Nonlinear Backpropagation: Doing Backpropagation Without Derivatives of the Activation Function

John Hertz, Anders Krogh, Benny Lautrup, and Torsten Lehmann

***Abstract*—The conventional linear backpropagation algorithm is replaced by a nonlinear 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 nonlinear backpropagation algorithms in the framework of recurrent backpropagation and present some numerical simulations of feedforward networks on the NetTalk problem. A discussion of implementation in analog very large scale integration (VLSI) electronics concludes the paper.**

***Index Terms*— Backpropagation, neural-network implementation, neural networks, recurrent backpropagation.**

I. Introduction

FROM A simple rewriting of the backpropagation algorithm [1] a new family of learning algorithms emerges which we call nonlinear backpropagation. In the normal backpropagation algorithm one calculates the errors on the hidden units from the errors on the output neurons by means of a linear expression. The nonlinear algorithms presented here have the advantage that the backpropagation of errors goes through the same nonlinear 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 backpropagation. 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 nonlinear backpropagation is very well suited for hardware implementation. This is the main motivation for this work.

In the limit of infinitely small learning rate the nonlinear algorithms become identical to standard backpropagation. For small learning rates the performance of the new algorithms is therefore comparable to standard backpropagation, whereas for larger learning rates it performs better. The algorithms generalize easily to recurrent backpropagation [2], [3] of which the standard backpropagation algorithm for feedforward networks is a special case.

In this paper we derive the nonlinear backpropagation (NLBP) algorithms in the framework of recurrent backpropagation and present some numerical simulations of feedforward networks on the NetTalk problem [4]. A discussion of implementation in analog very large scale integration (VLSI) electronics concludes the paper.

Manuscript received August 15, 1995; revised August 5, 1997.

J. Hertz is with Nordita, 2100 Copenhagen, Denmark.

A. Krogh is with the Center for Biological Sequence Analysis, Technical University of Denmark, 2800 Lyngby, Denmark.

B. Lautrup is with the The Niels Bohr Institute, 2100 Copenhagen, Denmark.

T. Lehmann is with the Department of Information Technology, Technical University of Denmark, 2800 Lyngby, Denmark.

Publisher Item Identifier S 1045-9227(97)07971-X.

II. The Algorithms

In this section we present the algorithms for nonlinear backpropagation. The derivation can be found in the next section.

We consider a general network, recurrent or feedforward, 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

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

where g is the activation function and ξ_i is external input to the unit. These equations are applied repeatedly for all neurons until the state of activity converges toward a fixed point. For a feedforward network this is guaranteed to happen, and the activities should be evaluated in the forward direction, i.e., from input toward output. For a recurrent network there is no guarantee that these equations will converge toward 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 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 nonoutput units $\epsilon_i \equiv 0$.

We define the *backward activations* as

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

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 feedforward 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 nonzero error, iteration of these equations is assumed to lead to a fixed point in the backward activation state [one can easily show that if the forward equations (1) converge, the backward 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 $\mu = 1, \cdots, p$, and assume that the squared error is used as the cost function

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

In terms of these new variables the nonlinear backpropagation is then like delta-rule learning

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

The constants α_i and η_i are replacements for the usual learning rate. 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 [5] the weight update is the same (5), but y_i is defined as

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

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 we obtain the same equations for y_i.

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

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

l	numbers the layers from zero (output) to L (input).
w_{ij}^l	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:

$\boldsymbol{\alpha_i = 1}$

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: $y_i^l = g(h_i^l + (\eta^l/\eta^{l-1}) \sum_k (y_k^{l-1} - V_k^{l-1}) w_{ki}^{l-1})$
Weight update: $\Delta w_{ij}^l = (y_i^l - V_i^l) V_j^{l+1}$

$\boldsymbol{\alpha_i = \nu_i}$

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

III. Derivation of NLBP

In this section we derive the nonlinear backpropagation in the framework of recurrent backpropagation. As an introduction, we follow the derivation of recurrent backpropagation in [5, p. 172–175]. See also [3].

A. Standard Recurrent Backpropagation

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 $\epsilon_i = -(\partial E / \partial V_i)$, 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

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

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

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

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

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

These are the standard backpropagation equations for a general network. In a feedforward network they converge to a fixed point when iterated in the backward direction from output toward input. For a general recurrent net they will converge toward a fixed point when the L-matrix is positive definite, as may easily be demonstrated.

