Non-Linear Back-propagation: Doing Back-Propagation without Derivatives of the Activation Function.

John Hertz
Nordita, Blegdamsvej 17, 2100 Copenhagen, Denmark
Email: hertz@nordita.dk

Anders Krogh
Electronics Institute, Technical University of Denmark, Building 349
2800 Lyngby, Denmark, Email: krogh@nordig.ei.dth.dk

Benny Lautrup
The Niels Bohr Institute, Blegdamsvej 17, 2100 Copenhagen, Denmark
Email: lautrup@nbivax.nbi.dk

Torsten Lehmann
Electronics Institute, Technical University of Denmark, Building 349
2800 Lyngby, Denmark, Email: tlehmann@eileen.ei.dth.dk

March 4, 1994

Abstract

The conventional linear back-propagation algorithm is replaced by a non-linear version, which avoids the necessity for calculating the derivative of the activation function. This may be exploited in hardware realizations of neural processors. In this paper we derive the non-linear back-propagation algorithms in the framework of recurrent back-propagation and present some numerical simulations of feed-forward networks on the NetTalk problem. A discussion of implementation in analog VLSI electronics concludes the paper.

1 Introduction

From a simple rewriting of the back-propagation algorithm [RHW86] a new family of learning algorithms emerges which we call non-linear back-propagation. In the normal back-propagation algorithm one calculates the errors on the hidden units from the errors on the output neurons by means of a linear expression. The non-linear algorithms presented here have the advantage that the back-propagation of errors goes through the same non-linear units as the forward propagation of activities.

Using this method it is no longer necessary to calculate the derivative of the activation function for each neuron in the network as it is in standard back-propagation. Whereas the derivatives are trivial to calculate in a simulation, they appear to be of major concern when implementing the algorithm in hardware, be it electronic or optical. For these reasons we believe that non-linear back-propagation is very well suited for hardware implementation. This is the main motivation for this work.

In the limit of infinitely small learning rate the non-linear algorithms become identical to standard back-propagation. For small learning rates the performance of the new algorithms is therefore comparable to standard back-propagation, whereas for larger learning rates it performs better. The algorithms generalize easily to recurrent back-propagation [Alm87, Pin87] of which the standard back-propagation algorithm for feed-forward networks is a special case.

In this paper we derive the non-linear back-propagation (NLBP) algorithms in the framework of recurrent back-propagation and present some numerical simulations of feed-forward networks on the NetTalk problem [SR87]. A discussion of implementation in analog VLSI electronics concludes the paper.

2 The Algorithms

In this section we present the algorithms for non-linear back-propagation. The derivation can be found in the next section.

We consider a general network, recurrent or feed-forward, with N neurons. The neuron activities are denoted V_i and the weight of the connection from neuron j to neuron i is denoted w_{ij}. Threshold values are included by means of a fixed bias neuron numbered $i = 0$.

The activation (output) of a unit i in the network is then

$$V_i = g(h_i), \qquad h_i = \sum_j w_{ij} V_j + \xi_i, \tag{1}$$

where g is the activation function and ξ_i is external input the unit. These equations are applied repeatedly for all neurons untill the state of activity converges towards a fixed point. For a feed-forward network this is guaranteed to happen, and the activities should be evaluated in the forward direction, *i.e.* from input towards output. For a recurrent network there is no guarantee that these equations will converge towards a fixed point, but we shall assume this to be the case. The equations are, however, simplest in the general form.

The error of the network on input pattern is defined as

$$\epsilon_k = \zeta_k - V_k \tag{2}$$

where ζ_k are the target values for the output units when the input is pattern ξ_i and V_k is the actual output for that same input pattern. For non-output units $\epsilon_i \equiv 0$.

