional case

In the n-dimensional case we want to minimize $f(\mathbf{x})$ as given by

$$f(\mathbf{x}) = \frac{1}{2}\,\lambda_1\,x_1^2 + \frac{1}{2}\,\lambda_2\,x_2^2 + \cdots + \frac{1}{2}\,\lambda_n\,x_n^2 \quad . \tag{28}$$

with $0 < \lambda_i \leq \lambda_j$ so for $1 \leq i < j \leq n$. Along each of the coordinate (= principal) axes and in case all λ_i are equal, we are back at the one-dimensional case. The worst possible case has $\lambda_1 \ll \lambda_2 \ll \cdots \ll \lambda_n$ and a starting point far away from the coordinate axes, e.g., along the vector $(1, 1, \ldots, 1)$. In analogy to the two-dimensional case, the procedure first converges along the x_n-axis towards the hyperplane spanned by the remaining eigendirections until $\lambda_n\,x_n \approx \lambda_{n-1}\,x_{n-1}$ and the direction $n-1$ takes over. Along the $(n-1)$-axis the system then proceeds accordingly to Eq. (23) with $\lambda := \lambda_n/\lambda_{n-1}$ until $\lambda_{n-1}\,x_{n-1} \approx \lambda_{n-2}\,x_{n-2}$. And so on until finally x_1 is reduced at an extremely

Table 2
Number of cycles for several encoders with given maximal error $\tau = 0.1$.

learning procedure	Encoder Problem				
	4-2-4	8-3-8	16-4-16	32-5-32	64-6-64
constant parameter	80	400	2 000	3 800	18 000
η self-adaptation	59	200	750	2 500	4 230
η & α self-adaptation	36	58	100	210	680

Table 3
Number of cycles for several encoders with given maximal error $\tau = 0.001$.

learning procedure	Encoder Problem				
	4-2-4	8-3-8	16-4-16	32-5-32	64-6-64
constant parameter	348 600	2 175 000	9 366 000	>20 000 000	∞
η self-adaptation	290	4 200	10 500	24 700	41 000
η & α self-adaptation	110	210	560	1 100	2 900

slow rate given by Eq. (23) with $\lambda := \lambda_n/\lambda_1$. The *final* convergence dominates the total convergence time. Thus the n-dimensional case can be reduced to the two-dimensional one.

4 Performance

Encoders constitute well-known test cases for any learning procedure. In this section, we analyze the performance of dynamic self-adaptation in teaching neural networks to solve several encoding and decoding tasks. The parity problem will be considered as well. In so doing we will compare our procedure with both standard backpropagation and with other approaches which we have discussed in the introductory section.

The encoder problem is a classical task to test performance; see, for instance, Rumelhart *et al.* (1986, pp. 335-339). An encoder has three layers, viz., an input layer, a hidden layer, and an output layer. There are feedforward connections from the input to the hidden layer and from the hidden layer to the output units. The task is to map the activity of the input units onto a prescribed activity of the output ones. To make the job more difficult the hidden layer usually contains less units than the input and the output layer.

Tables 2 and 3 present the number of iteration steps (i.e., epochs) needed for several encoders with a given maximal error. The learning procedure has been terminated once the errors for all patterns were lower than the prescribed

Table 4
Comparison of dynamic self-adaptation (DS) and the results obtained by Kramer and Sangiovanni-Vincentelli (1989). These authors used standard backpropagation (BP), line search (LS), and the conjugated gradient method introduced by Polak and Ribiere (PR). To give a correct comparison, the number of epochs required must be multiplied by an additional computation time (CT) factor, that takes into account the extra internal optimization effort as indicated by Kramer and Sangiovanni-Vincentelli. The third column DS-η, α shows that inclusion of dynamic self-adaptation of the momentum leads to a considerable speedup as compared to dynamic self-adaptation DS-η without momentum.

