Technical Report IDSIA-09-98

Online Local Gain Adaptation for Multi-Layer Perceptrons

Nicol N. Schraudolph
nic@idsia.ch

IDSIA, Corso Elvezia 36
6900 Lugano, Switzerland
http://www.idsia.ch/

March 27, 1998
revised May 15, 1998

Abstract

We introduce a new method for adapting the step size of each individual weight in a multi-layer perceptron trained by stochastic gradient descent. Our technique derives from the K1 algorithm for linear systems (Sutton, 1992b), which in turn is based on a diagonalized Kalman Filter. We expand upon Sutton's work in two regards: K1 is a) extended to multi-layer perceptrons, and b) made more efficient by linearizing an exponentiation operation. The resulting ELK1 (extended, linearized K1) algorithm is computationally little more expensive than alternative proposals (Zimmermann, 1994; Almeida et al., 1997, 1998), and does not require an arbitrary smoothing parameter. In our benchmark experiments, ELK1 consistently outperforms these alternatives, as well as stochastic gradient descent with momentum, even when the number of floating-point operations required per weight update is taken into account. Unlike the method of Almeida et al. (1997, 1998), ELK1 does not require statistical independence between successive training patterns, and handles large initial learning rates well.

1 Introduction

For neural network learning algorithms based on simple gradient descent, an appropriate choice of step size for each weight is crucial to achieve efficient learning (Schraudolph & Sejnowski, 1996). Most of the existing algorithms for controlling the step size in neural networks, however, adapt only the global learning rate (LeCun et al., 1993; Murata et al., 1997), can be used only in batch training (Jacobs, 1988; Silva & Almeida, 1990), or both (Lapedes & Farber, 1986; Vogl et al., 1988; Battiti, 1989; Yu et al., 1993).

Given the inherent efficiency of stochastic gradient descent, it would be desirable to have comparable online methods to adapt local step sizes. The little-known *vario-η* approach (Zimmermann, 1994, page 48) has been found to work well in practice, but is a gradient normalization rather than gain adaptation technique. A local gain adaptation method for online optimization has recently been proposed by Almeida et al. (1997, 1998) but has not yet been widely tested.

Sutton (1992a, 1992b) has developed three algorithms — IDBD, K1 and K2 — for online local gain adaptation in linear systems. Of these we have found K1 to be the most robust and effective; we are in agreement with Sutton on this (personal communication). Here we extend K1 to multi-layer perceptrons, and improve its efficiency by eliminating a costly exponentiation operation. The resulting ELK1 (extended, linearized K1) algorithm is then implemented and compared to the methods proposed by Zimmermann (1994) and Almeida et al. (1998).

2 Derivation of the Algorithm

Consider a neural network node with input vector $\vec{x}$ and activation[1]

$$z = f(y), \quad y = \vec{w}^T \vec{x}, \tag{1}$$

where f is a nonlinear (typically sigmoid) function. The weight vector $\vec{w}$ is adapted online via

$$\vec{w} \leftarrow \vec{w} + \delta \vec{k}, \quad \delta = -\frac{\partial E}{\partial z}, \tag{2}$$

where E is the objective function to be minimized, and $\vec{k}$ the node's *gain* vector. Regarding this system as an *extended Kalman Filter* (Singhal & Wu, 1989; Matthews, 1990; Puskorius & Feldkamp, 1991; Shah et al., 1992; Williams, 1992; Plumer, 1995) we obtain

$$\vec{k} = \frac{P\,\vec{x}\,f'(y)}{r + \vec{x}^T P\,\vec{x}\,f'(y)^2}, \tag{3}$$

where r is an estimate of noise variance in the target output, and P a recursively computed estimate of the error covariance matrix of $\vec{w}$. K1 (Sutton, 1992b) uses the diagonal approximation $P \equiv \mathrm{diag}(e^{\vec{q}})$ whose parameter vector $\vec{q}$ is adapted online. The exponentiation ensures that the components of P remain positive

[1] In the interest of clarity we suppress node and pattern indices; all variables, gradients, *etc.* are implied to be instantaneous values with respect to a given node and pattern. Where important, the order of updates for pattern-dependent variables is stated in the text.