We define the *backward activations* as

$$y_i = g(h_i + \frac{\eta_i}{\alpha_i}[\sum_k \frac{\alpha_k}{\eta_k}(y_k - V_k)w_{ki} + \epsilon_i]), \tag{3}$$

where the constants η_i and α_i will be discussed shortly. These variables are "effective" or "moving" targets for hidden units in the network. For output units in a feed-forward network the sum on k is empty, and if the errors ϵ_i are all zero, these equations have the simple solution $y_i = V_i$. For non-zero error iteration of these equations is likewise assumed to lead to a fixed point in the backwards activation state (one can easily show that if the forward equations (1) converge, the backwards equations will also converge). Notice that during iteration of the backward activations we keep the forward activations fixed.

Now, consider a set of input-output patterns indexed by $\mu = 1, \ldots, p$, and assume that the squared error is used as the cost function,

$$E_{sq} = \frac{1}{2}\sum_{\mu=1}^{p}\sum_k (\epsilon_k^\mu)^2. \tag{4}$$

In terms of these new variables the non-linear back-propagation is then like delta-rule learning,

$$\Delta w_{ij} = \alpha_i \sum_\mu (y_i^\mu - V_i^\mu)V_j^\mu. \tag{5}$$

The constants α_i and η_i are replacements for the usual learning rate, and it is required that η_i/α_i is "small". The reason we speak of a family of of algorithms is that different choices of α yield different algorithms. Here the parameters are allowed to differ from unit to unit, but usually they will be the same for large groups of units (*e.g.* ones forming a layer) which simplifies the equations. We consider two choices of α particularly interesting: $\alpha_i = \eta_i$ and $\alpha_i = 1$. For the first of these η plays a role similar to the learning rate in delta-rule learning, since α is replaced by η in (5).

For the entropic error measure [HKP91] the weight update is the same (5), but y_i is defined as

$$y_i = g(h_i + \frac{\eta_i}{\alpha_i} \sum_k \frac{\alpha_k}{\eta_k}(y_k - V_k) w_{ki}) + \frac{\eta_i}{\alpha_i}\epsilon_i. \qquad (6)$$

In this case the weight update for an output unit is exactly like the standard one, $\Delta w_{ij} = \eta_i \sum_\mu \epsilon_i^\mu V_j^\mu$. For a network with linear output units optimizing the squared error the two equations for y_i coincide.

Obviously these algorithms can be used *online*, *i.e.* changing the weights after each pattern, just as is common when using the standard back-propagation algorithm.

Finally we would like to explicitly show the important cases of $\alpha_i = 1$ and $\alpha_i = \eta_i$ for a *feed-forward* network. For simplicity the index μ will be dropped. Notation:

l labels the layers from 0 (output) to L (input).

w_{ij}^l is the weight from unit j in layer $l + 1$ to unit i in layer l.

Any other variable (like y and V) with superscript l refers to that variable in layer l.

It will be assumed that η_i is the same for all units in a layer, $\eta_i^l = \eta^l$. The error on the output units are denoted ϵ_i as before. Here are the two versions:

$\boldsymbol{\alpha}_i = \mathbf{1}.$

Output unit:
$y_i^0 = g(h_i^0 + \eta^0\epsilon_i)$ (squared error).
$y_i^0 = V_i^0 + \eta^l\epsilon_i$ (entropic error).

Hidden unit: $y_i^l = g(h_i^l + \frac{\eta^l}{\eta^{l-1}} \sum_k (y_k^{l-1} - V_k^{l-1}) w_{ki}^{l-1})$

Weight update $\Delta w_{ij}^l = (y_i^l - V_i^l) V_j^{l+1}$

$\boldsymbol{\alpha}_i = \boldsymbol{\eta}_i.$

Output unit:
$y_i^0 = g(h_i^0 + \epsilon_i)$ (squared error).
$y_i^0 = V_i^0 + \epsilon_i$ (entropic error).

Hidden unit: $y_i^l = g(h_i^l + \sum_k (y_k^{l-1} - V_k^{l-1}) w_{ki}^{l-1})$

Weight update $\Delta w_{ij}^l = \eta^l (y_i^l - V_i^l) V_j^{l+1}$

3 Derivation of NLBP

In this section we derive the non-linear back-propagation in the framework of recurrent back-propagation. As an introduction, we follow the derivation of recurrent back-propagation in [HKP91, p. 172–175]. See also [Pin87].