B. Nonlinear Backpropagation

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

$$\epsilon_k = \begin{cases} \zeta_k - V_k & \text{for the output units} \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

For an output unit in a feedforward network we thus find $\delta_k = V_k'(\zeta_k - V_k)$. One of the ideas of the nonlinear backpropagation 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 (11) can then be interpreted as a first-order Taylor expansion

$$\Delta w_{ij} = \eta \delta_i V_j = \eta V_i' \left(\sum_k \delta_k w_{ki} + \epsilon_i \right) V_j$$
$$\simeq \left[g\left(h_i + \eta \left[\sum_j \delta_j w_{ji} + \epsilon_i \right] \right) - g(h_i) \right] V_j \quad (15)$$

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 backpropagated error added—the effective target:

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

Using this definition of y_i and the definition (14) of δ_i, we see that for small η the weight change will be essentially the same as in standard backpropagation. Here, however, we treat these equations as an algorithm in its own right, independently of how good an approximation to standard backpropagation it may be. We show numerically below that it converges at least as well as standard backpropagation, even if it does not approximate the standard algorithm well. Note however that the "integration" in (15) is quite arbitrary; it is just one possibility out of many given by

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

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

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

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

Formally the "integration" in (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 backpropagation (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 $\eta_i \le \alpha_i \le 1$. 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

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

and

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

C. Entropic Error Measure

The entropic error measure is

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

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

$$\partial_i E = -\frac{\partial E}{\partial V_i} = \frac{\epsilon_i}{V_i'}. \quad (22)$$

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

D. Internal Representations

For a feedforward architecture with a single hidden layer, the weight change formulas resemble those obtained using the method of internal representations [6]. 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 backpropagating 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.

IV. Test of Algorithm

The algorithms have been tested on the NetTalk problem using a feedforward network with an input window of seven 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: $\eta_i, 0.25, 0.5, 0.75$, 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 backpropagation. 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 Fig. 1. Clearly the runs with $\alpha = 1$ are almost indistinguisable from standard backpropagation, 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 [7]). Also, NLBP seems superior to normal backpropagation for large learning rates which is important to hardware implementations (cf. [8]).

