Backpropagation and its Applications

Bernard Widrow Michael A. Lehr
Stanford University Department of Electrical Engineering, Stanford, CA 94305-4055

Backpropagation remains the most widely used neural network algorithm. We present a basic review of this method and its applications. The backpropagation algorithm's roots are also explored in a discussion of its predecessor, the LMS Algorithm.

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 [1] and the LMS algorithm (or Widrow-Hoff rule) [2]. Around 1961, Widrow and his students devised Madaline Rule I (MRI), the earliest learning rule for feedforward networks with multiple adaptive elements [3]. The major extension of the feedforward neural network beyond Madaline I took place in 1971 when Werbos developed a backpropagation algorithm for training multilayer networks. In 1974, this algorithm was incorporated in his doctoral dissertation [4]†. Werbos's work remained almost unknown in the scientific community. In 1982 Parker rediscovered the technique and in 1985, published a report on it at MIT [6]. Not long after Parker published his findings, Rumelhart, Hinton, and Williams [7] refined the technique and succeeded in making it widely known.

Early applications of LMS and MRI were developed by Widrow and his students in their studies of speech and pattern recognition, weather forecasting, and adaptive controls. After some success in these areas, work shifted in the mid-1960's to adaptive filtering and adaptive signal processing. This proved to be a fruitful avenue for research with applications including adaptive antennas, adaptive inverse controls, adaptive noise cancelling, and seismic signal processing. Outstanding work by R. W. Lucky and others at Bell Laboratories led to major commercial applications of adaptive filters and the LMS algorithm to adaptive equalization in high speed modems and to adaptive echo cancellers for long distance telephone and satellite circuits.

†We should note, however, that in the field of variational calculus the idea of error backpropagation through nonlinear systems existed centuries before Werbos first thought to apply the concept to neural networks. In the past 25 years, these methods have been used widely in the the field of optimal control, as discussed by Le Cun [5].

With the development of the backpropagation algorithm, it has now become possible to successfully attack problems requiring neural networks with high degrees of nonlinearity and high precision. Examples are shown in [8]. Backpropagation networks with fewer than 150 neural elements have been successfully applied in vehicular control simulations, speech generation, and undersea mine detection. Small networks have also been used successfully in airport explosive detection, expert systems, nonlinear system identification, 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 like continuous speech recognition.

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

The Adaptive Linear Combiner

The adaptive linear combiner is diagrammed in Fig. 1. Its output is 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 adaptive 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. By using a training algorithm to adapt its weights, the adaptive linear combiner has the ability to implement a wide range of responses to the patterns in a given training set.

When a nonlinearity is placed on the output of an adaptive linear combiner, the cascade is called an Adaline (ADAptive LINear Element). In the 1960's, the Adaline's nonlinearity was usually the sign or signum element. The Adaline was also known as a linear threshold element whose output was either +1 or −1. Modern neural networks usually use an "S"-shaped "sigmoid" function, a "soft" quantizer which varies smoothly from −1 to +1. An Adaline using a sigmoid nonlinearity is sometimes referred to as a Sigmoid Adaline. The Adaline usually includes a bias weight w_{0_k} which is connected to a constant input, $x_0 = +1$. This weight effectively

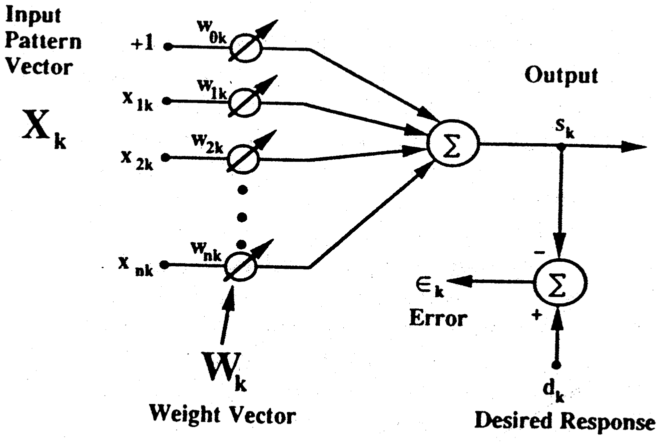

Figure 1: Adaptive linear combiner.

controls the threshold level of the quantizer or sigmoid.

Multilayer Networks

The Madaline networks of the 1960s had adaptive first layers and fixed threshold functions in the second (output) layers [8]. The feedforward neural networks of today often have many layers, and usually all layers are adaptive. The backpropagation networks of Rumelhart *et al.* [9] are perhaps the best-known examples of multilayer networks. A fully-connected three-layer feedforward adaptive network is illustrated in Fig. 2. In a fully-connected layered network, each Adaline receives inputs from every output in the preceding layer.