Encoder	maximum error	**DS-η,α**	PR	**DS-η**	LS	BP
10-5-10	0.447	**32**	64	**82**	109	197
10-5-10	0.374	**35**	71	**99**	142	300
10-5-10	0.037	**44**	105	**177**	431	3286
4-2-4	0.447	**27**	37	**49**	57	180
8-3-3	0.447	**43**	68	**154**	195	595
16-4-16	0.447	**77**	121	**331**	573	990
32-5-32	0.447	**111**	209	**645**	1379	1826
64-6-64	0.447	**252**	406	**2595**	4187	> 10000
Parity	maximum error	**DS-η,α**	PR	**DS-η**	LS	BP
2-2-1	0.447	**62**	73	**114**	95	686
4-4-1	0.447	**282**	352	**2112**	2052	47915
CT factor		$1.2-2$	$5-9$	$1.2-2$	$5-9$	1

maximal error. The first line of the tables represents standard backpropagation with optimal learning rate. The second line presents the results for dynamic self-adaptation of the learning rate and the number of epochs in the third line has been obtained by including self-adaptation of the *momentum* as well. As we have already pointed out in Section 2 the adaptation of η and α has been performed alternately. It is plain that dynamic self-adaptation of both parameters leads to a drastic improvement of the convergence rate. As compared to the original backpropagation procedure, an improvement of the epoch numbers by three orders of magnitude is obtained easily. From our point of view, however, the main advantage of our procedure is that it adapts itself *locally* to the cost function landscape. Experiments to optimize the parameters are not needed.

The results of the other procedures which we have reviewed in the introductory section have been presented in Tables 4 and 5. There one also finds results for the parity and the decoder problem. Parity concerns the question whether the

Table 5
Comparison of results obtained by using dynamic self-adaptation of the learning rate and the momentum (DS-η,α) and using the individual learning rates introduced by Jacobs (1988) and Silva and Almeida (1990).

Task	Maximum Total error	α	**DS-η,α**	Jacobs	Almeida
10-5-10 Encoder	0.49	0.0	**56**		73
10-5-10 Encoder	0.49	0.8	**26**		49
10-5-10 Encoder	0.10	0.0	**93**		117
10-5-10 Encoder	0.10	0.8	**36**		79
2-2-1 Parity	0.04	0.0	**236**	250	
2-2-1 Parity	0.04	0.5	**64**	250	
3-1-8-8 Decoder	0.49	0.0	**1351**		1828
3-1-8-8 Decoder	0.49	0.9	**315**		421
3-1-8-8 Decoder	0.64	0.0	**2350**	2154	
3-1-8-8 Decoder	0.64	0.9	**503**	872	

number of bits in the input layer is even or odd. A decoder (Jacobs 1988) has n binary input units, one hidden unit, next, 2^n hidden units and, finally, 2^n output units which represents the first 2^n positive integers. There are feedforward connections only. The task of the network is to map a binary encoded number represented by a configuration of the input layer onto the corresponding unit in the output layer that represents it explicitly. The alternative procedures, e.g., line search and conjugated gradient, which we have discussed in section 1 and whose results we have presented in Table 4 perform an internal optimization of their parameters during each iteration step. We have taken the extra optimization effort into account explicitly by the computation time (CT) factor in the bottom line of Table 4. As to dynamic self-adaptation, the CT factor equals 2 but in practice it turns out that one need not change η and α every iteration step – certainly not asymptotically. It often suffices to adapt the parameters, say, each tenth step so that one effectively ends up with a CT factor of the order 1.2.

5 Conclusion

We have seen that dynamic self-adaptation leads to a considerable improvement of the convergence rate and the accuracy – up to several orders of magnitude as compared to other procedures. Characteristic properties of dynamic

self-adaptation are the following:

- The procedure is simple and its implementation is straightforward.
- The procedure adapts the learning parameters to the local configuration of the cost function landscape.
- Time consuming internal optimizations and *a priori* knowledge are not needed.
- Dynamic self-adaptation attains a prescribed accuracy, whatever its precision, in a finite number of steps.

