Perceptrons, Adalines, and Backpropagation

Bernard Widrow and Michael A. Lehr

Introduction

The field of neural networks has enjoyed major advances since 1960, a year which saw the introduction of two of the earliest feedforward neural network algorithms: the perceptron rule (Rosenblatt, 1962) and the LMS algorithm (Widrow and Hoff, 1960). Around 1961, Widrow and his students devised Madaline Rule I (MRI), the earliest learning rule for feedforward networks with multiple adaptive elements. The major extension of the feedforward neural network beyond Madaline I took place in 1971 when Paul Werbos developed a backpropagation algorithm for training multilayer neural networks. He first published his findings in 1974 in his doctoral dissertation (see BACKPROPAGATION: BASICS AND NEW DEVELOPMENTS). Werbos's work remained almost unknown in the scientific community until 1986, when Rumelhart, Hinton, and Williams (1986) rediscovered the technique and, within a clear framework, succeeded in making the method widely known.

The development of backpropagation has made it possible to attack problems requiring neural networks with high degrees of nonlinearity and precision (Widrow and Lehr, 1990; Widrow, Rumelhart, and Lehr, 1994). Backpropagation networks with fewer than 150 neural elements have been successfully applied to vehicular control simulations, speech generation, and undersea mine detection. Small networks have also been used successfully in airport explosive detection, expert systems, and scores of other applications. Furthermore, efforts to develop parallel neural network hardware are advancing rapidly, and these systems are now becoming available for attacking more difficult problems such as continuous speech recognition.

The networks used to solve the above applications varied widely in size and topology. A basic component of nearly all neural networks, however, is the adaptive linear combiner.

The Adaptive Linear Combiner

The adaptive linear combiner has as output a linear combination of its inputs. In a digital implementation, this element receives at time k an input signal vector or input pattern vector $\mathbf{X}_k = [x_0, x_{1_k}, x_{2_k}, \ldots, x_{n_k}]^T$, and a desired response d_k, a special input used to effect learning. The components of the input vector are weighted by a set of coefficients, the weight vector $\mathbf{W}_k = [w_{0_k}, w_{1_k}, w_{2_k}, \ldots, w_{n_k}]^T$. The sum of the weighted inputs is then computed, producing a linear output, the inner product $s_k = \mathbf{X}_k^T\mathbf{W}_k$. The components of $\mathbf{X}_k$ may be either continuous analog values or binary values. The weights are essentially continuously variable and can take on negative as well as positive values.

During the training process, input patterns and corresponding desired responses are presented to the linear combiner. An adaptation algorithm automatically adjusts the weights so the output responses to the input patterns will be as close as possible to their respective desired responses. In signal processing applications, the most popular method for adapting the weights is the simple LMS (least mean square) algorithm (Widrow and Hoff, 1960), often called the Widrow-Hoff Delta Rule (Rumelhart, Hinton, and Williams, 1986). This algorithm minimizes the sum of squares of the linear errors over the training set. The linear error ε_k is defined to be the difference between the desired response d_k and the linear output s_k during presentation k. Having this error signal is necessary for adapting the weights. Both the LMS rule and Rosenblatt's perceptron rule will be detailed in later sections.

An important element used in many neural networks is the "ADAptive LInear NEuron," or *adaline* (Widrow and Hoff, 1960). In the neural network literature, such elements are often referred to as *adaptive neurons*. The adaline is an adaptive threshold logic element. It consists of an adaptive linear combiner cascaded with a hard-limiting quantizer which is used to produce a binary ± 1 output, $y_k = \text{sgn}(s_k)$. A bias weight, or *threshold*, w_{0_k}, which is connected to a constant input, $x_0 = +1$, effectively controls the threshold level of the quantizer. Such an element may be seen as a McCulloch-Pitts neuron augmented with a learning rule for adjusting its weights.

In single-element neural networks, the weights are often trained to classify binary patterns using binary desired responses. Once training is complete, the responses of the trained element can be tested by applying various input patterns. If the adaline responds correctly with high probability to input patterns that were not included in the training set, it is said that generalization has taken place. Learning and generalization are among the most useful attributes of adalines and neural networks.