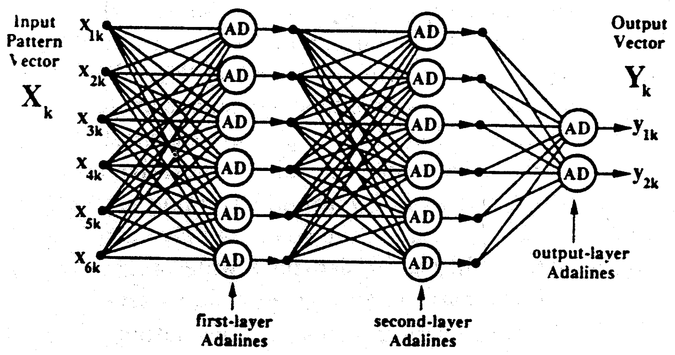

Figure 2: A three-layer adaptive neural network.

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.

Steepest-Descent Rules

Usually, the objective of adaptation for a feedforward neural network is to reduce the error between the desired response and the network's response averaged in some way over the training set. The most common error function is mean-square-error (MSE), although in some situations other error criteria may be more appropriate [8]. The most popular approaches to mean-square-error reduction in both single-element and multi-element networks are based upon the method of steepest descent. More sophisticated gradient approaches such as quasi-Newton and conjugate gradient techniques often have better convergence properties, but the conditions under which the additional complexity is warranted are not generally known. The discussion that follows is restricted to minimization of MSE by the method of steepest descent [10]. More sophisticated learning procedures usually require many of the same computations used in the basic steepest-descent procedure.

Adaptation of a network by steepest-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 corresponding to the negative of 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 steepest descent can be described by the relation

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

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 The LMS algorithm works by performing approximate steepest descent on the mean-square-error surface in weight space. Because it is a quadratic function of the weights, this surface is convex and has a unique (global) minimum[‡]. An instantaneous gradient based upon the square of the

[‡]unless the autocorrelation matrix of the pattern vector set has m zero eigenvalues, in which case the minimum MSE solution will be an m dimensional subspace in weight space [11].

instantaneous error is

$$\hat{\nabla}_k = \frac{\partial \epsilon_k^2}{\partial \mathbf{W}_k} = \left\{ \begin{array}{c} \frac{\partial \epsilon_k^2}{\partial w_{0k}} \\ \vdots \\ \frac{\epsilon_k^2}{\partial w_{nk}} \end{array} \right\}. \qquad (2)$$

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

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

The instantaneous gradient is used because (a) it is an unbiased estimate of the true [11] gradient, 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 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:

$$\epsilon_k \triangleq d_k - \mathbf{W}_k^T \mathbf{X}_k. \qquad (4)$$

Performing the differentiation in Eq. (3) and replacing the error with (4) gives

$$\begin{aligned} \mathbf{W}_{k+1} &= \mathbf{W}_k - 2\mu\epsilon_k \frac{\partial \epsilon_k}{\partial \mathbf{W}_k} \\ &= \mathbf{W}_k - 2\mu\epsilon_k \frac{\partial\left(d_k - \mathbf{W}_k^T \mathbf{X}_k\right)}{\partial \mathbf{W}_k}. \qquad (5) \end{aligned}$$

Noting that d_k and $\mathbf{X}_k$ are independent of $\mathbf{W}_k$, we obtain

$$\mathbf{W}_{k+1} = \mathbf{W}_k + 2\mu\epsilon_k \mathbf{X}_k. \qquad (6)$$

This is the LMS algorithm. The learning constant μ determines stability and convergence rate. For input patterns independent over time, convergence of the mean and variance of the weight vector is ensured under fairly general conditions [12, 8] if

$$0 < \mu < \frac{1}{3trace[\mathbf{R}]}, \qquad (7)$$

where $\mathbf{R}$ is the input correlation matrix $\mathrm{E}[\mathbf{X}_k \mathbf{X}_k^T]$ and where $trace[\mathbf{R}] = \sum$(diagonal elements of $\mathbf{R}$) is the sum of the signal powers of the components of the $\mathbf{X}$-vectors, i.e. $\mathrm{E}(\mathbf{X}^T\mathbf{X})$. With μ set within this range, the LMS algorithm converges in the mean to $\mathbf{W}^*$, the optimal Wiener solution [11].

In the LMS algorithm, and other iterative steepest-descent procedures, use of the instantaneous gradient is perfectly justified if the step size is small. For small μ, $\mathbf{W}$ will remain essentially constant over a relatively small number of training presentations, K. The total weight change during this period will be proportional to

$$\begin{aligned} -\sum_{\ell=0}^{K-1} \frac{\partial \epsilon_{k+\ell}^2}{\partial \mathbf{W}_{k+\ell}} &\simeq -K\left(\frac{1}{K}\sum_{\ell=0}^{K-1} \frac{\partial \epsilon_{k+\ell}^2}{\partial \mathbf{W}_k}\right) \\ &= -K\frac{\partial}{\partial \mathbf{W}_k}\left(\frac{1}{K}\sum_{\ell=0}^{K-1} \epsilon_{k+\ell}^2\right) \\ &\simeq -K\frac{\partial \xi}{\partial \mathbf{W}_k}, \qquad (8) \end{aligned}$$

where ξ denotes the mean-square-error function. Thus, on average the weights follow the true gradient.