and scale geometrically. $\vec{q}$ is optimized by stochastic gradient descent in E with constant step size given by the *meta*-parameter μ:

$$\begin{aligned}
\vec{q} \;&\leftarrow\; \vec{q} \,-\, \mu \frac{\partial E}{\partial \vec{q}} \\
&=\; \vec{q} \,-\, \mu \frac{\partial E}{\partial z}\frac{\partial z}{\partial y}\left(\frac{\partial y}{\partial \vec{w}^T}\frac{\partial \vec{w}}{\partial \vec{q}^T}\right)^T \\
&=\; \vec{q} \,+\, \mu\,\delta\, f'(y)\,\frac{\partial \vec{w}^T}{\partial \vec{q}}\,\vec{x} \\
&\approx\; \vec{q} \,+\, \mu\,\delta\, f'(y)\,(\vec{h}\cdot\vec{x})\,,
\end{aligned} \tag{4}$$

where $\vec{h}$ is a diagonal approximation of $\partial\vec{w}^T/\partial\vec{q}$, expressing the notion that each component of $\vec{q}$ primarily affects the corresponding component of $\vec{w}$.

K1 requires a computationally expensive exponentiation for each weight and training pattern. We have previously described a very fast but machine-dependent approximation of the exponential function (Schraudolph, 1998); here we prefer the linearization $e^u \approx 1+u$, valid for small $|u|$. Thus instead of $\vec{q}$ one can multiplicatively update a parameter vector $\vec{p} \equiv e^{\vec{q}}$:

$$\begin{aligned}
\vec{p} \;&\leftarrow\; e^{\vec{q}\,+\,\mu\delta f'(y)(\vec{h}\cdot\vec{x})} \\
&=\; \vec{p}\cdot e^{\mu\delta f'(y)(\vec{h}\cdot\vec{x})} \\
&\approx\; \vec{p}\cdot \max\left[\,\varrho,\; 1+\mu\,\delta\, f'(y)\,(\vec{h}\cdot\vec{x})\right],
\end{aligned} \tag{5}$$

with the multiplication factor bounded below by ϱ — which we generally set to 0.5 — as a safety measure. This linear approximation works well since μ is typically small. Note that (5) is invariant with respect to rescaling of $\vec{p}$; one can therefore set $r \equiv 1$ in (3) without loss of generality, yielding

$$\vec{k} \;=\; \frac{\vec{p}\cdot\vec{x}\, f'(y)}{1+\vec{x}^T(\vec{p}\cdot\vec{x})\, f'(y)^2}\,. \tag{6}$$

It remains to calculate the vector $\vec{h}$. We first require the auxiliary quantities

$$\mathrm{diag}\left(\frac{\partial \vec{k}^T}{\partial \vec{q}}\right) \;=\; \vec{k} \,-\, \vec{k}^2\cdot\vec{x}\, f'(y) \;=\; \vec{k}\cdot[\vec{1}-(\vec{k}\cdot\vec{x})\, f'(y)] \tag{7}$$

and

$$\frac{\partial \delta}{\partial \vec{q}} \;=\; \frac{\partial^2 E}{\partial \vec{q}\,\partial z} \;=\; -\frac{\partial}{\partial z}\left[\,\delta\, f'(y)\,\vec{h}\cdot\vec{x}\,\right] \;\approx\; -(\vec{h}\cdot\vec{x})\, f'(y)\,, \tag{8}$$

where unity curvature ($\partial^2 E/\partial z^2 \approx 1$) has been assumed in (8) for simplicity. Now $\vec{h}$ can be updated online in concert with $\vec{w}$ as follows:

$$\vec{h} \;\leftarrow\; \mathrm{diag}\left(\frac{\partial}{\partial \vec{q}}\left(\vec{w} \,+\, \delta\,\vec{k}\right)^T\right)$$

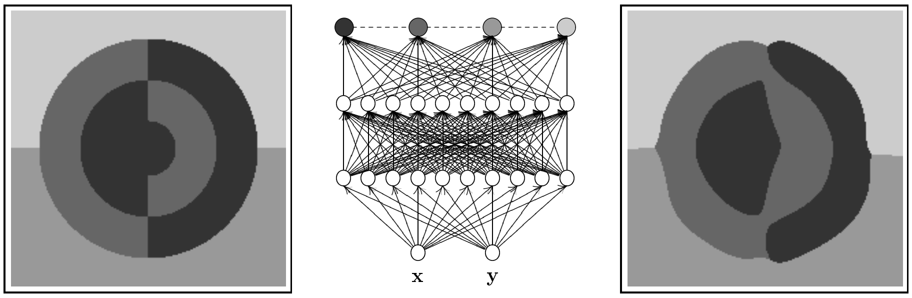

Figure 1: The four regions classification task (left), the neural network trained on it (center), and a typical solution found after 50 000 training patterns (right).