With n binary inputs and one binary output, a single adaline is capable of implementing certain logic functions. There are 2^n possible input patterns. A general logic implementation would be capable of classifying each pattern as either $+1$ or -1, in accordance with the desired response. Thus, there are 2^{2^n} possible logic functions connecting n inputs to a single binary output. A single adaline is capable of realizing only a small subset of these functions, known as the linearly separable logic functions or threshold logic functions. These are the set of logic functions that can be obtained with all possible weight variations. With two inputs, a single adaline can realize 14 of the 16 possible binary logic functions. The two it cannot learn are exclusive OR and exclusive NOR functions. With many inputs, however, only a small fraction of all possible logic functions are realizable, i.e., linearly separable. Combinations of elements or networks of elements can be used to realize functions which are not linearly separable.

Nonlinear Neural Networks

One of the earliest trainable layered neural networks with multiple adaptive elements was the *Madaline I* structure of Widrow and Hoff. In the early 1960s, a 1000-weight Madaline I was built out of hardware and used in pattern recognition research (Widrow and Lehr, 1990). The weights in this machine were memistors, electrically variable resistors developed by Widrow and Hoff which are adjusted by electroplating a resistive link in a sealed cell containing copper sulfate and sulfuric acid.

Madaline I was configured in the following way. Retinal inputs were connected to a layer of adaptive adaline elements, the outputs of which were connected to a fixed logic device that generated the system output. Methods for adapting such systems were developed at that time. An example of this kind of network is shown in Figure 1. Two adalines are connected to an AND logic device to provide an output. With weights suitably chosen, the separating boundary in pattern space for the system can implement any of the 16 two-input binary logic functions, including the exclusive OR and exclusive NOR functions.

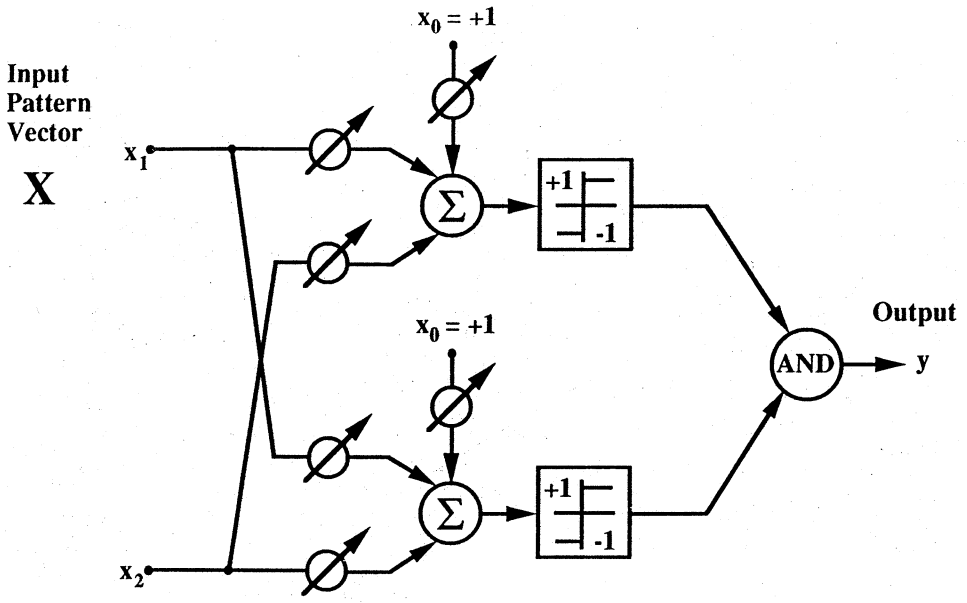

Figure 1. A two-adaline form of madaline.