Backpropagation for the Sigmoid Adaline Fig. 3 shows a "Sigmoid Adaline" element which incorporates a sigmoidal nonlinearity. The input-output relation of the sigmoid can be denoted by $y_k = sgm(s_k)$. A typical sigmoid function is the hyperbolic tangent:

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

We shall adapt this Adaline with the objective of

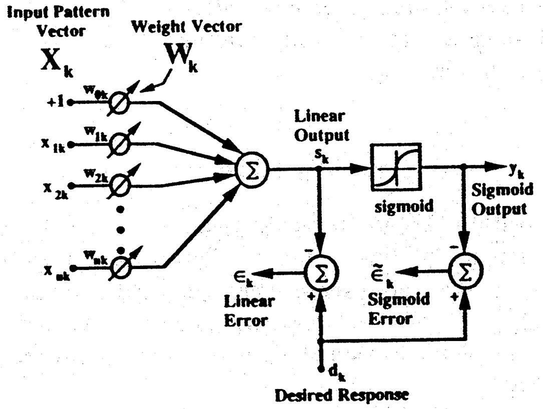

Figure 3: Adaline with sigmoidal nonlinearity.

minimizing the mean square of the sigmoid error $\tilde{\epsilon}_k$, defined as

$$\tilde{\epsilon}_k \triangleq d_k - y_k = d_k - sgm(s_k). \qquad (10)$$

Our objective is to minimize $\mathrm{E}[(\tilde{\epsilon}_k)^2]$, averaged over the set of training patterns, by proper choice of the weight vector. To accomplish this, we shall derive a backpropagation algorithm for the Sigmoid

Adaline element. An instantaneous gradient is obtained with each input vector presentation, and the method of steepest descent is used to minimize error as was done with the LMS algorithm of Eq. (6).

Referring to Fig. 3, the instantaneous gradient estimate obtained during presentation of the kth input vector $\mathbf{X}_k$ is given by

$$\hat{\nabla}_k = \frac{\partial(\tilde{\epsilon}_k)^2}{\partial \mathbf{W}_k} = 2\tilde{\epsilon}_k \frac{\partial \tilde{\epsilon}_k}{\partial \mathbf{W}_k}. \tag{11}$$

Differentiating Eq. (10) yields

$$\frac{\partial \tilde{\epsilon}_k}{\partial \mathbf{W}_k} = -\frac{\partial sgm(s_k)}{\partial \mathbf{W}_k} = -sgm'(s_k)\frac{\partial s_k}{\partial \mathbf{W}_k}. \tag{12}$$

We may note that

$$s_k = \mathbf{X}_k^T \mathbf{W}_k. \tag{13}$$

Therefore,

$$\frac{\partial s_k}{\partial \mathbf{W}_k} = \mathbf{X}_k. \tag{14}$$

Substituting into Eq. (12) gives

$$\frac{\partial \tilde{\epsilon}_k}{\partial \mathbf{W}_k} = -sgm'(s_k)\mathbf{X}_k. \tag{15}$$

Inserting this into Eq. (11) yields

$$\hat{\nabla}_k = -2\tilde{\epsilon}_k sgm'(s_k)\mathbf{X}_k. \tag{16}$$

Using this gradient estimate with the method of steepest 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

$$\begin{aligned} \mathbf{W}_{k+1} &= \mathbf{W}_k + \mu(-\hat{\nabla}_k) & (17)\\ &= \mathbf{W}_k + 2\mu\tilde{\epsilon}_k sgm'(s_k)\mathbf{X}_k. & (18) \end{aligned}$$

Algorithm (18) is the *backpropagation* algorithm for the single Adaline element. The backpropagation name makes more sense when the algorithm is utilized in a layered network, which will be studied below. Implementation of algorithm (18) is illustrated in Fig. 4.

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

$$\begin{aligned} sgm'(s_k) &= \frac{\partial(tanh(s_k))}{\partial s_k} \\ &= 1-(tanh(s_k))^2 = 1 - y_k^2. & (19) \end{aligned}$$

Accordingly Eq. (18) becomes

$$\mathbf{W}_{k+1} = \mathbf{W}_k + 2\mu\tilde{\epsilon}_k(1-y_k^2)\mathbf{X}_k. \tag{20}$$

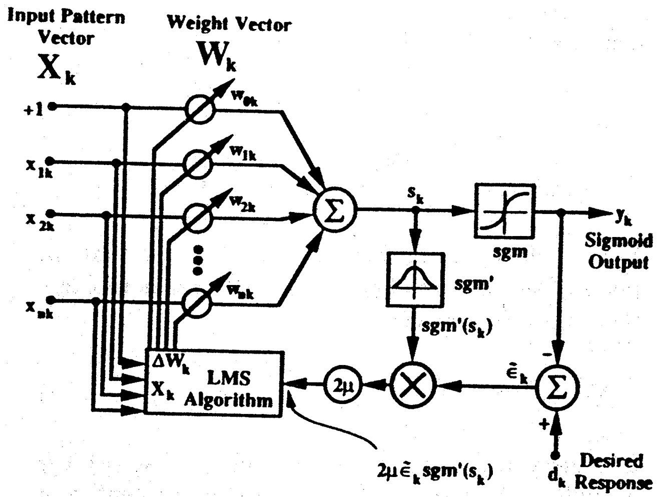

Figure 4: Implementation of backpropagation for the Sigmoid Adaline element.