We have presented a convergence proof, estimated the convergence rate, and indicated the generic convergence behavior in one dimension. Moreover, we have presented a worst case analysis for $n \geq 2$ dimensions. For a positive-definite quadratic form, the total convergence time is dominated by the ration $\lambda = \lambda_n/\lambda_1$ where λ_n is the largest and λ_1 the smallest eigenvalue as the accuracy is determined by $\exp(-2\,\zeta\,t/\lambda)$.

There still are several open problems. First and foremost, dynamic self-adaptation does not solve the problem of getting stuck in *local* minima. A way-out may be provided by a combination of genetic algorithms and dynamic self-adaptation. In addition, one can apply dynamic self-adaptation to a whole set of parameters. What is the optimal strategy in doing so?

Acknowledgement

It is a pleasure to thank Albrecht Biedl, Bernhard Sulzer, and Michael Scholz for helpful discussions. We gratefully acknowledge the hospitality of the TU München and the TU Berlin, in particular, Stefan Jähnichen, for stays that made completion of the present manuscript possible.

References

Battiti, R. (1989). Accelerated Backpropagation Learning: Two Optimization Methods. *Complex Systems*, **3**:331–342.

Becker, S. and le Cun, Y. (1989). Improving the convergence of back-propagation learning with second order methods. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pp. 29–37. Morgan Kaufmann Publishers, San Mateo, CA.

Gill, P.E., Murray, W., & Wright, M.H. (1981). *Practical optimization.* London: Academic Press.

Hush, D. R. and Salas, J. M. (1988) Improving the learning rate of backpropagation with the gradient reuse algorithm. In *IEEE International Conference on Neural Networks*, pp. 441–447. The Institute of Electrical and Electronic Engineers, San Diego, CA.

Jacobs, R. A. (1988) Increased rates of convergence through learning rate adaption. *Neural Networks*, **1**:295–307.

Kramer, A. H. and Sangiovanni-Vincentelli, A. (1989) Efficient parallel learning algorithms for neural networks. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, pp. 40–48. Morgan Kaufmann Publishers, San Mateo, CA.

Luenberger, D. G. (1984) *Linear and Nonlinear Programming.* Addison-Wesley Company, Menlo Park, CA.

Nelder, J.A. & Mead, R., A simplex method for function minimization. Computer J. **7** (1965) 308-313.

Van Ooyen and Nienhuis (1992) Improving the Convergence of the Back-Propagation Algorithm *Neural Networks*, **5**:465–471.

Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. (1987) *Numerical Recipes.* Cambridge University Press.

Rigler A., Irvine, J., and Vodel T. (1991), Rescaling of Variables in Back Propagation Learning. *Neural Networks*, **4**:225–229.

Rosenbrock, H.H., An automatic method for finding the greatest or least value of a function. Computer J. **3** (1960) 175-184.

Rumelhart *et al.* , editors (1986) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2, Psychological and Biological Models. MIT Press/Bradford Books, 1986.

Salomon, R. (1990) Beschleunigtes Lernen durch adaptive Regelung der Lernrate bei Back-propagation in feed-forward Netzen. In Georg Dorffner, editor, *Konnektionismus in Artifical Intelligence und Kognitionsforschung. 6. Österreichische Artificial-Intelligence-Tagung (KONNAI)*, pp. 173–178. Springer-Verlag.

Salomon, R. (1991) *Verbesserung konnektionistischer Lernverfahren, die nach der Gradientenmethode arbeiten.* PhD thesis, Technische Universität Berlin, October 1991.

Silva, F. M., and Almeida, L. B. (1990) Speeding up backpropagation. In *Proceeding of NSMS - International Symposium on Neural Networks for Sensory and Motor Systems.* Elsevier Science Publishers, Amsterdam.