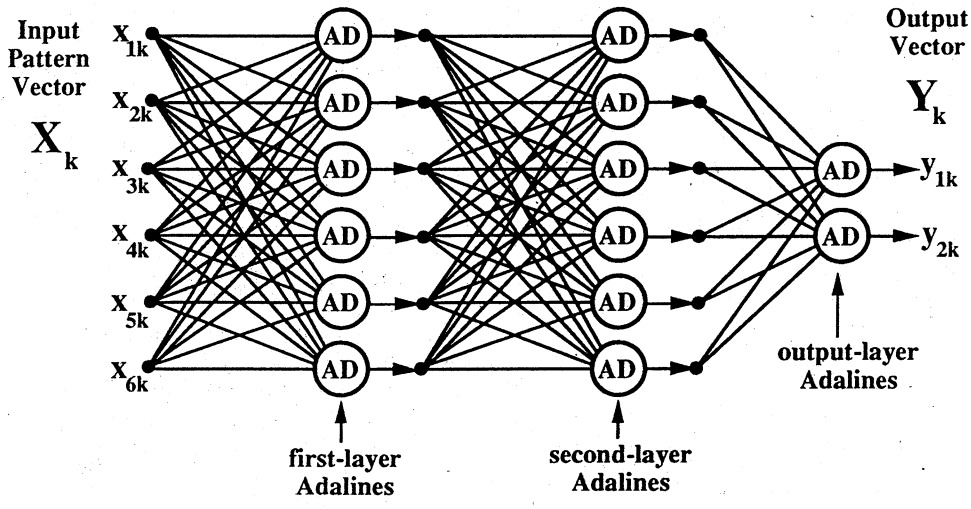

Figure 2. A three-layer adaptive neural network.

Madalines were constructed with many more inputs, with many more adaline elements in the first layer, and with various fixed logic devices such as AND, OR, and majority vote-taker elements in the second layer. Those three functions are all threshold logic functions.

Multilayer Networks

The madaline networks of the 1960s had an adaptive first layer and a fixed threshold function in the second (output) layer (Widrow and Lehr, 1990). The feedforward neural networks of today often have many layers, all of which are usually adaptive. The backpropagation networks of Rumelhart et al. (1986) are perhaps the best-known examples of multilayer networks. A three-layer feedforward adaptive network is illustrated in Figure 2. It is "fully connected" in the sense that each adaline receives inputs from every output in the preceding layer.

During training, the responses of the output elements in the network are compared with a corresponding set of desired responses. Error signals associated with the elements of the output layer are thus readily computed, so adaptation of the output layer is straightforward. The fundamental difficulty associated with adapting a layered network lies in obtaining *error signals* for hidden layer adalines, that is, for adalines in layers other than the output layer. The backpropagation algorithm provides a method for establishing these error signals.

Learning Algorithms

The iterative algorithms described here are all designed in accord with the *Principle of Minimal Disturbance: Adapt to reduce the output error for the current training pattern, with minimal disturbance to responses already learned.* Unless this principle is practiced, it is difficult to simultaneously store the required pattern responses. The minimal disturbance principle is intuitive. It was the motivating idea that led to the discovery of the LMS algorithm and the madaline rules. In fact, the LMS algorithm had existed for several months as an error reduction rule before it was discovered that the algorithm uses an instantaneous gradient to follow the path of steepest descent and minimizes the mean square error of the training set. It was then given the name LMS (least mean square) algorithm.

The LMS Algorithm

The objective of adaptation for a feedforward neural network is usually to reduce the error between the desired response and the network's actual response. The most common error function is the mean square error (MSE), averaged over the training set. The most popular approaches to mean-square-error reduction in both single-element and multielement networks are based on the method of gradient descent.

Adaptation of a network by gradient descent starts with an arbitrary initial value $\mathbf{W}_0$ for the system's weight vector. The gradient of the mean-square-error function is measured and the weight vector is altered in the direction opposite to the measured gradient. This procedure is repeated, causing the MSE to be successively reduced on average and causing the weight vector to approach a locally optimal value.

The method of gradient descent can be described by the relation

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu(-\nabla_k) \tag{1}$$

where μ is a parameter that controls stability and rate of convergence and ∇_k is the value of the gradient at a point on the MSE surface corresponding to $\mathbf{W} = \mathbf{W}_k$.

The LMS algorithm works by performing approximate steepest descent on the mean-square-error surface in weight space. This surface is a quadratic function of the weights and is therefore convex and has a unique (global) minimum. An instantaneous gradient based on the square of the instantaneous error is

$$\hat{\nabla}_k = \frac{\partial \varepsilon_k^2}{\partial \mathbf{W}_k} = \begin{Bmatrix} \frac{\partial \varepsilon_k^2}{\partial w_{0k}} \\ \vdots \\ \frac{\partial \varepsilon_k^2}{\partial w_{nk}} \end{Bmatrix} \tag{2}$$

LMS works by using this crude gradient estimate in place of the true gradient ∇_k. Making this replacement into Equation 1 yields

$$\mathbf{W}_{k+1} = \mathbf{W}_k = \mu(-\hat{\nabla}_k) = \mathbf{W}_k - \mu \frac{\partial \varepsilon_k^2}{\partial \mathbf{W}_k} \tag{3}$$

The instantaneous gradient is used because (a) it is an unbiased estimate of the true gradient (Widrow and Stearns, 1985), and (b) it is easily computed from single data samples. The true gradient is generally difficult to obtain. Computing it would involve averaging the instantaneous gradients associated with all patterns in the training set. This is usually impractical and almost always inefficient.

