

Optimal Algorithms for Adaptive Networks: Second Order Back Propagation, Second Order Direct Propagation, and Second Order Hebbian Learning

David B. Parker

925 Oak Lane #4, Menlo Park, CA, 94025

1 Introduction

An adaptive network (also known as an artificial neural network) consists of one or more adaptive cells that are connected together. The pattern of connections between the cells is called the architecture of the network. Many different types of adaptive cells and architectures are described in the literature. This paper describes a class of algorithms — called second order algorithms — that may be able to give any of these adaptive cells or architectures asymptotically optimal performance. These second order algorithms implement an efficient network approximation to a minimization technique called Newton's method.

We will proceed in two steps. First, we will derive an optimal algorithm. For several reasons, this optimal algorithm will be impossible to implement. From it, however, we can then derive the asymptotically optimal second order algorithms. There are many definitions of the word 'optimal', and any algorithm is optimal if you pick your definition carefully. The optimality criterion that we will use is both natural and not too narrow.

To derive the optimal and asymptotically optimal algorithms, we will make the following assumptions about whatever network we are working with:

1. There are m error signals, $\epsilon_1, \dots, \epsilon_m$, represented by the m dimensional vector $\mathbf{e}$.
2. Inside the network there are a total of n weights, $w_1, \dots, w_n$, represented by the n dimensional vector $\mathbf{w}$.
3. To simplify the math, we will assume that signals propagate instantaneously between the cells in the network, and that the error signals reflect the instantaneous performance of the network (this is known as supervised learning). Several people have noted that these are unrealistic assumptions for various kinds of learning tasks (see, for example, Barto [1985] or Sutton [1987], which also contain further references).

In a typical network, each cell contains a subset of the weights in the network. The output(s) of each cell may be any function(s) of its inputs and weights. These assumptions are weak enough, I believe, to allow the second order algorithms to apply to any type of network in the literature. They even apply to networks in which no explicit error signals are generated, such as networks consisting of the Hebbian cells that we will discuss later.

Before deriving the second order algorithms, it may be helpful to review some previous work in the field. One consideration in developing an algorithm for an adaptive network is that we want the algorithm to have nice global properties, but we must be able to implement it on local, neuron-like processors. One example of such an algorithm is known as backward error propagation (or BEP, or back propagation, or backprop). A back propagation network globally performs an approximation to a minimization algorithm

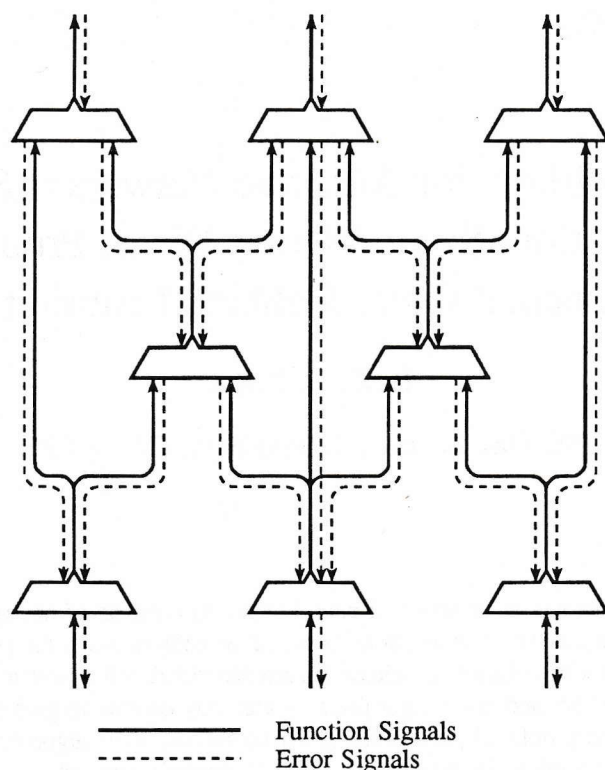


Figure 1: An example of a back propagation network.

known as steepest descent or gradient descent. The weights in the network are changing according to some variation of the algorithm:

$$\frac{\partial \mathbf{w}}{\partial t} = -2\mu \frac{\partial \epsilon^T}{\partial \mathbf{w}} \mathbf{e}.$$