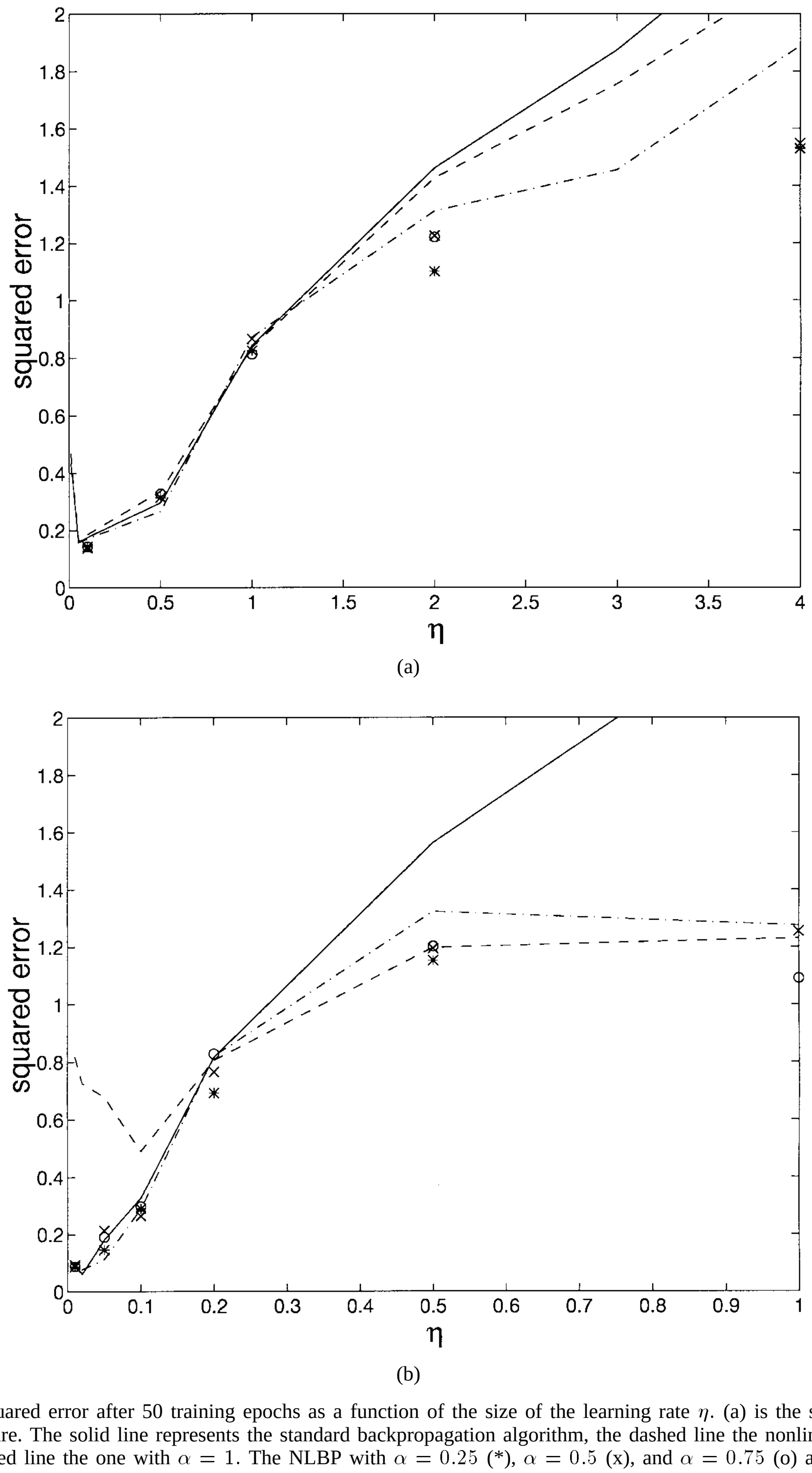

Fig. 1. Plots showing the squared error after 50 training epochs as a function of the size of the learning rate η. (a) is the squared error cost function and (b) is the entropic error measure. The solid line represents the standard backpropagation algorithm, the dashed line the nonlinear backpropagation 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.

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

V. 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 feedforward 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

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

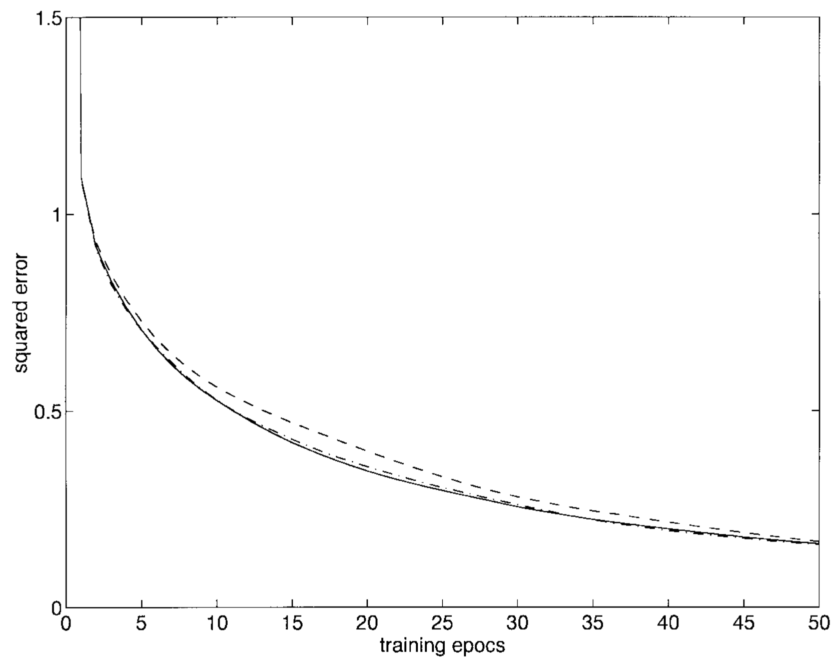

Fig. 2. The decrease in error as a function of learning time for standard backpropagation (solid line), NLBP with $\alpha = \nu$ (dashed line), and $\alpha = 1$ (dot-dashed line). In all three cases the squared error function and $\eta = 0.05$ was used.