The present error or *linear* error ε_k is defined to be the difference between the desired response d_k and the linear output $s_k = \mathbf{W}_k^T\mathbf{X}_k$ before adaptation:

$$\varepsilon_k \triangleq d_k - \mathbf{W}_k^T\mathbf{X}_k \tag{4}$$

Performing the differentiation in Equation 3 and replacing the linear error by the definition in Equation 4 gives

$$\mathbf{W}_{k+1} = \mathbf{W}_k - 2\mu\varepsilon_k \frac{\partial(d_k - \mathbf{W}_k^T\mathbf{X}_k)}{\partial \mathbf{W}_k} \tag{5}$$

Noting that d_k and $\mathbf{X}_k$ are independent of $\mathbf{W}_k$ yields

$$\mathbf{W}_{k+1} = \mathbf{W}_k + 2\mu\varepsilon_k\mathbf{X}_k \tag{6}$$

This is the LMS algorithm. The learning constant μ determines stability and convergence rate (Widrow and Stearns, 1985).

The Perceptron Learning Rule

The Rosenblatt α-perceptron (Rosenblatt, 1962), diagrammed in Figure 3, processed input patterns with a first layer of sparse, randomly connected, fixed-logic devices. The outputs of the fixed first layer fed a second layer which consisted of a single adaptive linear threshold element. Other than the convention that its input signals and its output signal were $\{1, 0\}$ binary, and that no bias weight was included, this element was equivalent to the adaline element. The learning rule for the α-perceptron was very similar to LMS, but its behavior was in fact quite different.

Adapting with the perceptron rule makes use of the *quantizer error* $\tilde{\varepsilon}_k$, defined to be the difference between the desired re-

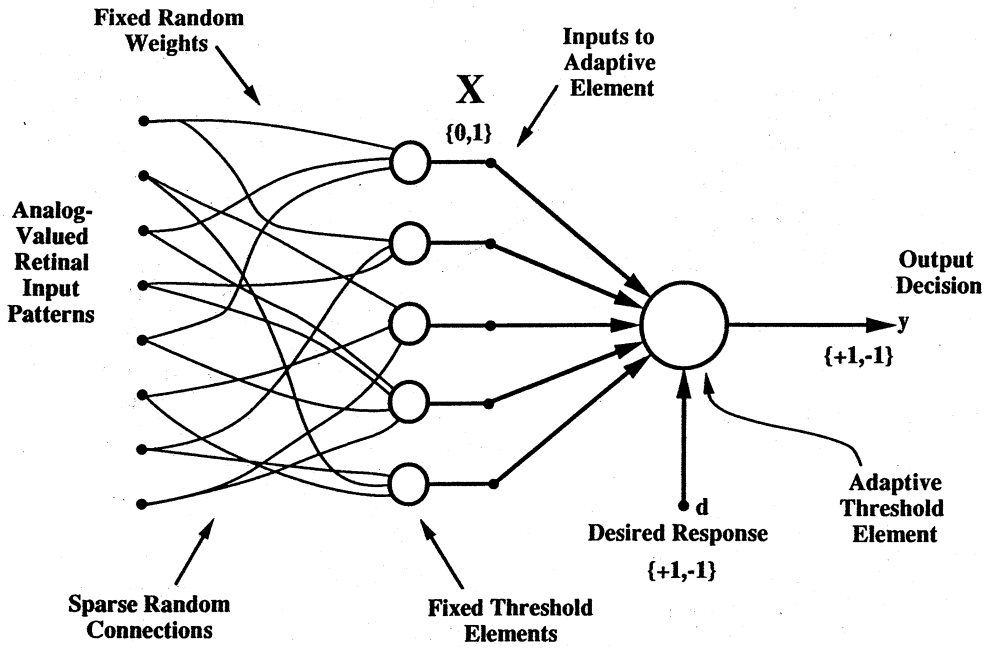

Figure 3. Rosenblatt's α-perceptron.