Backpropagation for Networks The publication of the backpropagation technique by Rumelhart *et al.* [7] has unquestionably been the most influential development in the field of neural networks during the past decade. In retrospect, the technique seems simple. Nonetheless, largely because early neural network research dealt almost exclusively with hard-limiting nonlinearities, the idea eluded neural network researchers throughout the 1960's.

The basic concepts of backpropagation are easily grasped. Unfortunately, these simple ideas are often obscured by relatively intricate notation, so formal derivations of the backpropagation rule are often tedious. We present an informal derivation of the algorithm and illustrate how it works for the simple network shown in Fig. 5. A more formal derivation would require the use of ordered derivatives to precisely identify which quantities are treated as variables in each of the partial derivatives used below [13].

The backpropagation technique is a substantial generalization of the single Sigmoid Adaline case discussed in the previous section. When applied to multi-element networks, the backpropagation technique adjusts the weights in the direction opposite to the instantaneous error gradient:

$$\hat{\nabla}_k = \frac{\partial \varepsilon_k^2}{\partial \mathbf{W}_k} = \left\{ \begin{array}{c} \frac{\partial \varepsilon_k^2}{\partial w_{1k}} \\ \vdots \\ \frac{\partial \varepsilon_k^2}{\partial w_{nk}} \end{array} \right\}. \tag{21}$$

Now, however, $\mathbf{W}_k$ is a long n-component vector of all weights in the entire network. The instantaneous sum squared error ε_k^2 is the sum of the squares of the errors at each of the N_y outputs of the network.

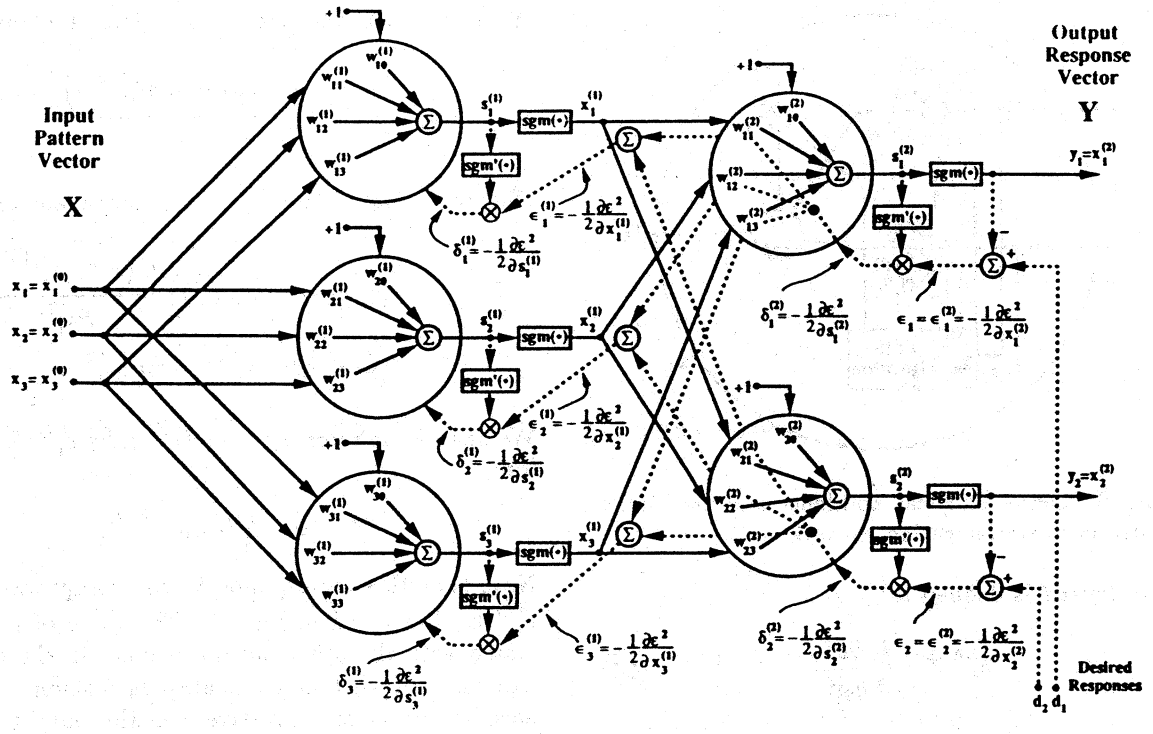

Figure 5: Example two-layer backpropagation network architecture.

Thus

$$\varepsilon_k^2 = \sum_{i=1}^{N_y} \epsilon_{ik}^2. \tag{22}$$

In the network example shown in Fig. 5, the sum square error is given by

$$\varepsilon^2 = (d_1 - y_1)^2 + (d_2 - y_2)^2, \tag{23}$$

where we now suppress the time index k for convenience.

In its simplest form, backpropagation training begins by presenting an input pattern vector **X** to the network, sweeping forward through the system to generate an output response vector **Y**, and computing the errors at each output. The next step involves 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 upon the corresponding gradient. A new pattern is then presented and the process is repeated. The initial weight values are normally set to small random numbers. The algorithm will not work properly with multilayer networks if the initial weights are either zero or poorly chosen nonzero values.