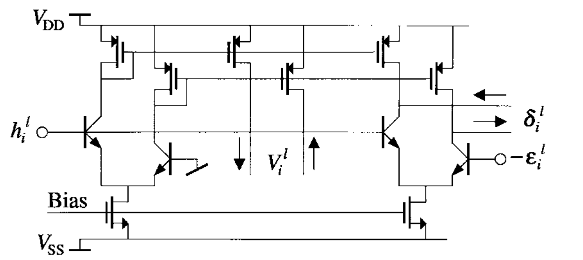

Fig. 3. Continuous time nonlinear backpropagation "neuron module" with hyperbolic tangent activation function. The precision is determined by the matching of the two differential pairs.

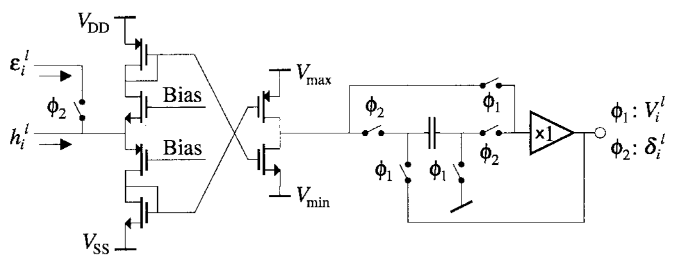

Fig. 4. Discrete time nonlinear backpropagation "neuron module" with nonlinear activation function. The precision is determined by the accuracy of the switched capacitor.

Thus, topologically this version of nonlinear backpropagation maps in exactly the same way on hardware as the original backpropagation algorithm—the only difference being how δ_i^l is calculated ($\delta_i^l = g'(h_i^l)\epsilon_i^l$ for backpropagation) [9] (see also [10]).

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 nonlinear backpropagation, it is thus necessary only to change the "neuron module."

A simple way to implement a sigmoid-like activation function is by the use of a differential pair. Fig. 3 shows a nonlinear backpropagation "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 is very similar to the one used in [11] 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).

In Fig. 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. The six transistors can be replaced by any current in/voltage out (nonlinear) circuit. Using design techniques to reduce charge injection and redistribution, the accuracy can be in the order of 0.1% of the output voltage swing.

The implementation of the "synapse module" compatible with the (nonlinear) backpropagation algorithm is not a trivial task. In particular the presence of various offset errors are problematic. It is not the purpose of the present paper to deal with these matters, though, which have been the concern of other authors (see [12] and [13]).

As illustrated, the nonlinear backpropagation learning algorithm is well suited for analog hardware implementation, though offset compensation techniques still have to be employed. It maps topologically on hardware in the same way as ordinary backpropagation, 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.

Regarding optical implementations of gradient descent like learning, we would expect NLBP to offer similar advantages over normal backpropagation as in electronic implementations (cf. [14]).

VI. Conclusion

A new family of learning algorithms have been derived that can be thought of as "nonlinear gradient descent" type algorithms. For appropriate values of the parameters they are almost identical to standard backpropagation. By numerical simulations of feedforward networks learning the NetTalk problem it was shown that the performance of these algorithms were very similar to standard backpropagation 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 backpropagation algorithm. First, no derivatives of the activation function need to be calculated. Second, the backpropagation of errors is through the same nonlinear network as the forward propagation, and not a linearized network as in standard backpropagation. Two examples of how analog electronic hardware can utilize these properties have been given. These advantages may also be expected to carry over to optical implementations.

References

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by backpropagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[2] L. B. Almeida, "A learning rule for asynchronous perceptrons with feedback in a combinatorial environment," in *IEEE Int. Conf. Neural Networks*, M. Caudill and C. Butler, Eds. San Diego, CA, vol. 2, 1987, pp. 609–618.

[3] F. J. Pineda, "Generalization of backpropagation to recurrent neural networks," *Phys. Rev. Letters*, vol. 59, pp. 2229–2232, 1987.

[4] T. J. Sejnowski and C. R. Rosenberg, "Parallel networks that learn to pronounce English text," *Complex Syst.*, vol. 1, pp. 145–168, 1987.

[5] J. A. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation.* Reading, MA: Addison-Wesley, 1991.