$$
\begin{aligned}
&\approx \quad \vec{h} \;+\; \delta\,\mathrm{diag}\left(\frac{\partial \vec{k}^T}{\partial \vec{q}}\right) \;+\; \frac{\partial \delta}{\partial \vec{q}} \cdot \vec{k} \\
&\approx \quad (\vec{h} + \delta\,\vec{k}) \cdot \left[1 - (\vec{k} \cdot \vec{x})\, f'(y)\right] \,. \qquad (9)
\end{aligned}
$$

Thus $\vec{h}$ integrates a history of recent weight changes $\delta\vec{k}$, a feature ELK1 has in common with other gain adaptation methods (Murata et al., 1997; Jacobs, 1988). In contrast to those methods, however, the time scale of integration is not constant, but varies from pattern to pattern, as determined by $1 - (\vec{k} \cdot \vec{x})\, f'(y)$. Note that each element of this factor is guaranteed to lie between zero and one; a positive-bounding operation as in (Sutton, 1992b) is therefore not necessary. Initially we set $\vec{h} \leftarrow 0$.

In order to make immediate use of the latest information available, the gain is adapted before the weights for each new pattern. The complete ELK1 (extended, linearized K1) update thus comprises the following steps (equations), in this order:

1. propagate activity forward (1),
2. backpropagate error to obtain δ,
3. update learning rates $\vec{p}$ (5),
4. calculate gain vectors $\vec{k}$ (6),
5. update the $\vec{w}$ (2) and $\vec{h}$ (9).

3 Benchmark Comparison

We evaluated ELK1 on the “four regions” classification task due to (Singhal & Wu, 1989), a well-established benchmark problem (Puskorius & Feldkamp, 1991; Shah et al., 1992; Yu et al., 1993; Plumer, 1995). A fully connected feedforward network with 2 hidden layers of 10 units each (tanh nonlinearity) is to classify two continuous inputs (range [-1,1]) as illustrated in Figure 1. We used "softmax" output units and a negated cross-entropy objective function E. We compared four online algorithms: gradient descent with momentum, vario-η (Zimmermann, 1994, page 48), ALAP — normalized local step size adaptation as proposed by Almeida et al. (1998), and ELK1.

The vario-η method normalizes the local gradient by its own standard deviation $\vec{\sigma}$, thus limiting the stochasticity of the weight update:

$$\vec{w} \;\leftarrow\; \vec{w} \;+\; \eta\,\delta\,f'(y)\,(\vec{x}/\vec{\sigma})\,, \tag{10}$$

where $\vec{\sigma}$ can be computed online from the two exponential traces

$$\vec{m} \;\leftarrow\; \alpha\,\vec{m} \;+\; (1-\alpha)\,\delta\,f'(y)\,\vec{x} \tag{11}$$

and

$$\vec{v} \;\leftarrow\; \alpha\,\vec{v} \;+\; (1-\alpha)\,[\,\delta\,f'(y)\,\vec{x}\,]^2\,. \tag{12}$$

In order to make vario-η more efficient in our online setting, we replaced the obvious formula $\vec{\sigma} = \sqrt{\vec{v} - \vec{m}^2}$ — which would require an expensive square root operation — by a single Newton-Raphson iteration that approximates it:

$$\vec{\sigma} \;\leftarrow\; \frac{1}{2}\left(\vec{\sigma} \;+\; \frac{\vec{v} - \vec{m}^2}{\vec{\sigma}}\right)\,. \tag{13}$$

The ALAP method was implemented with the weight update

$$\vec{w} \;\leftarrow\; \vec{w} \;+\; \delta\,f'(y)\,(\vec{p}\cdot\vec{x})\,, \tag{14}$$

where the local step size $\vec{p}$ was updated multiplicatively via

$$\vec{p} \;\leftarrow\; \vec{p}\left[1 \;+\; \mu\,\delta\,f'(y)\,(\vec{h}\cdot\vec{x})/\vec{v}\right], \tag{15}$$

with $\vec{v}$ computed by (12), and $\vec{h}$ storing the gradient with respect to the previous pattern — in other words, (15) is followed by $\vec{h} \leftarrow \delta\,f'(y)\,\vec{x}$.