Appendix

As its name says, dynamic self-adaptation adapts itself steadily to the cost-function landscape. In one dimension we can, and will, follow this process in detail. The argument has been split into three theorems and we start with a

simple but important lemma.

Lemma 1 $f(x_t - \eta_t \operatorname{sgn}(x_t)\,\zeta) \leq f(x_t - \eta_t \operatorname{sgn}(x_t)/\zeta)$ is equivalent to $|x_t| \geq u_t = \eta_t \frac{\zeta^2+1}{2\zeta}$, i.e., if x_t is outside the reversal interval, then the larger step $x_t - \eta_t \zeta$ ends nearer to the minimum than the smaller one $x_t - \eta_t/\zeta$. Inside the reversal interval, it is the other way around.

Proof. As mentioned before, we have $\zeta > 1$, take $x_t > 0$ and $f(x) = \frac{1}{2}\lambda x^2$. The proof of the lemma is simple,

$$\begin{aligned} f(x_t - \eta_t \zeta \operatorname{sgn}(x_t)) &\leq f(x_t - \eta_t \operatorname{sgn}(x_t)/\zeta) \\ \frac{1}{2}\lambda(x_t^2 - 2\,x_t\,\zeta\,\eta_t + \zeta^2\,\eta_t^2) &\leq \frac{1}{2}\lambda(x_t^2 - 2\,x_t\,\frac{1}{\zeta}\,\eta_t + \frac{1}{\zeta^2}\,\eta_t^2) \\ 2\,x_t\,\left(\zeta - \frac{1}{\zeta}\right) &\geq \eta_t\,\left(\zeta^2 - \frac{1}{\zeta^2}\right) \end{aligned}$$

and thus

$$x_t \geq \frac{\eta_t}{2}\,\frac{\zeta^2 - \frac{1}{\zeta^2}}{\zeta - \frac{1}{\zeta}} = \frac{\eta_t}{2}\,\frac{\zeta^4 - 1}{\zeta(\zeta^2 - 1)} = \eta_t\,\frac{\zeta^2 + 1}{2\,\zeta} \tag{29}$$

as announced. □

With the help of the above lemma it is easy to prove the following two theorems. The first states that, if at a certain time t we are outside the reversal interval, then the learning rate η *increases* and we can find a time $\tau > t$ such that x_τ is inside the corresponding reversal interval U_τ. The second theorem says that, if at time t we are in the reversal interval, then we stay so forever and η_t decreases monotonically. Consequently, the convergence rate is exponential.

Theorem 2 If $|x_t| \geq u_t$, then $f(x_t - \eta_t \operatorname{sgn}(x_t)\,\zeta) \leq f(x_t - \eta_t \operatorname{sgn}(x_t)/\zeta)$, the learning rate $\eta_{t+1} = \eta_t\,\zeta$ *increases*, and there exist an iteration step $\tau > t$ so that $|x_\tau| < u_\tau$.

Proof. Due to the lemma we only have to show that we can find a $\tau > t$ such that $|x_\tau| < u_\tau$. Since $x_{t+1} = x_t - \eta_{t+1} \operatorname{sgn}(x_t)$ we have

$$|x_t| \leq \max(|x_{t-1}|, \eta_t) \leq \cdots \leq \max_{t' \leq t}(|x_0|, \eta_{t'}) \tag{30}$$

Furthermore, by its definition $u_t = \eta_t \frac{\zeta^2+1}{2\zeta}$ with $\zeta > 1$ so we must have $u_t > \eta_t$. Since $\eta_t = \zeta^t\,\eta_0$ increases without bound, if there were no τ with $|x_\tau| < u_\tau$, we

would find for t large enough $|x_t| \leq \eta_t$ whereas on the other hand $|x_t| \geq u_t > \eta_t$. This is a contradiction. Hence we can find a τ such that $|x_\tau| < u_\tau$. $\square$

One can estimate the above τ by noting that at worst the contradiction arises when $\eta_t = \zeta^t \eta_0$ has the same order of magnitude as $|x_0|$.