3.1 Standard recurrent back-propagation

Assume fixed points of the network are given by (1), and that the learning is governed by an error measure E like (4). If we define $\epsilon_i = -\frac{\partial E}{\partial V_i}$, which for (4) is identical to (2), the gradient descent learning rule is

$$\Delta w_{pq} = \eta \sum_k \epsilon_k \frac{\partial V_k}{\partial w_{pq}}. \tag{7}$$

Differentiation of the fixed point equation (1) for V_i yields

$$\frac{\partial V_k}{\partial w_{pq}} = (\mathsf{L}^{-1})_{kp} V_p' V_q \tag{8}$$

with $V_i' = g'(h_i)$ and the matrix L given by

$$L_{ij} = \delta_{ij} - V_i' w_{ij}. \tag{9}$$

If this matrix is positive definite, the dynamics will be contractive around a fixed point. According to our assumptions this must therefore be the case.

Defining

$$\delta_p = V_p' \sum_k \epsilon_k (\mathsf{L}^{-1})_{kp}, \tag{10}$$

the weight update can be written as

$$\Delta w_{pq} = \eta \delta_p V_q \tag{11}$$

and the δ's are the solutions to

$$\delta_k = V_k' (\sum_i \delta_i w_{ik} + \epsilon_k). \tag{12}$$

These are the standard back-propagation equations for a general network. In a feed-forward network they converge to a fixed point when iterated in the backwards direction from output towards input. For a general recurrent net they will converge towards a fixed point when the L-matrix is positive definite, as may easily be demonstrated.

3.2 Non-linear back-propagation

If the error measure is given by (4) the derivatives of the error measure are

$$\epsilon_k = \zeta_k - V_k \qquad \text{for the output units, 0 otherwise} \tag{13}$$

For an output unit in a feed-forward network we thus find $\delta_k = V_k'(\zeta_k - V_k)$. One of the ideas of the non-linear back-propagation is to *force* that interpretation on all the units, defining 'effective targets' y such that

$$\delta_i \propto y_i - V_i. \tag{14}$$

For small η eq. (11) can then be interpreted as a first order Taylor expansion:

$$\begin{aligned} \Delta w_{ij} &= \eta \delta_i V_j = \eta V_i' (\sum_k \delta_k w_{ki} + \epsilon_i) V_j \\ &\simeq [g(h_i + \eta[\sum_j \delta_j w_{ji} + \epsilon_i]) - g(h_i)] V_j \end{aligned} \tag{15}$$

where $g(h_i) = V_i$. The first term in the brackets is just the output of a unit with the normal input and the back-propagated error added – the effective target:

$$y_i = g(h_i + \eta[\sum_k \delta_k w_{ki} + \epsilon_i]). \tag{16}$$

Note however that the "integration" in (15) is quite arbitrary; it is just one possibility out of many given by

$$\Delta w_{ij} = \eta \delta_i V_j \simeq \alpha_i [g(h_i + \frac{\eta_i}{\alpha_i}[\sum_k \delta_k w_{ki} + \epsilon_i]) - g(h_i)] V_j \tag{17}$$

where the α_i's are arbitrary parameters similar to the learning rates η_i. For consistency one now has to replace δ_k by

$$\delta_k = \frac{\alpha_k}{\eta_k}(y_k - V_k) \tag{18}$$

Then y_i is finally given by (3) and the weight update by (5).

Formally the "integration" in eq. (15) is only valid for small η (or small η_i/α_i in (17)). But for larger η there is no guarantee that the clean gradient descent converges anyway, and these nonlinear versions might well turn out to work better.