The scalar parameter μ controls the convergence of the algorithm, and the superscript T denotes the vector transpose. If ϵ_c is the total error signal for an individual cell in the network, then the weights, $\mathbf{w}_c$, in that cell change according to some version of the algorithm

$$\frac{\partial \mathbf{w}_c}{\partial t} = -2\mu \frac{\partial \epsilon_c}{\partial \mathbf{w}_c} \epsilon_c. \quad (1)$$

This single-cell algorithm is known as the Widrow-Hoff, or LMS, algorithm (see Widrow and Stearns [1985] for further references). Figure 1 is an example of a back propagation network. Input signals come in at the bottom of the network, propagate forward through the network, and come out at the top as output signals. Error signals come in at the top and propagate backwards. I call the forward going signals function signals because they presumably perform some useful function when they come out as output signals, and because each function signal is calculated as some function of the inputs and weights of the cell it comes out of. Each cell in the network not only calculates a forward going function signal, but also calculates the derivatives $\partial \epsilon^T / \partial \mathbf{w}$ needed for the backward propagating error signals. Cells which are not in the top layer are referred to as hidden cells.

Back propagation, including several extensions, was developed by Werbos [1974,1982,1983,1986,1987]. Due to the state of artificial neural network research at the time, Werbos' early work failed to receive the attention it deserved; I have only recently learned of it. Werbos [1987] contains pointers to some of his other publications. He applies back propagation to tasks such as political forecasting, energy modeling, and modeling parts of the brain. Werbos makes the important point that any algorithm used to update the weights

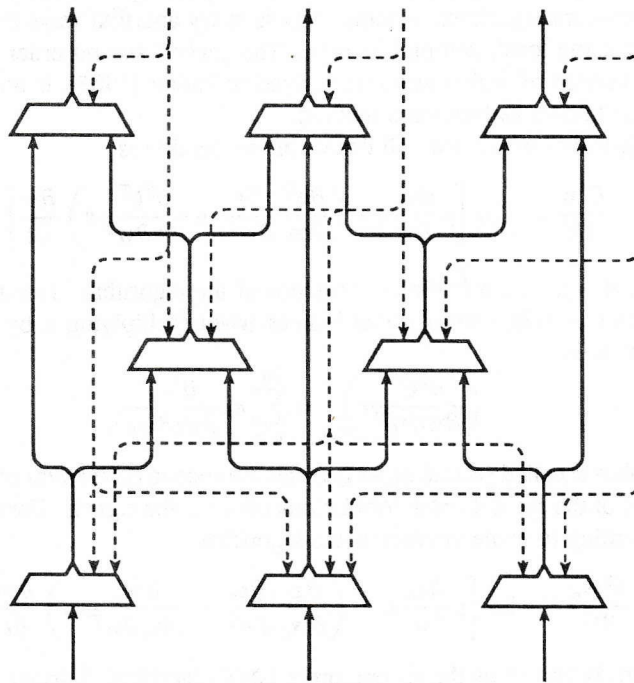


Figure 2: An example of a direct propagation network.

in a supervised learning network is only one of three components needed to make an intelligent system. I regret that I didn't learn of his work in time to comment on it more fully. I independently rederived back propagation in Parker [1982].

One problem with back propagation, as implemented in Figure 1, is that it doesn't seem neurobiologically plausible. Information propagates in two directions on the connections between the cells — function signals propagate forward and errors signals propagate backward — but the axons of real neurons conduct signals in only one direction. One way to overcome this difficulty is to estimate the derivatives $\partial e^T / \partial w$, rather than calculate them exactly. This was first done in Parker [1985] using separate function and error cells. The function cells propagated the function signals, and the error cells estimated the appropriate derivatives for the error signals. A rearrangement of the calculations leads to the architecture of Figure 2. I call this algorithm — in which the derivatives are estimated instead of calculated — direct propagation, since the error signals can propagate directly to the hidden cells. An architecture which looked similar, but which didn't include estimating the derivatives, was examined in the 1960's (Widrow [1986] — unfortunately I couldn't get the source reference in time for this paper).