Theorem 3 If $|x_t| < u_t$, then $f(x_t - \eta_t \operatorname{sgn}(x_t)\zeta) > f(x_t - \eta_t \operatorname{sgn}(x_t)/\zeta)$, and the learning rate $\eta_{t+1} = \eta_t/\zeta$ *decreases*. Furthermore, $|x_{t+1}| < u_{t+1}$ is guaranteed for every $\zeta \leq \zeta_c \approx 1.839$.

Proof. We only have to verify the existence of a critical ζ_c. To this end we require

$$|x_{t+1}| = \left| x_t - \frac{1}{\zeta}\eta_t \right| < \eta_{t+1}\,\frac{\zeta^2+1}{2\,\zeta} \quad , \tag{31}$$

an inequality which is of the form $|x-a| < b$ and thus can be expressed as $a - b < x < a + b$. The lower bound leads to

$$x_t > -\eta_t\,\frac{\zeta^2 - 2\,\zeta + 1}{2\,\zeta^2} = -\eta_t\,\frac{(\zeta-1)^2}{2\,\zeta^2} \tag{32}$$

and is true for any $\zeta > 1$. The upper bound to be shown reads

$$x_t < \frac{\eta_t\,(\zeta^2+1)}{2\,\zeta^2} + \frac{2\,\eta_t\,\zeta}{2\,\zeta^2} = \frac{\eta_t\,(\zeta+1)^2}{2\,\zeta^2} \tag{33}$$

By assumption we have $|x_t| < u_t = \eta_t\,\frac{\zeta^2+1}{2\,\zeta}$ and as a consequence we obtain

$$x_t < \frac{\eta_t\,(\zeta^2+1)}{2\,\zeta} < \frac{\eta_t\,(\zeta+1)^2}{2\,\zeta^2} \quad . \tag{34}$$

The first inequality is a direct consequence of $x_t < u_t$. We only have to verify the second inequality, viz.

$$\zeta^2 + 1 < \frac{(\zeta+1)^2}{\zeta} \Rightarrow \zeta^3 - \zeta^2 - \zeta - 1 < 0 \tag{35}$$

so that by the Cardano formulae

$$\zeta < \zeta_c = \frac{1}{3}\left(\sqrt[3]{19+\sqrt{297}} + \frac{4}{\sqrt[3]{19+\sqrt{297}}} + 1\right) \approx 1.8392867. \qquad \square \quad (36)$$

It turns out that ζ_c is optimal. That is, for $\zeta \leq \zeta_c$ the system stays in the reversal interval forever, provided it is already there, whereas for $\zeta > \zeta_c$ it may happen that Theorem 2 has to be applied at least once even though we were already in the reversal interval. (This need not be, as Table 1 shows.)

In passing we note that, if we would have worked with $\zeta_1 > 1$ and $\zeta_2 > 1$ instead of $\zeta > 1$ and $1/\zeta$, then a steady decrease of the learning rate is guaranteed, if

$$\zeta_1(\zeta_2^2 - \zeta_2) - \zeta_2 \leq 1. \qquad (37)$$

In case of equality and $\zeta_1 = 1/\zeta_2$ we regain $\zeta_c = 1.839$.

Convergence for all $\zeta > 1$

One may wonder what happens, if $\zeta > \zeta_c$. A typical example is presented in Table 6 for the case $f(x) = \frac{1}{2}x^2$, $x_0 = 54$, $\eta_0 = 1$, and $\zeta = 10 \gg \zeta_c$. It is evident that the convergence in Table 6 is not as good as the one in Table 1 where $\zeta = 2$. Furthermore, we see that typically the steps are performed according to Theorems 2 and 3 alternately. Finally, during the steps 1 to 3 the learning rate η is reduced *twice*. The process is repeated after ten ($= \zeta$) iteration steps. This behavior is formalized in Theorem 5. As a preparation we first prove a lemma.