By making α_i very large compared to η_i, one can make the NLBP indistinguishable from standard back-propagation (the Taylor expansions will be almost exact). That would be at the expense of high numerical instability, because y_i would be very close to V_i and the formula for the weight update, $\Delta w_{ij} = \alpha_i(y_i - V_i)$, would require very high precision. On the other hand, very small α's are likely to take the algorithm too far from gradient descent. For these reasons we believe that the most interesting range is $\eta_i \leq \alpha_i \leq 1$ (assuming that $\eta_i < 1$). The limit $\alpha_i = \eta_i$ is the most stable, numerically, and $\alpha_i = 1$ is the most gradient-descent-like limit. Notice that if the ratios $\lambda = \eta_i/\alpha_i$ are the same for all neurons in the network then the equations take the simpler form

$$y_i = g(h_i + \sum_k (y_k - V_k) w_{ki} + \lambda \epsilon_i), \tag{19}$$

and

$$\Delta w_{ij} = \alpha_i (y_i - V_i) V_j \tag{20}$$

3.3 Entropic error measure

The entropic error measure is

$$E = \sum_i \left[\tfrac{1}{2}(1 + V_i) \log \frac{1 + V_i}{1 + \zeta_i} + \tfrac{1}{2}(1 - V_i) \log \frac{1 - V_i}{1 - \zeta_i} \right] \tag{21}$$

if the activation function g is equal to tanh. A similar error measures exists for other activation functions like $g(x) = (1 + e^{-x})^{-1}$. It can be shown that for this and similar error measures

$$\partial_i E = -\frac{\partial E}{\partial V_i} = \frac{\epsilon_i}{V_i'}. \tag{22}$$

Instead of (3) y_i should then be defined as (6).

3.4 Internal representations

For a feedforward architecture with a single hidden layer, the weight change formulas resemble those obtained using the method of internal representations [AKH90]. However, they are not quite the same. Using our present notation, in the present method we find a change for the weight from hidden unit j to output unit i of $\eta[\zeta_i - g(\sum_k w_{ik} V_k)] V_j$, while the internal representation approach it is $\eta[\zeta_i - g(\sum_k w_{ik} y_k)] y_j$. For the input-to-hidden layer the expressions for the weight changes in the two approaches look the same, but the effective targets y_j in them are different. They are both calculated by back-propagating errors $\zeta_i - V_i$ from the output units, but in the present case these V_i are simply the result $g(\sum_j w_{ij} V_j)$ of the forward propagation, while in the internal representations approach, $V_i = g(\sum_j w_{ij} y_j)$, i.e. they are obtained by propagating the effective targets on the hidden layer forward through the hidden-to-output weights.

4 Test of Algorithm

The algorithms have been tested on the NetTalk problem using a feed-forward network with an input window of 7 letters and one hidden layer consisting of 80 hidden units. The algorithms were run in online mode and a momentum term of 0.8 was used. They were tested for these values of α_i: η_i, 0.25, 0.5, 0.75, and 1.0. The learning rate η_i was scaled according to the number n_i of connections feeding into unit i, such that $\eta_i = \eta/\sqrt{n_i}$ — a standard method often used in back-propagation. Several values were tried for η. The initial weights were chosen at random uniformly between $-0.5/\sqrt{n_i}$ and $0.5/\sqrt{n_i}$. For each value of α and η, 50 cycles of learning was done, and the squared error normalized by the number of examples was recorded at each cycle.

For all runs the final error was plotted as a function of η, see figure 1. Clearly the runs with $\alpha = 1$ are almost indistinguisable from standard back-propagation, which is also shown. As a "corollary" these plots show that the entropic error measure is superior – even when the object is to minimize squared error (see also [SLF88]).

In figure 2 the time development of the error is shown for the extreme values of α and η fixed.