This discussion of back propagation has been necessarily brief. Rumelhart, Hinton, and Williams [1985], who deserve much of the credit for bringing back propagation to people's attention, apply it to several classical neural network problems and describe various implementation details.

2 Derivation of the Second Order Algorithms

From here on out we will refer to the back propagation algorithm described in the last section as first order back propagation, to contrast it with the second order back propagation algorithm which we will derive in this section. Similarly, we will refer to the LMS algorithm of equation (1) as first order LMS. Nearly everyone who has worked with first order back propagation has noticed that it is slow (among other problems), and

has developed methods to speed it up [9,3,7]. One way to speed up first order back propagation is to try variants of the steepest descent algorithm. Another way is to try and find ways to implement more powerful minimization algorithms using local, neuron-like cells. The general second order algorithm which we derive in this section, a crude version of which was first derived in Parker [1985], is an efficient approximation to a minimization technique known as Newton's method.

The second order algorithm which we will derive in this section is

$$\frac{\partial^2 \mathbf{w}}{\partial t^2} = -2\alpha \left[\mu \frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \mathbf{e} + \left(\frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \frac{\partial \mathbf{e}}{\partial \mathbf{w}^T} + \frac{\partial^2 \mathbf{e}^T}{\partial \mathbf{w} \partial \mathbf{w}^T} \mathbf{e} \right) \frac{\partial \mathbf{w}}{\partial t} \right]. \quad (2)$$

The scalar parameter α , like μ , controls the convergence of the algorithm. The quantity $\partial^2 \mathbf{e}^T / \partial \mathbf{w} \partial \mathbf{w}^T$ is a tensor-like object, so one has to be careful about indices when multiplying it by $\mathbf{e}$. The result is an n -by- n matrix, whose ij th element is:

$$\left(\frac{\partial^2 \mathbf{e}^T}{\partial \mathbf{w} \partial \mathbf{w}^T} \mathbf{e} \right)_{ij} = \sum_{k=1}^m \epsilon_k \frac{\partial^2 \epsilon_k}{\partial w_i \partial w_j}.$$

The second order algorithm is called second order because the second derivatives of the weights are calculated, and because it makes use of the second order correlations between the signals. For a single cell in the network, the weights change according to some version of the algorithm

$$\frac{\partial^2 \mathbf{w}_c}{\partial t^2} = -2\alpha \left[\mu \frac{\partial \epsilon_c}{\partial \mathbf{w}_c} \epsilon_c + \left(\frac{\partial \epsilon_c}{\partial \mathbf{w}_c} \frac{\partial \epsilon_c}{\partial \mathbf{w}^T} + \frac{\partial^2 \epsilon_c}{\partial \mathbf{w}_c \partial \mathbf{w}^T} \epsilon_c \right) \frac{\partial \mathbf{w}}{\partial t} \right].$$

This single-cell algorithm is known as the second order LMS algorithm. It looks more complicated than the equivalent first order LMS algorithm in equation (1), but for many functions — such as a linear function or a linear function with a non-linear sigmoid output — several terms cancel or merge, so that the second order algorithm is really only slightly more complex. For example, the linear second order LMS algorithm for a single linear cell whose inputs are the vector $\mathbf{x}$, whose output is $f = \mathbf{x}^T \mathbf{w}$, and which is receiving an error signal ϵ , is

$$\frac{\partial^2 \mathbf{w}}{\partial t^2} = 2\alpha [\mu \epsilon - \mathbf{x}^T \frac{\partial \mathbf{w}}{\partial t}] \mathbf{x}. \quad (3)$$

We can write a little computer program-like set of statements that implements the linear second order LMS algorithm. The vector $\mathbf{w}_k$ is the set of weights in the cell at time step k , $\mathbf{x}_k$ is the vector of inputs to the cell, $\Delta \mathbf{w}_k$ is the array of increments to the weights, d_k is the desired function for the cell, f_k is the output of the cell, g_k is a temporary variable, ϵ_k is the error signal at time step k , l_1 and l_2 are leakage rates for the two integrations, and Δt is the amount of time between time step k and time step $k+1$ (Δt is often taken to be 1).