Finally, ELK1 was implemented by equations (2), (5), (6), and (9), as described in Section 2. The momentum approach simply used (11) followed by the weight update $\vec{w} \leftarrow \vec{w} + \eta\,\vec{m}$. To provide stable starting conditions for all algorithms, we initialized all elements of $\vec{p}$ to η, of $\vec{h}$ and $\vec{m}$ to 0, and of $\vec{v}$ to 1.

Table 1 lists the complexity of the above algorithms in terms of how many numbers must be stored per weight, and how many floating-point operations (flops) must be performed per weight update[2]. Also shown are the values we used for the free parameters of each method; they were chosen to minimize E

[2]This figure does not take into account the computation required on a per-node basis, the cost of non-floating-point or non-arithmetic operations, the fact that errors need not be backpropagated into input units, the cost of obtaining a training pattern, *etc.*

Algorithm:	momentum	vario-η	ALAP	ELK1
storage/weight	2	4	3	3
flops/weight update	8	18	15	20
CPU ms/pattern*	7.3	8.6	8.0	9.8
Parameters:				
(initial) learning rate η	0.01	0.01	(0.1)	(0.1)
meta-learning rate μ	—	—	0.01	0.1
smoothing parameter α	0.95	0.95	0.95	—
Results (25 runs):				
$\bar{E}$ @ 50k mean $\pm$ st.d.	0.53 ± 0.06	0.52 ± 0.05	0.40 ± 0.07	0.25 ± 0.06
patterns min – max	0.43 – 0.63	0.43 – 0.60	0.31 – 0.54	0.20 – 0.45
k patterns mean $\pm$ st.d.	52.6 ± 8.0	$30.6 \pm 5.4^{\dagger}$	$26.4 \pm 6.3^{\ddagger}$	17.4 ± 5.0
to $\bar{E} \leq 0.5$ min – max	42.8 – 77.2	21.6 – >50	15.4 – >50	10.1 – 30.6

*Unoptimized octave 2.0.10 code under Linux 2.0.29 on Intel Pentium Pro @ 240 MHz.
† Excludes 4, ‡ excludes 2 runs that did not reach $\bar{E} \leq 0.5$ within 50 000 patterns.

Table 1: Comparison of four stochastic gradient descent methods as to computational complexity, free parameters, and performance on the four regions task.

after 50k training patterns. About equal, significant time was spent tuning each value; note that ELK1 is the only algorithm which does not require the arbitrary smoothing parameter α.

We trained each algorithm 25 times on a uniformly random sequence of 50k patterns (100k for momentum), starting from different sets of uniformly random initial weights (range $[-0.3, 0.3]$; biases initialized to zero). Figure 2 shows the average value of $\bar{E}$ — an exponential trace (smoothing parameter 0.999) of E — during training. Since each pattern was seen only once, E measures expected generalization ability. Table 1 lists the empirical mean, standard deviation, minimum and maximum (over the 25 runs) for two quantities of interest: the value of $\bar{E}$ at the end of training, and the number of patterns (in thousands) before $\bar{E}$ first fell below the (arbitrary) threshold of 0.5.

Figure 2 (left) shows that while all three accelerated methods convey a similar initial increase in convergence speed over gradient descent with momentum, there are clear differences in their later performance: vario-η slows down so much that after 50k patterns it is essentially no better than the momentum approach, while ALAP is better able to hold on to its gains. Both are, however, clearly outperformed by ELK1, whose error after 50k patterns is far lower.[3]

Such results, however, obscure the fact that the accelerated methods are

[3]The empirical standard error on all curves in Figure 2 at 50k patterns is ≤ 0.014.