We can get some idea about what is involved in the calculations associated with the backpropagation algorithm by examining the network of Fig. 5. Each of the five large circles represents a linear combiner, as well as some associated signal paths for error backpropagation, and the corresponding adaptive machinery for updating the weights. This detail is shown in Fig. 6. The solid lines in these diagrams represent forward signal paths through the network, and the dotted lines represent the separate backward paths that are used in association with calculations of the square error derivatives δ. From Fig. 5, we see that the calculations associated with the backward sweep are of a complexity which is roughly equal to that represented by the forward pass through the network. The backward sweep requires the same number of function calculations as the forward sweep, but it requires no weight multiplications in the first layer.

As stated above, after a pattern has been presented to the network, and the response error of each output has been calculated, the next step of the backpropagation algorithm involves finding the instantaneous square error derivative δ associated with each summing junction in the network. The square error derivative associated with the jth Ada-

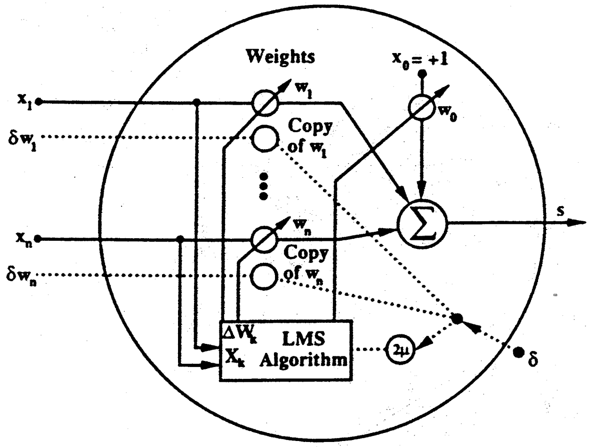

Figure 6: Detail of linear combiner and associated circuitry in backpropagation network.

line in layer ℓ is defined as[§]

$$\delta_j^{(\ell)} \triangleq -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial s_j^{(\ell)}}. \qquad (24)$$

Each of these derivatives in essence tells us how sensitive the sum square output error of the network is to changes in the linear output of the associated Adaline element.

The instantaneous square error derivatives are first computed for each element in the output layer. The calculation is simple. As an example, below we derive the required expression for $\delta_1^{(2)}$, the derivative associated with the top Adaline element in the output layer of Fig. 5. We begin with the definition of $\delta_1^{(2)}$ from Eq. (24),

$$\delta_1^{(2)} \triangleq -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial s_1^{(2)}}. \qquad (25)$$

Expanding the squared error term ε^2 by Eq. (23) yields

$$\delta_1^{(2)} = -\frac{1}{2}\frac{\partial\left((d_1-y_1)^2+(d_2-y_2)^2\right)}{\partial s_1^{(2)}} \qquad (26)$$

$$= -\frac{1}{2}\frac{\partial\left(d_1 - sgm(s_1^{(2)})\right)^2}{\partial s_1^{(2)}}$$

$$-\frac{1}{2}\frac{\partial\left(d_2 - sgm(s_2^{(2)})\right)^2}{\partial s_1^{(2)}}. \qquad (27)$$

[§]In Fig. 5, all notation follows the convention that superscripts within parentheses indicate the layer number of the associated Adaline or input node, while subscripts identify the associated Adaline(s) within a layer.

We note that the second term is zero. Accordingly,

$$\delta_1^{(2)} = -\frac{1}{2}\frac{\partial\left(d_1 - sgm(s_1^{(2)})\right)^2}{\partial s_1^{(2)}}. \qquad (28)$$

Observing that d_1 and $s_1^{(2)}$ are independent yields

$$\delta_1^{(2)} = -\left(d_1 - sgm(s_1^{(2)})\right)\frac{\partial\left(-sgm(s_1^{(2)})\right)}{\partial s_1^{(2)}}$$

$$= \left(d_1 - sgm(s_1^{(2)})\right) sgm'(s_1^{(2)}). \qquad (29)$$

We denote the error $d_1 - sgm(s_1^{(2)})$, by $\epsilon_1^{(2)}$. Therefore,

$$\delta_1^{(2)} = \epsilon_1^{(2)} sgm'(s_1^{(2)}). \qquad (30)$$

Note that this corresponds to the computation of $\delta_1^{(2)}$ as illustrated in Fig. 5. The value of δ associated with the other output element in the figure can be expressed in an analogous fashion. Thus each square error derivative δ in the output layer is computed by multiplying the output error associated with that element by the derivative of the associated sigmoidal nonlinearity. Note from Eq. (19) that if the sigmoid function is the hyperbolic tangent, Eq. (30) becomes simply

$$\delta_1^{(2)} = \epsilon_1^{(2)}(1-(y_1)^2). \qquad (31)$$

Developing expressions for the square error derivatives associated with hidden layers is not much more difficult (refer to Fig. 5). We need an expression for $\delta_1^{(1)}$, the square error derivative associated with the top element in the first layer of Fig. 5. The derivative $\delta_1^{(1)}$ is defined by

$$\delta_1^{(1)} \triangleq -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial s_1^{(1)}}. \qquad (32)$$