$$\begin{aligned} f_k &:= \mathbf{x}_k^T \mathbf{w}_k \\ g_k &:= \mathbf{x}_k^T \Delta \mathbf{w}_k \\ \epsilon_k &:= d_k - f_k \\ \Delta \mathbf{w}_{k+1} &:= (1 - l_1 \Delta t) \Delta \mathbf{w}_k + 2\alpha \Delta t (\mu \Delta t \epsilon_k - g_k) \mathbf{x}_k \\ \mathbf{w}_{k+1} &:= (1 - l_2 \Delta t) \mathbf{w}_k + \Delta t \Delta \mathbf{w}_{k+1} \end{aligned}$$

Only about twice as many arithmetic operations are needed to implement the linear second order LMS algorithm as the linear first order LMS algorithm. The linear second order LMS algorithm may be important in many applications such as adaptive equalization and adaptive filtering, so its properties are extensively analyzed in [5].

We begin deriving the optimal algorithm by requiring that the network try to minimize the average squared error. We could have the network try to minimize other norms, such as the average absolute value of the errors, but the average squared error is mathematically convenient. The total squared error at any instant

is given by $\mathbf{e}^T \mathbf{e}$. The average squared error can be expressed using an exponentially weighted average with time constant μ (other types of averages can be used, too). Using a little bar over a quantity to indicate that it is an average, and where e is the base of the natural logarithms (2.718...), we can write the average squared error as

$$\overline{\mathbf{e}^T \mathbf{e}} = \mu \int_{-\infty}^t \mathbf{e}^T \mathbf{e} e^{-\mu[t-\tau]} d\tau.$$

If we temporarily hold t fixed, we know from calculus that when the weights are at the values that minimize the average squared error, then the derivative of the average squared error with respect to the weights will be zero:

$$\frac{\partial \overline{\mathbf{e}^T \mathbf{e}}}{\partial \mathbf{w}} = 0 = 2\mu \int_{-\infty}^t \frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \mathbf{e} e^{-\mu[t-\tau]} d\tau. \quad (4)$$

We can now define our optimality criterion in terms of the optimal path through weight-space that the weights should follow as the network is trained. As the network is trained, more error information will be obtained. This additional error information will usually shift the minimum of the error surface. The optimal path for the weights follows that shifting minimum, i.e. the weights change so that equation (4) is true at every instant of time. To get an implicit formula for the optimal path, we now let t become a variable and take the derivative of equation (4) with respect to time,

$$\frac{\partial}{\partial t} \frac{\partial \overline{\mathbf{e}^T \mathbf{e}}}{\partial \mathbf{w}} = 0 = 2\mu \frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \mathbf{e} + \left[2\mu \int_{-\infty}^t \left(\frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \frac{\partial \mathbf{e}}{\partial \mathbf{w}^T} + \frac{\partial^2 \mathbf{e}^T}{\partial \mathbf{w} \partial \mathbf{w}^T} \mathbf{e} \right) e^{-\mu[t-\tau]} d\tau \right] \frac{\partial \mathbf{w}}{\partial t},$$

which we can solve for $\partial \mathbf{w} / \partial t$:

$$\frac{\partial \mathbf{w}}{\partial t} = - \left[2\mu \int_{-\infty}^t \left(\frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \frac{\partial \mathbf{e}}{\partial \mathbf{w}^T} + \frac{\partial^2 \mathbf{e}^T}{\partial \mathbf{w} \partial \mathbf{w}^T} \mathbf{e} \right) e^{-\mu[t-\tau]} d\tau \right]^{-1} 2\mu \frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \mathbf{e}. \quad (5)$$

This differential equation is the optimal algorithm.