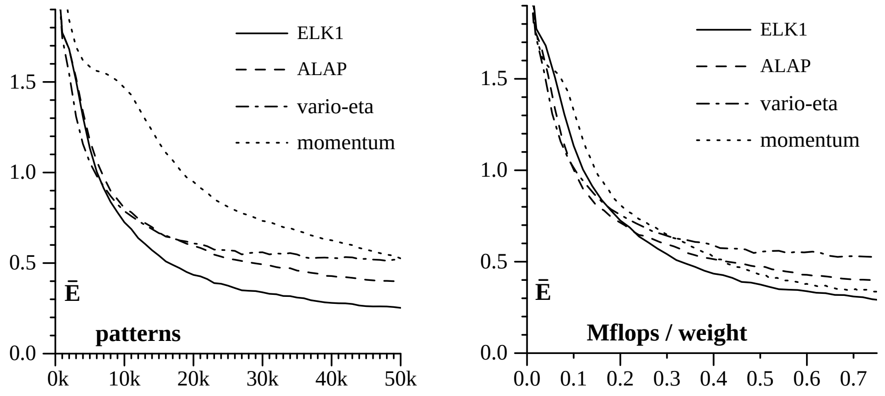

Figure 2: Evolution of the average smoothed objective $\bar{E}$ during online training of the four algorithms on the four regions task, *vs.* number of training patterns (left), and number of floating-point operations per weight (right), respectively.

computationally more expensive. In Figure 2 (right) we have therefore multiplied the number of patterns on the ordinate by the floating-point operations per weight update that each method requires, as listed in Table 1. Now the momentum approach looks far more compelling, and in fact even overtakes ALAP in late performance. ELK1 still has a significant[3] (though much reduced) lead.

The two graphs in Figure 2 measure algorithmic performance under the assumption that in relation to the remainder of the computation, the cost of updating the weights is either negligible (left) or dominant (right). In practice neither is likely to be the case, and the computational truth is bound to lie in the middle. As a subjective guide, Table 1 lists the CPU time (in milliseconds per pattern) consumed by each algorithm in our prototype implementation.

There are other differences between the above algorithms. For instance, ALAP is derived under the assumption that successive training patterns are statistically independent (Almeida et al., 1998) — a condition that may be hard to meet *e.g.* in *situated* applications, where the input is provided by a real-time environment over which the learner does not have complete control.

In order to explore the sensitivity of the present algorithms to dependencies between successive training patterns, we repeated the above experiment while sampling the input space according to the Sobol sequence (Press et al., 1992, pp. 311–314). This deterministic, fractal sequence is designed to cover the input space super-uniformly (see Figure 3, center), and thus can sometimes accelerate convergence relative to a naïve Monte Carlo approach. In contrast to a pseudo-random number generator (Figure 3, left), however, the Sobol sequence makes no pretense of being random, and dependencies between successive patterns are

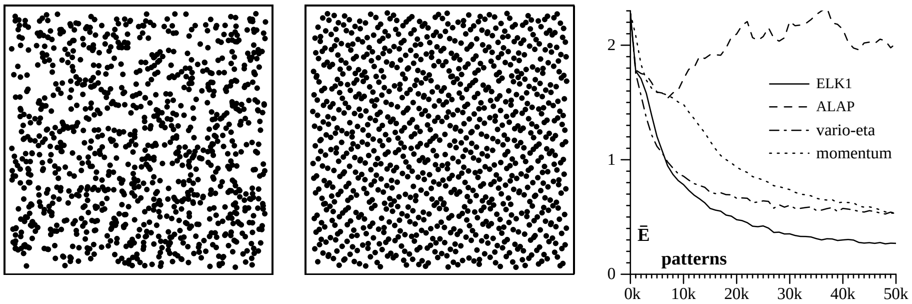

Figure 3: Results (right; *cf.* Figure 2, left) obtained when sampling input space according to the Sobol sequence (center, first 1000 points) instead of pseudo-randomly (left, 1000 points). ALAP fails; the other methods remain unaffected.

easy to spot even when the underlying algorithmic generator is not known.

Figure 3 (right) shows the results we obtained with the Sobol sequence. While the other three methods remained unaffected (compare Figure 2, left), ALAP failed to converge in all 25 runs. We tried varying its free parameters, but found that performance was improved only by essentially turning off the gain adaptation ($\mu \to 0$). This casts some doubt on ALAP's reliability in situations where independence between successive training patterns cannot be guaranteed.