sponse and the output of the quantizer

$$\tilde{\tilde{\varepsilon}}_k \triangleq d_k - y_k \tag{7}$$

The perceptron rule, sometimes called the *perceptron convergence procedure*, does not adapt the weights if the output decision y_k is correct, i.e., if $\tilde{\tilde{\varepsilon}}_k = 0$. If the output decision disagrees with the binary desired response d_k, however, adaptation is effected by adding the input vector to the weight vector when the error $\tilde{\tilde{\varepsilon}}_k$ is positive, or subtracting the input vector from the weight vector when the error $\tilde{\tilde{\varepsilon}}_k$ is negative. Note that the quantizer error $\tilde{\tilde{\varepsilon}}_k$ is always equal to either $+1$, -1, or 0. Thus, the product of the input vector and the quantizer error $\tilde{\tilde{\varepsilon}}_k$ is added to the weight vector. The perceptron rule is identical to the LMS algorithm, except that with the perceptron rule, one-half of the quantizer error, $\tilde{\tilde{\varepsilon}}_k/4$, is used in place of the linear error ε_k of the LMS rule. The perceptron rule is nonlinear, in contrast to the LMS rule, which is linear. Nonetheless, it can be written in a form which is very similar to the LMS rule of Equation 6:

$$\mathbf{W}_{k+1} = \mathbf{W}_k + 2\mu \frac{\tilde{\tilde{\varepsilon}}_k}{2} \mathbf{X}_k \tag{8}$$

Rosenblatt normally set μ to one. In contrast to LMS, the choice of μ does not affect the stability of the perceptron algorithm, and it affects convergence time only if the initial weight vector is non-zero. Also, while LMS can be used with either analog or binary desired responses, Rosenblatt's rule can be used only with binary desired responses.

The perceptron rule stops adapting when the training patterns are correctly separated. There is no restraining force controlling the magnitude of the weights, however. The direction of the weight vector, not its magnitude, determines the decision function. The perceptron rule has been proven capable of separating any linearly separable set of training patterns (Rosenblatt, 1962; Nilsson, 1965). If the training patterns are not linearly separable, the perceptron algorithm goes on forever, and in most cases the weight vector gravitates toward zero. As a result, on problems which are not linearly separable, the perceptron often does not yield a low-error solution, even if one exists.

This behavior is very different from that of the LMS algorithm. Continued use of LMS does not lead to an unreasonable weight solution if the pattern set is not linearly separable. Nor, however, is this algorithm guaranteed to separate any linearly separable pattern set. LMS typically comes close to achieving such separation, but its objective is different, i.e., error reduction at the linear output of the adaptive element.

"Backpropagation" for the Sigmoid Adaline

A *sigmoid adaline* element incorporates a sigmoidal nonlinearity. The input-output relation of the sigmoid can be denoted by $y_k = \text{sgm}(s_k)$. A typical sigmoid function is the hyperbolic tangent

$$y_k = \tanh(s_k) = \left(\frac{1 - e^{-2s_k}}{1 + e^{-2s_k}}\right). \tag{9}$$

We shall adapt this adaline with the objective of minimizing the mean square of the *sigmoid error* $\tilde{\varepsilon}_k$, defined as

$$\tilde{\varepsilon}_k \triangleq d_k - y_k = d_k - \text{sgm}(s_k) \tag{10}$$

The method of gradient descent is used to adapt the weight vector. By following the same line of reasoning used to develop LMS, the instantaneous gradient estimate obtained during presentation of the kth input vector $\mathbf{X}_k$ can be found to be

$$\hat{\nabla}_k = \frac{\partial(\tilde{\varepsilon}_k)^2}{\partial \mathbf{W}_k} = 2\tilde{\varepsilon}_k \frac{\partial \tilde{\varepsilon}_k}{\partial \mathbf{W}_k} = -2\tilde{\varepsilon}_k \,\text{sgm}'(s_k)\mathbf{X}_k \tag{11}$$

Using this gradient estimate with the method of gradient descent provides a means for minimizing the mean square error even after the summed signal s_k goes through the nonlinear sigmoid. The algorithm is

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu(-\hat{\nabla}_k) = \mathbf{W}_k + 2\mu\delta_k\mathbf{X}_k \tag{12}$$

where δ_k denotes $\tilde{\varepsilon}_k \,\text{sgm}'(s_k)$. The algorithm of Equation 12 is the *backpropagation* algorithm for the single adaline element, though the backpropagation name only makes sense when the algorithm is utilized in a layered network, which will be studied later.