Expanding this by the chain rule, noting that ε^2 is determined entirely by the values of $s_1^{(2)}$ and $s_2^{(2)}$, yields

$$\delta_1^{(1)} = -\frac{1}{2}\left(\frac{\partial \varepsilon^2}{\partial s_1^{(2)}}\frac{\partial s_1^{(2)}}{\partial s_1^{(1)}} + \frac{\partial \varepsilon^2}{\partial s_2^{(2)}}\frac{\partial s_2^{(2)}}{\partial s_1^{(1)}}\right). \qquad (33)$$

Using the definitions of $\delta_1^{(2)}$ and $\delta_2^{(2)}$, and then substituting expanded versions of Adaline linear outputs $s_1^{(2)}$ and $s_2^{(2)}$ gives

$$\delta_1^{(1)} = \delta_1^{(2)}\frac{\partial s_1^{(2)}}{\partial s_1^{(1)}} + \delta_2^{(2)}\frac{\partial s_2^{(2)}}{\partial s_1^{(1)}} \qquad (34)$$

$$= \delta_1^{(2)} \frac{\partial}{\partial s_1^{(1)}} \left(w_{10}^{(2)} + \sum_{i=1}^{3} w_{1i}^{(2)} sgm(s_i^{(1)}) \right)$$
$$+\delta_2^{(2)} \frac{\partial}{\partial s_1^{(1)}} \left(w_{20}^{(2)} + \sum_{i=1}^{3} w_{2i}^{(2)} sgm(s_i^{(1)}) \right).$$

Noting that $\partial[sgm(s_i^{(\ell)})]/\partial s_j^{(\ell)} = 0, i \neq j$, leaves

$$\begin{aligned} \delta_1^{(1)} &= \delta_1^{(2)} w_{11}^{(2)} sgm'(s_1^{(1)}) + \delta_2^{(2)} w_{21}^{(2)} sgm'(s_1^{(1)}) \\ &= \left[\delta_1^{(2)} w_{11}^{(2)} + \delta_2^{(2)} w_{21}^{(2)}\right] sgm'(s_1^{(1)}). \end{aligned} \quad (35)$$

Now, we make the following definition:

$$\epsilon_1^{(1)} \triangleq \delta_1^{(2)} w_{11}^{(2)} + \delta_2^{(2)} w_{21}^{(2)}. \quad (36)$$

Accordingly,

$$\delta_1^{(1)} = \epsilon_1^{(1)} sgm'(s_1^{(1)}). \quad (37)$$

Referring to Fig. 5, we can trace through the circuit to verify that $\delta_1^{(1)}$ is computed in accord with Eqs. (36) and (37). The easiest way to find values of δ for all the Adaline elements in the network is to follow the schematic diagram of Fig. 5.

Thus, the procedure for finding $\delta^{(\ell)}$, the square error derivative associated with a given Adaline in hidden layer ℓ, involves respectively multiplying each derivative $\delta^{(\ell+1)}$ associated with each element in the layer immediately downstream from a given Adaline by the weight which connects it to the given Adaline. These weighted square error derivatives are then added together, producing an error term $\epsilon^{(\ell)}$, which, in turn, is multiplied by $sgm'(s^{(\ell)})$, the derivative of the given Adaline's sigmoid function at its current operating point. If a network has more than two layers, this process of backpropagating the instantaneous square error derivatives from one layer to the immediately preceding layer is successively repeated until a square error derivative δ is computed for each Adaline in the network. This is easily shown at each layer by repeating the chain rule argument associated with Eq. (33).

We now have a general method for finding a derivative δ for each Adaline element in the network. The next step is to use these δ's to obtain the corresponding gradients. Consider an Adaline somewhere in the network which, during presentation k, has a weight vector $\mathbf{W}_k$, an input vector $\mathbf{X}_k$, and a linear output $s_k = \mathbf{W}_k^T \mathbf{X}_k$.
The instantaneous gradient for this Adaline element is

$$\hat{\nabla}_k = \frac{\partial \varepsilon_k^2}{\partial \mathbf{W}_k}. \quad (38)$$

This can be written as

$$\hat{\nabla}_k = \frac{\partial \varepsilon_k^2}{\partial \mathbf{W}_k} = \frac{\partial \varepsilon_k^2}{\partial s_k} \frac{\partial s_k}{\partial \mathbf{W}_k}. \quad (39)$$

Note that $\mathbf{W}_k$ and $\mathbf{X}_k$ are independent so

$$\frac{\partial s_k}{\partial \mathbf{W}_k} = \frac{\partial \mathbf{W}_k^T \mathbf{X}_k}{\partial \mathbf{W}_k} = \mathbf{X}_k. \quad (40)$$

Therefore,

$$\hat{\nabla}_k = \frac{\partial \varepsilon_k^2}{\partial s_k} \mathbf{X}_k. \quad (41)$$

For this element,

$$\delta_k = -\frac{1}{2} \frac{\partial \varepsilon_k^2}{\partial s_k}. \quad (42)$$

Accordingly,

$$\hat{\nabla}_k = -2\delta_k \mathbf{X}_k. \quad (43)$$

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

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu(-\hat{\nabla}_k) = \mathbf{W}_k + 2\mu\delta_k \mathbf{X}_k. \quad (44)$$

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. Eq. (44) and the means for finding δ_k comprise the general weight update rule of the backpropagation algorithm.