Finally, we evaluated the sensitivity of the four algorithms to the (initial) learning rate η on a benchmark problem due to Almeida et al. (1998): a fully connected neural network with 2 inputs, 40 hidden units (tanh nonlinearity), and a single linear output is to learn the function

$$g(x,y) \;=\; \frac{\sin(20\sqrt{x^2+y^2})}{20\sqrt{x^2+y^2}} \;+\; \frac{1}{5}\cos(10\sqrt{x^2+y^2}) \;+\; \frac{y}{2} \;-\; 0.3\,, \tag{16}$$

shown in Figure 3 (left). In contrast to Almeida et al. (1998) we started from uniformly random initial weights in the range $[-0.3, 0.3]$, and avoided overfitting by training on uniformly random inputs (x, y) drawn from $[-1, 1]^2$. About equal, significant effort was expended in optimizing the free parameters of each method, yielding the values:

		mom.	vario-η	ALAP	ELK1
meta-learning rate	μ	—	—	0.5	40
smoothing parameter	α	0.7	0.5	0.7	—

Figure 3 (right) shows the number of training patterns required by each algorithm for an exponential trace (smoothing parameter 0.9) of its squared error to fall below 0.05. Each curve plots the average of 25 restarts from different initial weights against η, the value of the (intitial) learning rate parameter. With the

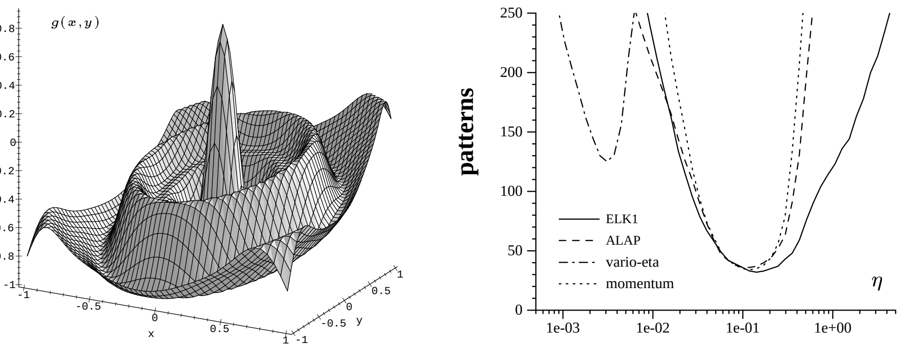

Figure 4: A benchmark function due to Almeida et al. (1998), and the number of training patterns (averaged over 25 runs) required by the four tested algorithms to fit it to criterion, *vs.* the (initial) learning rate η.

exception of vario-η, the performance of the algorithms near the optimal value of η is nearly identical. Away from the optimum, however, a clear difference emerges: while ALAP does offer some improvement over gradient descent with momentum, in particular for small η, ELK1 performs far better when the initial learning rate is larger than optimal. This characteristic robustness of ELK1 for large learning rates is the result of the normalization inherent in the calculation of the local gain vector $\vec{k}$ (6).

4 Conclusion

We have derived and tested a new method, based on work by Sutton (1992b), to adapt the local step sizes for stochastic gradient descent in multi-layer perceptrons. Compared to the alternative approaches of Zimmermann (1994) and Almeida et al. (1997, 1998), our ELK1 algorithm is computationally not much more expensive while eliminating an arbitrary smoothing parameter. On our benchmark problems it consistently outperforms the above alternatives; more experiments are needed to determine whether this holds in general. In contrast to the method of Almeida et al. (1997, 1998), ELK1 is not derailed by dependencies between successive training patterns, and handles large initial learning rates well. The algorithm might be improved further by incorporating diagonal Hessian information so as to relax the unity curvature assumption in (8).

Acknowledgments

This work was supported by the Swiss National Science Foundation under grants numbers 2100–045700.95/1 and 2000–052678.97/1.

References

Almeida, L. B., Langlois, T., & Amaral, J. D. (1997). On-line step size adaptation. Tech. rep. RT07/97, INESC, 9 Rua Alves Redol, 1000 Lisboa, Portugal, ◁ http://146.193.1.145/~tl/RT0797/ ▷.

Almeida, L. B., Langlois, T., Amaral, J. D., & Plakhov, A. (1998). Parameter adaptation in stochastic optimization. Tech. rep., INESC, 9 Rua Alves Redol, 1000 Lisboa, Portugal, ◁ ftp://146.193.2.131/pub/lba/-papers/adsteps.ps.gz ▷.