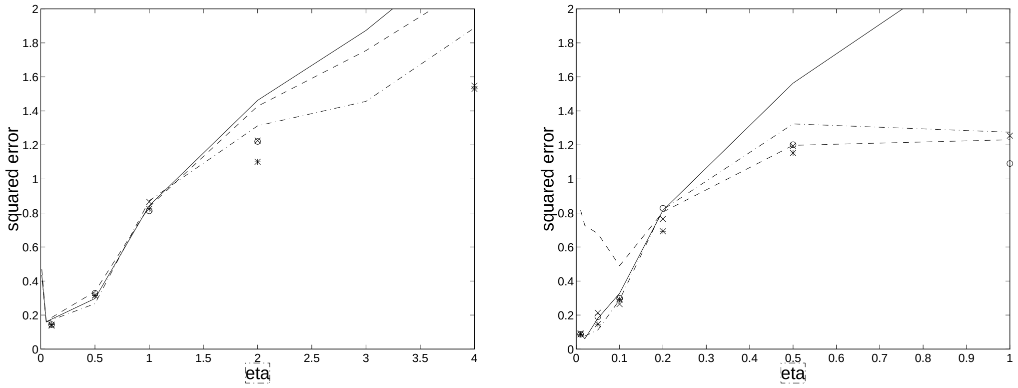

Figure 1: Plots showing the squared error after 50 training epocs as a function of the size of the learning rate η. The left plot is for the squared error cost function and the right plot for the entropic error measure. The solid line represents the standard back-propagation algorithm, the dashed line the nonlinear back-propagation algorithm with $\alpha = \eta$, and the dot-dashed line the one with $\alpha = 1$. The NLBP with $\alpha = 0.25$ (*), $\alpha = 0.5$ (x), and $\alpha = 0.75$ (o) are also shown.

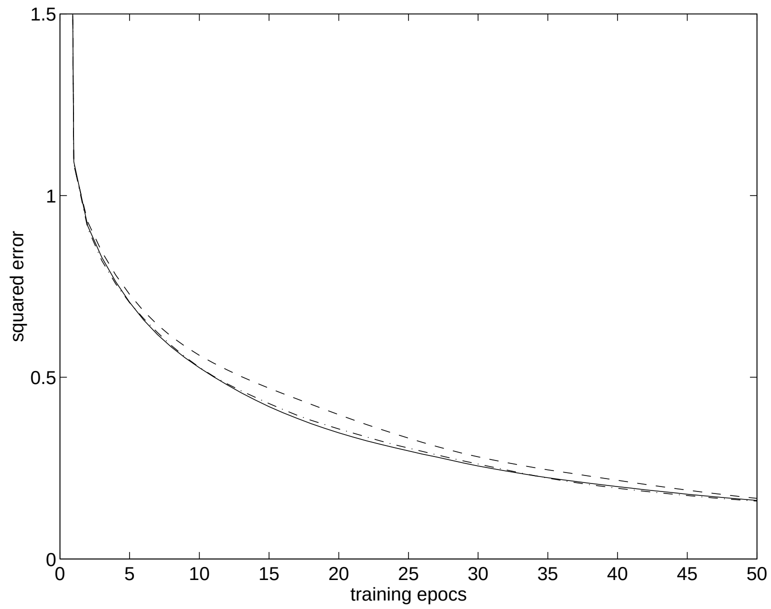

Figure 2: The decrease in error as a function of learning time for standard back-propagation (solid line), NLBP with $\alpha = 1$ (dashed line), and $\alpha = 1$ (dot-dashed line). In all three cases the squared error function and $\eta = 0.05$ was used.

5 Hardware Implementations

All the algorithms can be mapped topologically onto analog VLSI in a straight-forward manner, though selecting the most stable is the best choice because of the limited precision of this technology. In this section, we will give two examples of the implementation of the algorithms for a feed-forward network using $\alpha_i = \eta_i$ and the squared error cost function:

Defining a neuron *error*, ϵ_i^l, for each layer (note $\epsilon_i^l \equiv \epsilon_i$ for the output layer)

$$\epsilon_i^l = \begin{cases} \zeta_i - V_i^0, & \text{for } l = 0 \\ \sum_k \delta_k^{l-1} w_{ki}^{l-1}, & \text{for } l > 0 \end{cases}, \tag{23}$$

we can write (18) for $\alpha_i = \eta_i$ as

$$\delta_i^l = g(h_i^l + \epsilon_i^l) - g(h_i^l). \tag{24}$$

Thus, topologically this version of non-linear back-propagation maps in exactly the same way on hardware as the original back-propagation algorithm — the only difference being how δ_i^l is calculated ($\delta_i^l = g'(h_i^l)\epsilon_i^l$ for back-propagation) [LB93]:

The system consists of two modules: A "synapse module" which calculates Δw_{ij}^l, propagates h_i^l forward and propagates backward ϵ_j^{l+1}; and a "neuron module" which forward propagates V_i^l and backward propagates δ_i^l. To change the learning algorithm to non-linear back-propagation, it is thus necessary only to change the "neuron module".

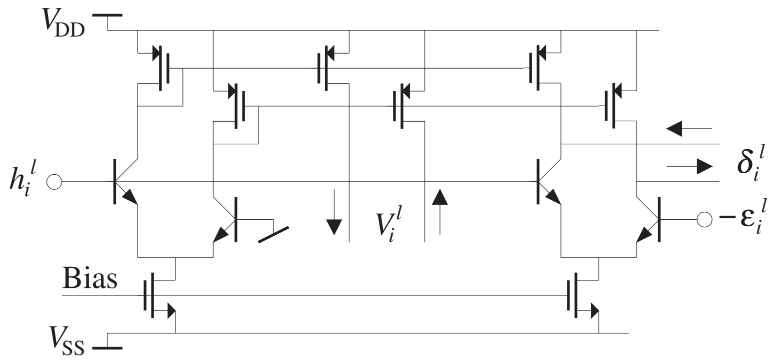

Figure 3: Continuous time non-linear back-propagation "neuron module" with hyperbolic tangent activation function. The precision is determined by the matching of the two differential pairs.

A simple way to implement a sigmoid-like activation function is by the use of a differential pair. Figure 3 shows a non-linear back-propagation "neuron module" with a hyperbolic tangent activation function. Applying h_i^l and $-\epsilon_i^l$ as voltages gives the V_i^l and δ_i^l outputs as differential currents (the "synapse module" can just as well calculate $-\epsilon_i^l$ as ϵ_i^l). It is interesting to notice that the circuit structure is identical to the one used in [Bog93] to calculate the derivative of the activation function: Replacing ϵ_i^l by a small constant, Δ, the δ_i^l output will approximate $g'(h_i^l) \cdot \Delta$. Using the circuit in the proposed way, however, gives better accuracy: The ϵ_i^l is not a "small" quantity which makes the inherent inaccuracies less significant, relatively. Further, the circuit calculates the desired δ_i^l directly, eliminating the need of an extra multiplier — and thus eliminating a source of error. The accuracy of the circuit is determined by the matching of the two differential pairs and of the bias sources. This can be in the order of 1% of the output current magnitude.

In a "standard implementation" of the "synapse module", the h_i^l and ϵ_i^l outputs will be available as currents and the V_i^l and δ_i^l inputs must be applied as voltages. Thus the above implementation requires accurate transresistances to function properly. Also, as the same function is used to calculate the V_i^ls and the δ_i^ls, it would be preferable to use the same hardware as this eliminates the need of matched components. This is possible if the system is not required to function in continuous time, though the output has to be sampled (which introduces errors).

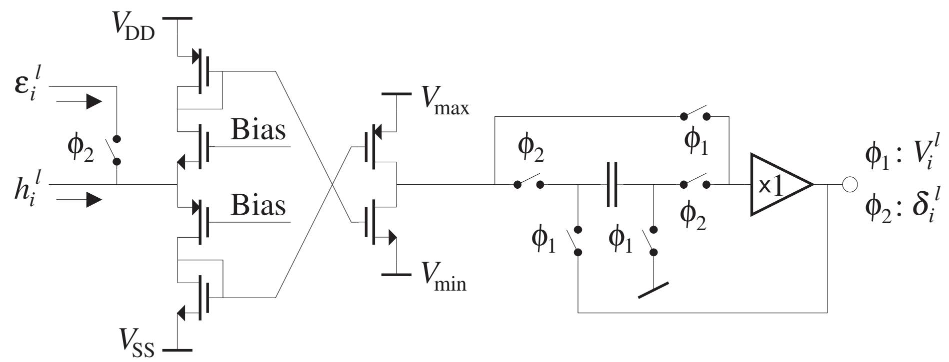

Figure 4: Discrete time non-linear back-propagation "neuron module" with non-linear activation function. The precision is determined by the accuracy of the switched capacitor.