This algorithm is impractical for two reasons. The first is that inverting the matrix in the square brackets is too complex an operation (not to mention that the matrix might not even be invertible). One way to overcome this first difficulty is to use an iterative method to solve for $\partial \mathbf{w} / \partial t$. We can move the matrix back to the left hand side of the equation, then multiply both sides by $-\alpha \Delta t$, where α is a parameter controlling the convergence of the iterative method, and Δt is a small amount of time that we will later let go to zero:

$$2\alpha \Delta t \left[\mu \int_{-\infty}^t \left(\frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \frac{\partial \mathbf{e}}{\partial \mathbf{w}^T} + \frac{\partial^2 \mathbf{e}^T}{\partial \mathbf{w} \partial \mathbf{w}^T} \mathbf{e} \right) e^{-\mu[t-\tau]} d\tau \right] \frac{\partial \mathbf{w}}{\partial t} = -2\alpha \Delta t \mu \frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \mathbf{e}.$$

The next step in developing the iterative method is to split the scaled matrix on the left hand side by adding and subtracting another matrix. A convenient choice is the identity matrix, I (other matrices may be used, too):

$$\left[2\alpha \Delta t \left[\mu \int_{-\infty}^t \left(\frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \frac{\partial \mathbf{e}}{\partial \mathbf{w}^T} + \frac{\partial^2 \mathbf{e}^T}{\partial \mathbf{w} \partial \mathbf{w}^T} \mathbf{e} \right) e^{-\mu[t-\tau]} d\tau \right] + I - I \right] \frac{\partial \mathbf{w}}{\partial t} = -2\alpha \Delta t \mu \frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \mathbf{e}.$$

Next, we move most of the left hand side back to the right hand side:

$$\frac{\partial \mathbf{w}}{\partial t} = \frac{\partial \mathbf{w}}{\partial t} - 2\alpha \Delta t \mu \frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \mathbf{e} - 2\alpha \Delta t \left[\mu \int_{-\infty}^t \left(\frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \frac{\partial \mathbf{e}}{\partial \mathbf{w}^T} + \frac{\partial^2 \mathbf{e}^T}{\partial \mathbf{w} \partial \mathbf{w}^T} \mathbf{e} \right) e^{-\mu[t-\tau]} d\tau \right] \frac{\partial \mathbf{w}}{\partial t}.$$

This equation looks rather odd because $\partial \mathbf{w} / \partial t$ appears alone on both sides. However, we can now perform the crucial step of the iterative algorithm, which is to use our estimate of the value of $\partial \mathbf{w} / \partial t$ at time t as the basis for an improved estimate at time $t + \Delta t$:

$$\left. \frac{\partial \mathbf{w}}{\partial t} \right|_{t+\Delta t} = \left. \frac{\partial \mathbf{w}}{\partial t} \right|_t - 2\alpha \Delta t \mu \frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \mathbf{e} - 2\alpha \Delta t \left[\mu \int_{-\infty}^t \left(\frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \frac{\partial \mathbf{e}}{\partial \mathbf{w}^T} + \frac{\partial^2 \mathbf{e}^T}{\partial \mathbf{w} \partial \mathbf{w}^T} \mathbf{e} \right) e^{-\mu[t-\tau]} d\tau \right] \left. \frac{\partial \mathbf{w}}{\partial t} \right|_t.$$

We may move the lone value of $\partial \mathbf{w} / \partial t|_t$ from the right hand side to the left hand side, and then divide both sides by Δt :

$$\frac{\frac{\partial \mathbf{w}}{\partial t}|_{t+\Delta t} - \frac{\partial \mathbf{w}}{\partial t}|_t}{\Delta t} = -2\alpha\mu \frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \mathbf{e} - 2\alpha \left[\mu \int_{-\infty}^t \left(\frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \frac{\partial \mathbf{e}}{\partial \mathbf{w}^T} + \frac{\partial^2 \mathbf{e}^T}{\partial \mathbf{w} \partial \mathbf{w}^T} \mathbf{e} \right) e^{-\mu[t-\tau]} d\tau \right] \frac{\partial \mathbf{w}}{\partial t}|_t.$$

If we let Δt go to zero, this equation approaches the second order differential equation:

$$\frac{\partial^2 \mathbf{w}}{\partial t^2} = -2\alpha\mu \frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \mathbf{e} - 2\alpha \left[\mu \int_{-\infty}^t \left(\frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \frac{\partial \mathbf{e}}{\partial \mathbf{w}^T} + \frac{\partial^2 \mathbf{e}^T}{\partial \mathbf{w} \partial \mathbf{w}^T} \mathbf{e} \right) e^{-\mu[t-\tau]} d\tau \right] \frac{\partial \mathbf{w}}{\partial t}.$$

We have now overcome our first difficulty by removing the need to invert the matrix in the square brackets. Not much has been lost in the process, either. The path followed by the weights will be arbitrarily close to the optimal path if the circuitry to update the weights is sufficiently faster than the maximum rate of change of the other signals in the network.