If the sigmoid is chosen to be the hyperbolic tangent function (Equation 9), then the derivative $\text{sgm}'(s_k)$ is given by

$$\begin{aligned}\text{sgm}'(s_k) &= \frac{\partial(\tanh(s_k))}{\partial s_k} \\ &= 1 - (\tanh(s_k))^2 = 1 - y_k^2\end{aligned} \tag{13}$$

Accordingly, Equation 12 becomes

$$\mathbf{W}_{k+1} = \mathbf{W}_k + 2\mu\tilde{\varepsilon}_k(1 - y_k^2)\mathbf{X}_k \tag{14}$$

The single sigmoid adaline trained by backpropagation shares some advantages with both the adaline trained by LMS and the perceptron trained by Rosenblatt's perceptron rule. If a pattern set is linearly separable, the objective function of the sigmoid adaline, the mean square error, is minimized only when the pattern set is separated. This is because, as the weights of the sigmoid adaline grow large, its response approximates that of a perceptron with weights in the same direction. The sigmoid adaline trained by backpropagation however, also shares the advantage of the adaline trained by LMS: it tends to give reasonable results even if the training set is not separable.

Backpropagation training of the sigmoid adaline does have one drawback, however. Unlike the linear error of the adaline, the output error of the sigmoid adaline is a nonlinear function of the weights. As a result, its mean square error surface is not quadratic, and may have local minima in addition to the optimal solution. Thus, unlike the perceptron rule, it cannot be guaranteed that backpropagation training of the sigmoid adaline will successfully separate a linearly separable training set. Nonetheless, the single sigmoid adaline performs admirably in many filtering and pattern classification applications. Its most important role, however, occurs in multilayer networks, to which we now turn.

Backpropagation for Networks

The backpropagation technique is a substantial generalization of the single sigmoid adaline case discussed in the previous section. When applied to multilayer feedforward networks, the backpropagation technique adjusts the weights in the direction opposite to the instantaneous gradient of the sum square error in weight space. Derivations of the algorithm are widely available in the literature (Rumelhart, Hinton, and Williams, 1986; Widrow and Lehr, 1990). Here we provide only a brief summary of the result.

The instantaneous sum square error ε_k^2 is the sum of the squares of the errors at each of the N_y outputs of the network. Thus

$$\varepsilon_k^2 = \sum_{i=1}^{N_y} \varepsilon_{ik}^2 \tag{15}$$

In its simplest form, backpropagation training begins by presenting an input pattern vector $\mathbf{X}$ to the network, sweeping

forward through the system to generate an output response vector **Y**, and computing the errors at each output. We continue by sweeping the effects of the errors backward through the network to associate a *square error derivative* δ with each adaline, computing a gradient from each δ, and finally updating the weights of each adaline based on the corresponding gradient. A new pattern is then presented and the process is repeated. The initial weight values are normally set to small random values. The algorithm will not work properly with multilayer networks if the initial weights are either zero or poorly chosen non-zero values.

The δ's in the output layer are computed just as they are for the sigmoid adaline element. For a given output adaline,

$$\delta = \tilde{\varepsilon}\,\mathrm{sgm}'(s) \tag{16}$$

where $\tilde{\varepsilon}$ is the error at the output of the adaline and s is the summing junction output of the same unit.

Hidden layer calculations, however, are more complicated. The procedure for finding the value of $\delta^{(l)}$, the value of δ associated with a given adaline in hidden layer l, involves respectively multiplying each derivative $\delta^{(l+1)}$ associated with each element in the layer immediately downstream from the given adaline by the weight connecting it to the given adaline. These weighted square error derivatives are then added together, producing an error term $\varepsilon^{(l)}$, which in turn is multiplied by $\mathrm{sgm}'(s^{(l)})$, the derivative of the given adaline's sigmoid function at its current operating point. Thus, the δ corresponding to adaline j in hidden layer l is given by

$$\delta_j^{(l)} = \mathrm{sgm}'(s_j^{(l)}) \sum_{i \in N^{(l+1)}} \delta_i^{(l+1)} w_{ij}^{(l+1)} \tag{17}$$

where $N^{(l+1)}$ is a set containing the indices of all adalines in layer $l+1$ and $w_{ij}^{(l+1)}$ is the weight connecting adaline i in layer $l+1$ to the output of adaline j in layer l.