Lemma 4 If $|x_t| < u_t = \eta_t \frac{\zeta^2+1}{2\zeta}$ and all further steps are of the form η_t/ζ^k with $k \geq 0$, then we have $\left|x_t - \eta_t \operatorname{sgn}(x_t)/\zeta^k\right| < u_t$ and $|x_{t'}| < u_t$ for all $t' > t$.

The proof of is easy. Since $|x_t| < u_t$ and $k \geq 0$ we obtain the inequality $\left|x_t - \eta_t \operatorname{sgn}(x_t)/\zeta^k\right| < \max(|x_t|, \eta_t/\zeta^k)$. The final part is proven by induction on t'.

Theorem 5 If $|x_t| < u_t = \eta_t \frac{\zeta^2+1}{2\zeta}$, then for all $t' \geq t$ we have $|x_{t'}| < u_t$. Furthermore, after at most n steps with n even and $\zeta \leq n < \zeta + 2$, we obtain the inequality $|x_{t+n}| < u_t/\zeta$.

Proof. The first part of the Theorem 5 is a consequence of the algorithm and Lemma 4, with alternatingly $k = 0$ and $k = 1$. To prove the second

Table 6
An example of convergence with $\zeta = 10 \gg \zeta_c$.

t	x_t	η_t	u_t	$x_t - \eta_t \frac{1}{\zeta}\,\mathrm{sgn}(x_t)$	$x_t - \eta_t\,\zeta\,\mathrm{sgn}(x_t)$	η_{t+1}/η_t	x_{t+1}	η_{t+1}
0	15	1	5.05	14.9	5	ζ	5	10
1	5	10	50.5	4	-95	$1/\zeta$	4	1
2	4	1	5.05	3.9	-6	$1/\zeta$	3.9	0.1
3	3.9	0.1	0.505	3.89	2.96	ζ	2.9	1
4	2.9	1	5.05	2.8	-7.1	$1/\zeta$	2.8	0.1
5	2.8	0.1	0.505	2.79	1.8	ζ	1.8	1
6	1.8	1	5.05	1.7	-8.2	$1/\zeta$	1.7	0.1
7	1.7	0.1	0.505	1.69	0.7	ζ	0.7	1
8	0.7	1	5.05	0.6	-9.3	$1/\zeta$	0.6	0.1
9	0.6	0.1	0.505	0.59	0.4	ζ	-0.4	1
10	-0.4	1	5.05	-0.3	9.6	$1/\zeta$	-0.3	0.1
11	-0.3	0.1	0.505	-0.29	0.7	$1/\zeta$	-0.29	0.01
12	-0.29	0.01	0.0505	0.289	-0.19	ζ	-0.19	0.1
13	-0.19	0.1	0.505	-0.18	0.81	$1/\zeta$	-0.18	0.01

part we start by noting that, if $|x_t| < u_t$, then a step is performed according to Theorem 3 and consequently $\eta_{t+1} = \eta_t/\zeta$. If in addition $|x_{t+1}| < u_t/\zeta$, then a second step is performed according to Theorem 3 and we are done. If on the other hand $|x_{t+1}| \geq u_t/\zeta$, then we have to perform a step according to Theorem 2 and $\eta_{t+2} = \eta_t$. Let us assume for the moment that ζ is a positive, even integer. The algorithm alternately performs a step of size η_t or one of size η_t/ζ. According to Theorem 5 we aim at a precision of at least $u_t/\zeta = \eta_t/2 + \eta_t/(2\,\zeta^2)$, i.e., to a rather good accuracy $\eta_t/2$. According to the gradient-descent method proper a step size η_t gives an ultimate accuracy $\eta_t/2$ and "ultimately" means after at most $(u_t - \eta_t/2)/\eta_t < \zeta/2$ steps. Adding the other steps of size η_t/ζ which we have neglected, we arrive at the upper bound ζ for attaining u_t/ζ. The rest is plain sailing. □