[6] A. Krogh, G. I. Thorbergsson, and J. A. Hertz, "A cost function for internal representations," in *Advances in Neural Inform. Processing Syst.*, D. S. Touretzky, Ed., Denver, CO, 1989, vol. 2, pp. 733–740.

[7] S. A. Solla, E. Levin, and M. Fleisher, "Accelerated learning in layered neural networks," *Complex Syst.*, vol. 2, pp. 625–639, 1988.

[8] L. Tarassenko, J. Tombs, and G. Cairns, "On-chip learning with analog VLSI neural networks," *Int. J. Neural Syst.*, vol. 4, no. 4, pp. 419–426, 1993.

[9] T. Lehmann and E. Bruun, "Analog VLSI implementation of backpropagation learning in artificial neural networks," in *Proc. 11th European Conf. Circuit Theory and Design*, 1993, pp. 491–496.

[10] T. Shima, T. Kimura, Y. Kamatani, T. Itakura, Y. Fujita, and T. Iida, "Neuro chips with on-chip backpropagation and/or Hebbian learning," *IEEE J. Solid-State Circuits*, vol. 27, pp. 1868–1876, 1992.

[11] G. Bogason, "Generation of a neuron transfer function and its derivative," *Electron. Lett.*, vol. 29, pp. 1867–1869, 1993.

[12] T. Morie and Y. Amemiya, "An all-analog expandable neural-network LSI with on-chip backpropagation learning," *IEEE J. Solid-State Circuits*, vol. 29, pp. 1086–1093, 1994.

[13] A. J. Montalvo, P. W. Hollis, and J. J. Paulos, "On-chip learning in the analog domain with limited precision circuits," in *Int. Symp. Circuits Syst.*, 1992, pp. I-196–I-201.

[14] S. R. Skinner, J. E. Steck, and E. C. Behrman, "Optical neural-network using Kerr-type nonlinear materials," in *4th Int. Conf. Microelectron. Neural Networks Fuzzy Syst.*, 1994, pp. 12–15.

John Hertz was born in Bethlehem, PA, in 1945. He received the B.A. degree in physics from Harvard University in 1966 and the M.S. and Ph.D. degrees from the University of Pennsylvania in 1967 and 1970, respectively. After postdoctoral research at Cambridge University, he joined the faculty of the James Franck Institute and the Department of Physics at the University of Chicago, 1973–1980. In 1980 he became professor at NORDITA, the Nordic Institute for Theoretical Physics, in Copenhagen. He has worked in condensed matter and statistical physics and, more recently, in the theory of neural networks and neural computation. He also collaborates actively in experimental studies of visual information processing in the Laboratory of Neuropsychology at the National Institute of Mental Health, Bethesda, MD.

Anders Krogh was born in Denmark in 1959. He received the Ph.D. degree in physics from the University of Copenhagen in 1991 for theoretical work on neural networks.

Although still interested in the foundations of neural networks, his main interest is now in applications of machine learning and statistics to problems in molecular biology. Currently he is research assistant professor at the Center for Biological Sequence analysis, Technical University of Denmark.

Benny Lautrup is Professor of Theoretical Physics at the Niels Bohr Institute, University of Copenhagen, Denmark. He is Head of the Computational Neural Networks Center (CONNECT), the Niels Bohr Institute, Copenhagen, Denmark. His interests include quantum field theory, lattice gauge theories, spin glass systems, neural networks, and brain imaging analysis. He is the author of one book on neural networks.

He is Editor-in-Charge of the *International Journal of Neural Systems*, World Scientific Publishing Co., London, U.K.

Torsten Lehmann was born in Bagsværd, Denmark, in 1967. He received the M.Sc. degree in electrical engineering from the Technical University of Denmark, Lyngby, Denmark, in 1991, and the Ph.D. degree in 1995 for the work "hardware learning in analog VLSI neural networks." During his Ph.D. studies he was associated with CONNECT, the Computational Neural Network Center.

He spent two years as a Research Fellow funded by the European Union at the University of Edinburgh, Scotland, where he worked with biologically inspired neural systems using pulse stream techniques. Presently, he is an Assistant Professor in microelectronics at the Technical University of Denmark. His main research interests inlcude solid-state circuits and systems (analog and digital) and artificial neural networks.