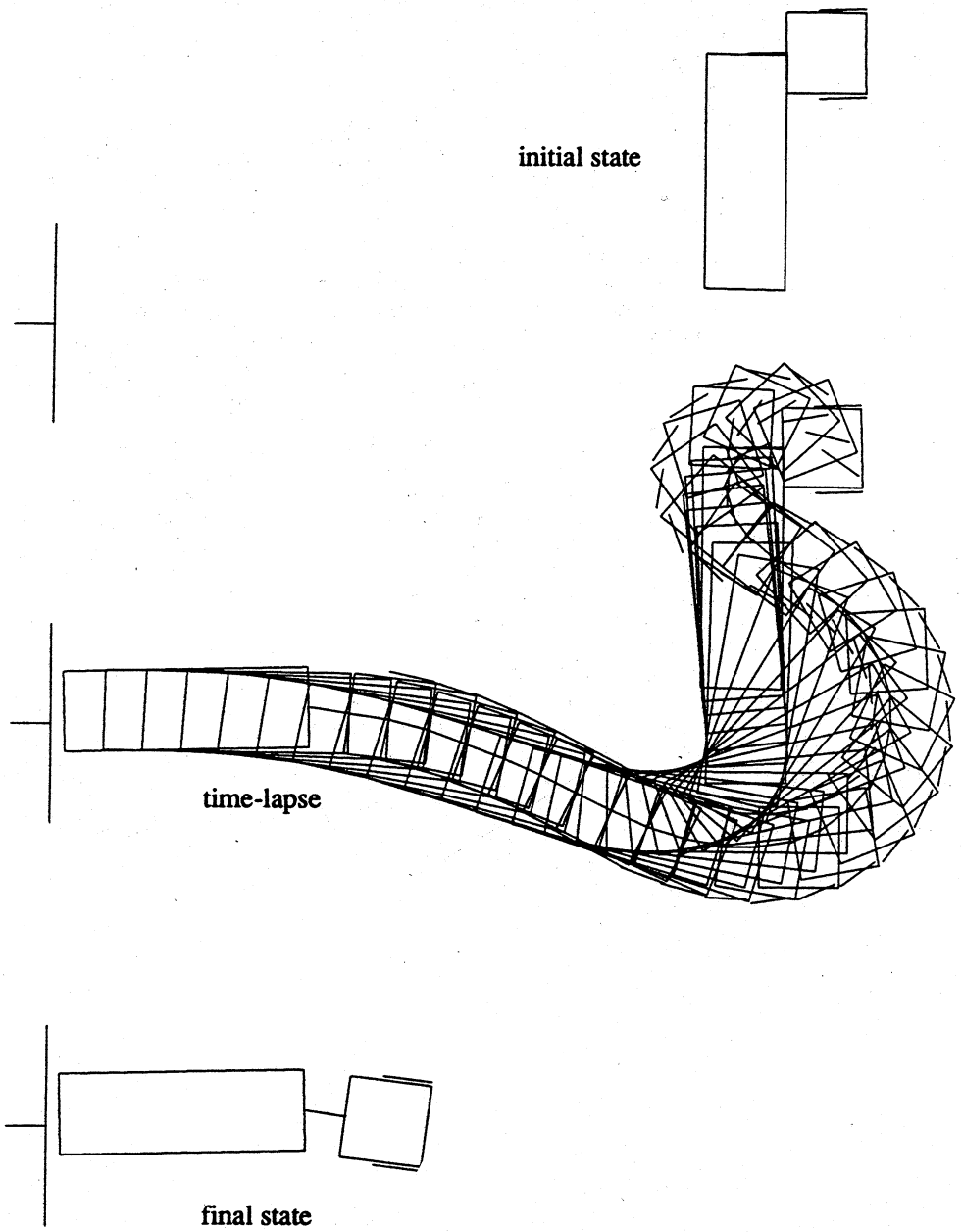

Figure 4. Example of a truck backup sequence.

Updating the weights of the adaline element using the method of gradient descent with the instantaneous gradient is a process represented by

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu(-\hat{\nabla}_k) = \mathbf{W}_k + 2\mu\delta_k\mathbf{X}_k \qquad (18)$$

where $\mathbf{W}$ is the adaline's weight vector and $\mathbf{X}$ is the vector of inputs to the adaline. Thus, after backpropagating all square error derivatives, we complete a backpropagation iteration by adding to each weight vector the corresponding input vector scaled by the associated square error derivative. Equations 16, 17, and 18 comprise the general weight update rule of the backpropagation algorithm for layered neural networks.