Having removed the need to invert the matrix in the square brackets in equation (5), the second difficulty now presents itself. We need to calculate the values of the elements of the matrix. These elements depend on what the behavior of the cell would have been at some point in the past if the current values of the weights had been used, so in general we cannot calculate them without going back in time and redoing history (a special case is if the adaptive network is completely linear, in which case we can save enough information about the past in order to effectively repeat it). So, we must estimate the matrix. Since the matrix is an exponentially weighted average, a convenient estimate is the current value of the matrix:

$$\frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \frac{\partial \mathbf{e}}{\partial \mathbf{w}^T} + \frac{\partial^2 \mathbf{e}^T}{\partial \mathbf{w} \partial \mathbf{w}^T} \mathbf{e} = \text{estimated value of } \mu \int_{-\infty}^t \left(\frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \frac{\partial \mathbf{e}}{\partial \mathbf{w}^T} + \frac{\partial^2 \mathbf{e}^T}{\partial \mathbf{w} \partial \mathbf{w}^T} \mathbf{e} \right) e^{-\mu[t-\tau]} d\tau. \quad (6)$$

This estimate is unbiased because the exponentially weighted average of the estimates is identical to the average we are estimating.

Now that we have solved both computational difficulties with the optimal algorithm (5), we can replace the matrix in square brackets by its instantaneous estimate to get the asymptotically optimal algorithm — which is the second order algorithm introduced in equation (2):

$$\frac{\partial^2 \mathbf{w}}{\partial t^2} = -2\alpha \left[\mu \frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \mathbf{e} + \left(\frac{\partial \mathbf{e}^T}{\partial \mathbf{w}} \frac{\partial \mathbf{e}}{\partial \mathbf{w}^T} + \frac{\partial^2 \mathbf{e}^T}{\partial \mathbf{w} \partial \mathbf{w}^T} \mathbf{e} \right) \frac{\partial \mathbf{w}}{\partial t} \right].$$

Other estimates of the matrix lead to different versions of the algorithm.

This second order algorithm is optimal in that no algorithm which uses an estimate of the matrix in equation (6) (and all realistic algorithms have to) can on average follow the optimal path for the weights better than second order algorithm. The weights in the second order algorithm will jiggle along the optimal path, following it on average.

This second order algorithm can be implemented in different ways. Using the network architecture of Figure 1, the algorithm becomes second order back propagation; using the architecture of Figure 2 it becomes second order direct propagation.

Implementing the algorithm in these architectures is not as straightforward as it looks, however. One of the problems is that each cell in the network needs information about the derivatives of its input signals. Taking the derivatives of signals is usually a highly unstable process. Fortunately, Nature provides a solution. Neurons in our brain seem to transmit information as the approximate frequency of a series of action potentials. Since the pulses are integrated to determine their frequency, each pulse is approximately the derivative of the frequency. The same trick — of transmitting the derivative of the signal of interest, instead of the signal itself — can be used to implement the second order algorithms. However, the algorithms in equations (3) and (9) don't take account of the derivatives of the input signals, so probably aren't the best algorithms to use in a multi-layered network. The extended version of this paper will describe more suitable algorithms.

3 Second Order Hebbian Learning

For a Hebbian cell, there is no error signal. For example, the formula for the changes to the weights in a simple linear Hebbian cell whose output is $f = \mathbf{x}^T \mathbf{w}$ can be written as

$$\frac{\partial \mathbf{w}}{\partial t} = 2\mu f \mathbf{x}. \quad (7)$$

At first sight it would seem that we can not apply the results we have derived because they all involve an error signal. However, we can note that for a linear cell using the first order LMS algorithm in equation (1) the weight change would be

$$\frac{\partial \mathbf{w}}{\partial t} = 2\mu \epsilon \mathbf{x}. \quad (8)$$

Comparing equations (7) and (8), it is clear that the Hebbian cell is equivalent to an LMS cell that is using its own output as an error signal. Thus we can use equation (3) to generate the corresponding second order Hebbian learning algorithm for a linear Hebbian cell:

$$\frac{\partial^2 \mathbf{w}}{\partial t^2} = 2\alpha [\mu f - \mathbf{x}^T \frac{\partial \mathbf{w}}{\partial t}] \mathbf{x}. \quad (9)$$