In figure 4 such a simplified discrete time "neuron module" which reuses the activation function block and which has current input/voltage outputs is shown. During the ϕ_1 clock phase, V_i^l is available at the output and is sampled at the capacitor. During the ϕ_2 clock phase, δ_i^l is available at the output. The activation function saturates such that $V_{\min} \leq V_i^l \leq V_{\max}$, though it is not very well defined. This is of no major concern, however; the accuracy is determined by the switched capacitor. Using design techniques to reduce charge injection and redistribution, the accuracy can be in the order of 0.1% of the output voltage swing.

As illustrated, the non-linear back-propagation learning algorithm is well suited for analogue hardware implementation, though offset compensation techniques still have to be employed. It maps topologically on hardware in the same way as ordinary back-propagation, but the circuit to calculate the δ_i^ls is much more efficient: It can approximate the learning algorithm equations more accurately and as the algorithm requires only simple operations apart from the activation function, design efforts can be put on the electrical specifications of the hardware (input impedance, speed, noise immunity, etc.) and on the general shape of the sigmoid-like activation function. Further, as the algorithm requires only one "special function", it has the potential of very high accuracy through reuse of this function block.

6 Conclusion

A new family of learning algorithms have been derived that can be thought of as "non-linear gradient descent" type algorithms. For appropriate values of the parameters they are almost identical to standard back-propagation. By numerical simulations of feed-forward networks learning the NetTalk problem it was shown that the performance of these algorithms were very similar to standard back-propagation for the range of parameters tested.

The algorithms have two important properties that we believe make them easier to implement in electronic hardware than the standard back-propagation algorithm. First, no derivatives of the activation function need to be calculated. Second, the back-propagation of errors is through the same non-linear network as the forward propagation, and not a linearized network as in standard back-propagation. Two examples of how analogue electronic hardware can utilize these properties have been given. These advantages may also be expected to carry over to optical implementations.

References

[AKH90] G. I. Thorbergsson A. Krogh and J. A. Hertz. A cost function for internal representations. In *Neural Information Processing Systems* **2**, pages 733–740, D. S Touretzky, ed. San Mateo CA: Morgan Kauffmann, 1990.

[Alm87] L.B. Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In M. Caudill and C. Butler, editors, *IEEE International Conference on Neural Networks*, volume 2, pages 609–618, San Diego 1987, 1987. IEEE, New York.

[AR88] J.A. Anderson and E. Rosenfeld, editors. *Neurocomputing: Foundations of Research.* MIT Press, Cambridge, 1988.

[Bog93] G. Bogason. Generation of a neuron transfer function and its derivative. *Electronics Letters*, 29:1867–1869, 1993.

[HKP91] J.A. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, Redwood City, 1991.

[LB93] T. Lehmann and E. Bruun. Analogue VLSI implementation of back-propagation learning in artificial neural networks. In *Proc. 11'th European Conference on Circuit Theory and Design*, pages 491–496, Davos 1993, 1993.

[Pin87] F.J. Pineda. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 59:2229–2232, 1987.

[RHW86] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. Reprinted in [AR88].

[SLF88] S.A. Solla, E. Levin, and M. Fleisher. Accelerated learning in layered neural networks. *Complex Systems*, 2:625–639, 1988.

[SR87] T.J. Sejnowski and C.R. Rosenberg. Parallel networks that learn to pronounce english text. *Complex Systems*, 1:145–168, 1987.