There is a great similarity between Eq. (44) and the LMS algorithm (6), but one should view this similarity with caution. The quantity δ_k, defined as a squared error derivative, might appear to play the same role in backpropagation as that played by the error in the LMS algorithm. However, δ_k is not an error. Adaptation of the given Adaline is effected to reduce the squared output error ε_k^2, not δ_k of the given Adaline or of any other Adaline in the network. The objective is not to reduce the δ_k's of the network, but to reduce ε_k^2 at the network output.

It is interesting to examine the weight updates that backpropagation imposes on the Adaline elements in the output layer. Substituting Eq. (29) into Eq. (44) reveals that the Adaline which provides output y_1 in Fig. 5 is updated by the rule

$$\mathbf{W}_{k+1} = \mathbf{W}_k + 2\mu\epsilon_1^{(2)} sgm'(s_1^{(2)}) \mathbf{X}_k. \quad (45)$$

This rule turns out to be identical to the single Adaline version (18) of the backpropagation rule. This is not surprising since the output Adaline is provided with both input signals and desired responses, so its training circumstance is the same as that experienced by an Adaline trained in isolation.

In our experience, weight initialization and input normalization can strongly affect the training speed of neural networks. In recent years, we have relied primarily on a heuristic designed to ensure that none of the sigmoids will be saturated during the early stages of training. This saturation can impede learning since the derivative of a saturated sigmoid is near zero, and this derivative is proportional to the gradient vector of the corresponding Adaline. In the input layer, we avoid this problem by normalizing the input patterns so each component is zero-mean with a variance of roughly 0.2[¶]. The weights are then randomly initialized to have a distribution which is zero-mean i.i.d. Gaussian with a variance $1.5/N$, where N is the number of inputs to the Adaline being initialized. If the standard $\tanh(\cdot)$ sigmoid is used, this causes the signal level at the input of each sigmoid in the network to be distributed roughly zero-mean Gaussian with a variance of 0.3 when training commences. The output of each sigmoid will then be distributed with a variance of roughly 0.2, just like the network inputs. Due to the linearity of the sigmoid at small signal levels, the use of relatively low-variance sigmoid inputs helps keep the size of the derivatives backpropagated through the network relatively uniform, if the number of Adalines per layer does not vary drastically.

Another weight initialization approach with which we have had considerable success in many instances is one devised by Nguyen [14]. This approach involves choosing the initial weights of each hidden layer in a quasi-random manner which ensures that at each position in a layer's input space the outputs of all but a few of its Adalines will be saturated, while ensuring that each Adaline in the layer is unsaturated in some region of its input space.

One of the most promising new areas of neural network research involves backpropagation variants for training various recurrent (signal feedback) networks. Recently, backpropagation rules have been devised for training recurrent networks to learn static associations [15, 16]. More interesting is the on-line technique of Williams and Zipser [17] which allows a wide class of recurrent networks to learn dynamic associations and trajectories. A more general and computationally viable variant of this technique has been advanced by Narendra and Parthasarathy [18]. These on-line methods are generalizations of a well-known steepest-descent algorithm for training linear IIR filters [19, 11].

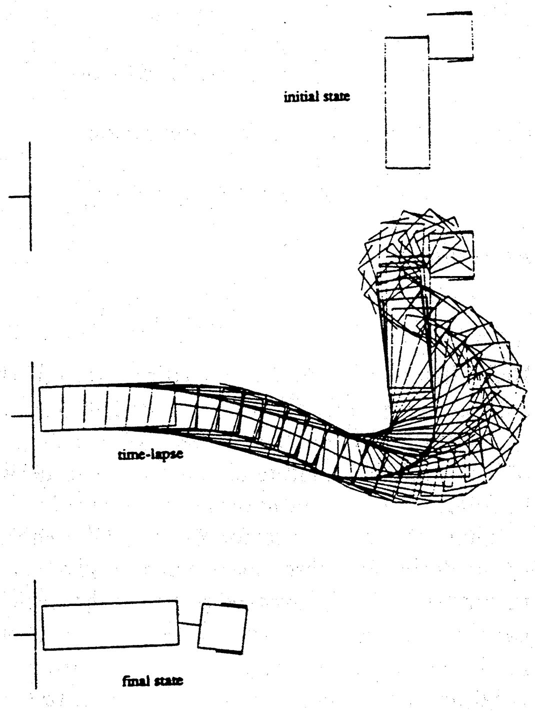

Figure 7: Example Truck Backup Sequence.