This same idea, of a cell generating its own error signal, can be used to create second order algorithms for various types of cells and networks that don't appear to have any error signals.

4 Conclusion

The second order algorithms derived herein give promise of being able to make any adaptive network asymptotically optimal, at least under the criterion presented in this paper. These optimal algorithms implement an efficient network approximation to the minimization technique called Newton's method. Space constraints did not permit the inclusion of a discussion of implementation details, simulation results, and comparisons with other algorithms. I will include them in an extended version of this paper that I will submit to the new journal being announced at this conference. Also, space did not permit the inclusion of proofs of optimality. These will be found in Parker [1987], in my slowly-but-surely upcoming book [6], and in future papers.

5 Acknowledgements

I would like to thank Yann LeCun for insightful discussions, Dan Greenwood for suggestions on cleaning up the math, and Bernard Widrow for his advice and encouragement. I would especially like to thank Paul Werbos for our discussions.

References

- [1] Andrew G. Barto, "Learning by Statistical Cooperation of Self-interested Neuron-like Computing Elements", *Human Neurobiology*, Vol 4, pp.229-256, Springer-Verlag, 1985
- [2] David B. Parker, "Learning-Logic", Invention Report, S81-64, File 1, Office of Technology Licensing, Stanford University, October, 1982 (copies also available from author)
- [3] —, "Learning-Logic", TR-47, Center for Computational Research in Economics and Management Science, MIT, April 1985 (copies also available from author)
- [4] —, "A Comparison of Algorithms for Neuron-Like Cells", *AIP Conference Proceedings 151 — Neural Networks for Computing*, American Institute of Physics, 1986

- [5] —, "Convergence Properties of the Linear Second Order LMS Algorithm", submitted to the IEEE Transactions on Acoustics, Speech, and Signal Processing, 1987
- [6] —, *Adaptive Networks and Their Applications*, textbook in preparation
- [7] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams, "Learning Internal Representations by Error Propagation", ICS Report 8506, Institute for Cognitive Science, University of California, San Diego, Sept 1985 (this also appears as a chapter in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, MIT Press, 1986)
- [8] Richard S. Sutton, "Learning to Predict by the Methods of Temporal Differences", Technical Report TR87-509.1, GTE Laboratories Inc., Waltham, MA, 02254, Jan 1987
- [9] Paul J. Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences", thesis in applied mathematics, Harvard University, Aug 1974
- [10] —, "Applications of Advances in Nonlinear Sensitivity Analysis", in *Lecture Notes in Control and Information Sciences, Vol. 38: System Modeling and Optimization, Proceedings of the 10th IFIP Conference*, A. V. Balakrishnan, M. Thoma, R. F. Drenick, and F. Kozin, eds., Springer-Verlag, 1982
- [11] —, "Solving and Optimizing Complex Systems: Lessons from the EIA Long-Term Energy Model", in *Studies in Management Science and Systems, Vol 9: Energy Models and Studies*, B. V. Dean and B. Lev, eds., North-Holland Publishing Company, 1983
- [12] —, "Generalized Information Requirements of Intelligent Decision-making Systems", in *SUGI 11 Proc.*, Cary, NC: SAS Institute, 1986 (an amplified and easier to read version is available from the author)
- [13] —, "Building and Understanding Adaptive Systems: A Statistical/Numerical Approach to Factory Automation and Brain Research", IEEE Trans. Syst. Man & Cyber., Vol. SMC-17, No. 1, Jan/Feb 1987
- [14] Bernard Widrow and Samuel D. Stearns, *Adaptive Signal Processing*, Prentice-Hall, 1985
- [15] Bernard Widrow, personal communication, Sept 1986