Battiti, R. (1989). Accelerated back-propagation learning: Two optimization methods. *Complex Systems*, *3*, 331–342.

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

Lapedes, A., & Farber, R. (1986). A self-optimizing, nonsymmetrical neural net for content addressable memory and pattern recognition. *Physica*, *D 22*, 247–259.

LeCun, Y., Simard, P. Y., & Pearlmutter, B. (1993). Automatic learning rate maximization in large adaptive machines. In Hanson, S. J., Cowan, J. D., & Giles, C. L. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 5, pp. 156–163. Morgan Kaufmann, San Mateo, CA.

Matthews, M. B. (1990). Neural network nonlinear adaptive filtering using the extended Kalman filter algorithm. In *Proceedings of the International Neural Networks Conference*, Vol. I, pp. 115–119 Paris, France.

Murata, N., Müller, K.-R., Ziehe, A., & Amari, S.-i. (1997). Adaptive on-line learning in changing environments. In Mozer, M. C., Jordan, M. I., & Petsche, T. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 9, pp. 599–605. The MIT Press, Cambridge, MA.

Plumer, E. S. (1995). Training neural networks using sequential-update forms of the extended Kalman filter. Tech. rep. LA-UR-95-422, Los Alamos National Laboratory.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing* (Second edition). Cambridge University Press.

Puskorius, G. V., & Feldkamp, L. A. (1991). Decoupled extended Kalman filter training of feedforward layered networks. In *Proceedings of the International Joint Conference on Neural Networks*, Vol. I, pp. 771–777 Seattle, WA. IEEE.

Schraudolph, N. N. (1998). A fast, compact approximation of the exponential function. Tech. rep. IDSIA-07-98, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, Corso Elvezia 36, 6900 Lugano, Switzerland. To appear in *Neural Computation.* ◁ ftp://ftp.idsia.ch/pub/nic/-exp.ps.gz ▷.

Schraudolph, N. N., & Sejnowski, T. J. (1996). Tempering backpropagation networks: Not all weights are created equal. In Touretzky, D. S., Mozer, M. C., & Hasselmo, M. E. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 8, pp. 563–569. The MIT Press, Cambridge, MA.

Shah, S., Palmieri, F., & Datum, M. (1992). Optimal filtering algorithms for fast learning in feedforward neural networks. *Neural Networks*, *5*, 779–787.

Silva, F. M., & Almeida, L. B. (1990). Speeding up back-propagation. In Eckmiller, R. (Ed.), *Advanced Neural Computers*, pp. 151–158 Amsterdam. Elsevier.

Singhal, S., & Wu, L. (1989). Training multilayer perceptrons with the extended Kalman filter. In Touretzky, D. S. (Ed.), *Advances in Neural Information Processing Systems. Proceedings of the 1988 Conference*, pp. 133–140 San Mateo, CA. Morgan Kaufmann.

Sutton, R. S. (1992a). Adapting bias by gradient descent: an incremental version of delta-bar-delta. In *Proc. 10th National Conference on Artificial Intelligence*, pp. 171–176 ◁ ftp://ftp.cs.umass.edu/pub/anw/pub/sutton/-sutton-92a.ps.gz ▷. The MIT Press, Cambridge, MA.

Sutton, R. S. (1992b). Gain adaptation beats least squares?. In *Proc. 7th Yale Workshop on Adaptive and Learning Systems*, pp. 161–166 ◁ ftp://ftp.cs.-umass.edu/pub/anw/pub/sutton/sutton-92b.ps.gz ▷.

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

Williams, R. J. (1992). Some observations on the use of the extended Kalman filter as a recurrent network learning algorithm. Tech. rep. NU-CCS-92-1, College of Computer Science, Northeastern University, Boston, MA 02115.

Yu, X.-H., Chen, G.-A., & Cheng, S.-X. (1993). Acceleration of backpropagation learning using optimised learning rate and momentum. *Electronics Letters*, *29*(14), 1288–1290.

Zimmermann, H. G. (1994). Neuronale Netze als Entscheidungskalkül. In Rehkugler, H., & Zimmermann, H. G. (Eds.), *Neuronale Netze in der Ökonomie: Grundlagen und finanzwirtschaftliche Anwendungen*, pp. 1–87. Vahlen Verlag, Munich.