An equivalent technique which is usually far less computationally intensive also exists [4, 7, 20]. This approach, called "backpropagation-through-time," has been used by Nguyen and Widrow [21] to enable a neural network to learn without a teacher how to back up a computer-simulated trailer truck to a loading dock (Fig. 7). The approach used is described in detail in another paper in these proceedings [22]. It involved two networks: a neural emulator trained by backpropagation to model the kinematics of the truck, and a neural controller trained by backpropagation-through-time to generate the required steering signals. The process of backing up a truck is a complicated and highly nonlinear steering task. Nevertheless, the controller network was able to learn of its own accord to solve this problem. Once trained, the controller network could successfully back up the truck from any initial position and orientation in front of the loading dock.

[¶]To do this we merely compute a mean and sample-variance for each network input over the training pattern set. Then, for each input pattern vector $\mathbf{X}$ in both the training and generalization sets, we subtract the appropriate mean from each component, divide by the corresponding standard deviation, and multiply the result by the square root of 0.2.

Summary

The backpropagation algorithm can be employed to solve a wide range of problems. There is an enormous pool of applications to which it has has been successfully applied—only a few are mentioned in this paper—and there are scores more that will become realities in the next few years. Just as neural networks can be used to emulate the kinematics of a trailer truck and control its steering, they can also be used to model and control processes in the power industry. Likely applications may include magnetic bearing control to increase generator efficiency and reduce bearing wear; nuclear reactor monitoring and modeling for safety; and control of fuel metering to improve voltage regulation and efficiency. In the coming years, we expect to see a growing amount of research on backpropagation applications in the power industry. In the long run, we hope and believe the industry will benefit greatly from the fruits of this activity.

Acknowledgments
This work was sponsored by SDIO Innovative Science and Technology Office and managed by ONR under contract #N00014-86-K-0718, by the Department of the Army Belvoir R D & E Center under contract #DAAK70-92-K-0001, by a grant from the Lockheed Missiles and Space Company, by NASA under contract #NCA2-389, and by Rome Air Development Center under contract #F30602-88-D-0025, subcontract E-21-T22-S1.

References

[1] F. Rosenblatt. On the convergence of reinforcement procedures in simple perceptrons. *Cornell Aeronautical Laboratory Report VG-1196-G-4*, Buffalo, New York, February 1960.

[2] B. Widrow and M. E. Hoff, Jr. Adaptive switching circuits. In *1960 IRE Western Electric Show and Convention Record, Part 4*, pages 96–104, August 23 1960.

[3] B. Widrow. Generalization and information storage in networks of adaline "neurons". In M. Yovitz, G. Jacobi, and G. Goldstein, editors, *Self-Organizing Systems 1962*, pages 435–461. Spartan Books, Washington, DC, 1962.

[4] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, Cambridge, MA, August 1974.

[5] Y. le Cun. A theoretical framework for backpropagation. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 21–28, San Mateo, CA, June 17-26 1988. Morgan Kaufmann.

[6] D. Parker. Learning-logic. Technical Report TR-47, Center for Computational Research in Economics and Management Science, MIT, April 1985.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, volume 1, chapter 8. The MIT Press, Cambridge, MA, 1986.

[8] B. Widrow and M. A. Lehr. 30 years of adaptive neural networks: Perceptron, madaline, and backpropagation. *Proc. IEEE*, pages 1415–1442, September 1990.

[9] D. E. Rumelhart and J. L. McClelland, editors. *Parallel Distributed Processing*, volume 1 and 2. The MIT Press, Cambridge, MA, 1986.

[10] R. V. Southwell. *Relaxation Methods in Engineering Science.* Oxford, New York, 1940.

[11] B. Widrow and S. D. Stearns. *Adaptive Signal Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1985.

[12] L. L. Horowitz and K. D. Senne. Performance advantage of complex lms for controlling narrow-band adaptive arrays. *IEEE Trans. Circuits Systems*, CAS-28(6):562–576, June 1981.

[13] P. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1:339–356, 1988.

[14] D. Nguyen and B. Widrow. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In *Proceedings of the International Joint Conference on Neural Networks*, San Diego, CA, June 1990.

[15] F. J. Pineda. Generalization of backpropagation to recurrent neural networks. *Physical Review Letters*, 18(59):2229–2232, 1987.

[16] L. B. Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Proceedings of the IEEE First International Conference on Neural Networks*, volume II, pages 609–618, San Diego, CA, June 1987.

[17] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. ICS Report 8805, Institute for Cognitive Science, University of California at San Diego, La Jolla, CA 92093, October 1988.

[18] K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, March 1990.

[19] S. A. White. An adaptive recursive digital filter. In *Proc. 9th Asilomar Conf. Circuits Syst. Comput.*, page 21, November 1975.

[20] B. Pearlmutter. Learning state space trajectories in recurrent neural networks. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 113–117, San Mateo, CA, June 17-26 1988. Morgan Kaufmann.

[21] D. Nguyen and B. Widrow. The truck backer-upper: An example of self-learning in neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, volume II, pages 357–363, Washington, DC, June 1989.

[22] B. Widrow and F. Beaufays. Neural control systems. In *Proceedings of the EPRI/INNS Workshop on Neural Network Computing for the Electric Power Industry*, Stanford, Ca, August 17-19 1992.