Many useful techniques based on the backpropagation algorithm have been developed. One popular method, called *backpropagation through time*, allows dynamical recurrent networks to be trained. Essentially, this is accomplished by running the recurrent neural network for several time steps and then "unrolling" the network in time. This results in a virtual network with a number of layers equal to the product of the original number of layers and the number of time steps. The ordinary backpropagation algorithm is then applied to this virtual network and the result is used to update the weights of the original network. This approach was used by Nguyen and Widrow (1989) to enable a neural network to learn without a teacher how to back up a computer-simulated trailer truck to a loading dock (Figure 4). This is a complicated and highly nonlinear steering task. Nevertheless, with just six inputs providing information about the current position of the truck, a two-layer neural network with only 26 sigmoid adalines was able to learn of its own accord to solve this problem. Once trained, the network could successfully back up the truck from any initial position and orientation in front of the loading dock.

Discussion

Although this article has focused on pattern classification issues, nonlinear neural networks are equally useful for such tasks as interpolation, system modeling, state estimation, adaptive filtering, and nonlinear control. Unlike their linear counterparts which have a long track record of success, nonlinear neural networks have only recently begun proving themselves in commercial applications. The capabilities of multielement neural networks have improved markedly since the introduction of Madaline Rule I. This has resulted largely from development of the backpropagation algorithm, easily the most useful and popular neural network training algorithm currently available. As we have seen, backpropagation is a generalization of LMS which allows complex networks of sigmoid adalines to be efficiently adapted. Backpropagation and related algorithms are in a large part responsible for the dramatic growth the field of neural networks is currently experiencing.

The timing of the current boom in the field of neural networks is also due to the rapid advance of computer and microprocessor performance which continues to improve the feasibility and cost-effectiveness of computationally expensive techniques in relation to classical approaches of engineering and statistics. Although single-element linear adaptive filters are still used more extensively than nonlinear multielement neural networks, the latter are potentially applicable to a much wider range of problems. Furthermore, the applications for which multielement neural networks are best suited often involve complicated nonlinear relationships for which classical solutions are either ineffective or unavailable. The continued advancement of neural network algorithms and techniques, in conjunction with improvements in the special and general purpose computer hardware used to implement them, sets the stage for a future in which neural networks will play an increasing role in commercial and industrial applications.

Acknowledgments. This work was sponsored by NSF under grant IRI 91–12531, by ONR under contract N00014–92–J–1787, by EPRI under contract RP 8010–13, and by the U.S. Army under contract DAAK70–92–K–0003.

Road Map: Learning in Artificial Neural Networks, Deterministic
Background: I.3. Dynamics and Adaptation in Neural Networks
Related Reading: Adaptive Control: Neural Network Applications; Adaptive Filtering; Learning as Hill-Climbing in Weight Space

References

Nilsson, N., 1965, *Learning Machines*, New York: McGraw-Hill. ◆

Nguyen, D., and Widrow, B., 1989, The truck backer-upper: An example of self-learning in neural networks, in *Proceedings of the International Joint Conference on Neural Networks*, vol. 2, New York: IEEE, pp. 357–363.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J., 1986, Learning internal representations by error propagation, in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (D. E. Rumelhart, J. L. McClelland, and PDP Research Group, Eds.), vol. 1, *Foundations*, Cambridge, MA: MIT Press, chap. 8. ◆

Rosenblatt, F., 1962, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Washington, DC: Spartan.

Widrow, B., and Hoff, M. E., Jr., 1960, Adaptive switching circuits, in *1960 IRE WESCON Convention Record*, Part 4, New York: IRE, pp. 96–104.

Widrow, B., and Lehr, M. A., 1990, 30 years of adaptive neural networks: Perceptron, madaline, and backpropagation, *Proc. IEEE*, 78: 1415–1442. ◆

Widrow, B., Rumelhart, D., and Lehr, M. A., 1994, Neural networks: Applications in industry, business, and science, *Commun. ACM*, 37(3):93–105.

Widrow, B., and Stearns, S. D., 1985, *Adaptive Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall. ◆

The Handbook of Brain Theory and Neural Networks

EDITED BY

Michael A. Arbib

A Bradford Book

THE MIT PRESS

Cambridge, Massachusetts

London, England