Feedforward Networks

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

The feedforward network is one of the most widely used neural network paradigms. We present here a brief tutorial on two of the most popular feedforward network training algorithms: LMS and backpropagation.

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 neural networks, which in 1974, he first published in his doctoral dissertation. Unfortunately, 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. Not long after Parker published his findings, Rumelhart, Hinton, and Williams [4] also rediscovered the technique and, largely as a result of the clear framework within which they presented their ideas, they finally 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. More recently, the development of backpropagation has made it possible to attack problems requiring neural networks with high degrees of nonlinearity and precision. Examples are shown in [5]. 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 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 assume 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.

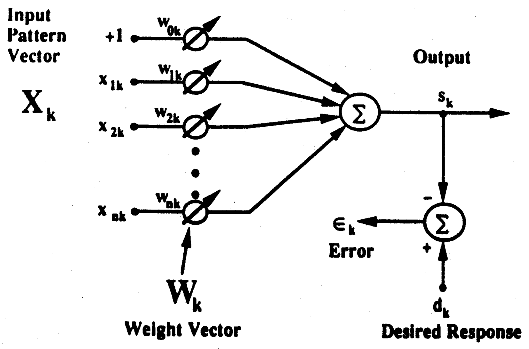

Figure 1: Adaptive linear combiner.

When a nonlinearity is placed on the output of an adaptive linear combiner, we call the cascade an Adaline (ADAptive LInear NEuron). 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, and its output was either +1 or −1. Modern neural networks usually use an "S"-shaped "sigmoid" function, a "soft" quantizer varying smoothly from −1 to +1. An Adaline using this nonlinearity is sometimes referred to as a Sigmoid Adaline. The Adaline usually includes a bias weight w_{0_k} connected to a constant input, $x_0 = +1$. This weight effectively controls the threshold level of the quantizer or sigmoid.

[†]If the neural network was being trained as a pattern classifier, the desired response would be a representation of the class of the training pattern (+1 or −1, for example).

Multilayer Networks

The Madaline networks of the 1960s had an adaptive first layer and a fixed threshold function in the second (output) layer [5]. The feedforward neural networks of today often have many layers, all of which are usually adaptive. The backpropagation networks of Rumelhart *et al.* [4] 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.

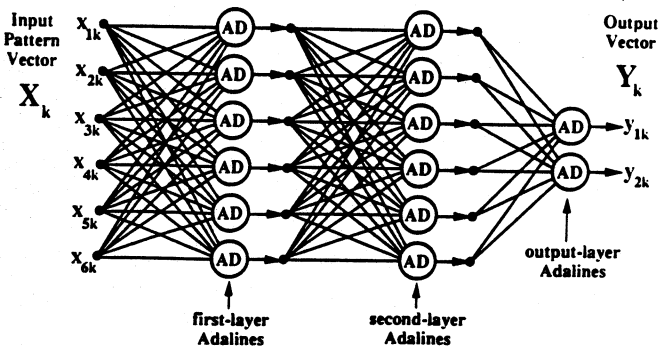

Figure 2: A three-layer adaptive neural network.

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

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 mean-square-error (MSE), averaged over the training set. 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.

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 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 steepest 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 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 upon the square of the instantaneous error is

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

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

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

The instantaneous gradient is used because (a) it is an unbiased estimate of the true gradient [6], 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:

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

Performing the differentiation in Eq. (3) and replacing the linear error by definition (4) gives

$$\mathbf{W}_{k+1} = \mathbf{W}_k - 2\mu\epsilon_k \frac{\partial \epsilon_k}{\partial \mathbf{W}_k} \tag{5}$$

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

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

This is the LMS algorithm. The learning constant μ determines stability and convergence rate [6].

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:

$$y_k = tanh(s_k) = \left(\frac{1 - e^{-2s_k}}{1 + e^{-2s_k}}\right). \tag{7}$$

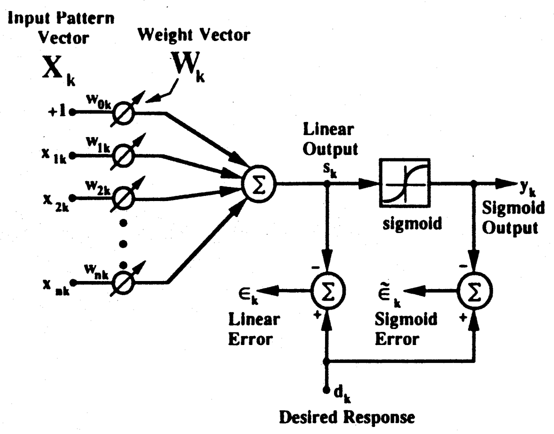

Figure 3: Adaline with sigmoidal nonlinearity.

We shall adapt this Adaline with the objective of minimizing the mean square of the "sigmoid error" $\tilde{\epsilon}_k$, defined as

$$\tilde{\epsilon}_k \triangleq d_k - y_k = d_k - sgm(s_k). \tag{8}$$

The method of steepest descent is again 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

$$\hat{\nabla}_k = \frac{\partial(\tilde{\epsilon}_k)^2}{\partial \mathbf{W}_k} = 2\tilde{\epsilon}_k \frac{\partial \tilde{\epsilon}_k}{\partial \mathbf{W}_k} = -2\tilde{\epsilon}_k sgm'(s_k)\mathbf{X}_k. \tag{9}$$

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

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu(-\hat{\nabla}_k) = \mathbf{W}_k + 2\mu\delta_k\mathbf{X}_k, \tag{10}$$

where δ_k denotes $\tilde{\epsilon}_k sgm'(s_k)$. Algorithm (10) 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 (10) is illustrated in Fig. 4.

If the sigmoid is chosen to be the hyperbolic tangent function (7), then the derivative $sgm'(s_k)$ is given by

$$\begin{aligned} sgm'(s_k) &= \frac{\partial\,(tanh(s_k))}{\partial s_k} \\ &= 1 - (tanh(s_k))^2 = 1 - y_k^2. \end{aligned} \tag{11}$$

Accordingly Eq. (10) becomes

$$\mathbf{W}_{k+1} = \mathbf{W}_k + 2\mu\tilde{\epsilon}_k(1 - y_k^2)\mathbf{X}_k. \tag{12}$$

Backpropagation for networks The basic concepts of backpropagation are easily grasped. Formal derivations are often tedious, however, so here we merely present the algorithm and illustrate how it works for the simple network shown in Fig. 5.

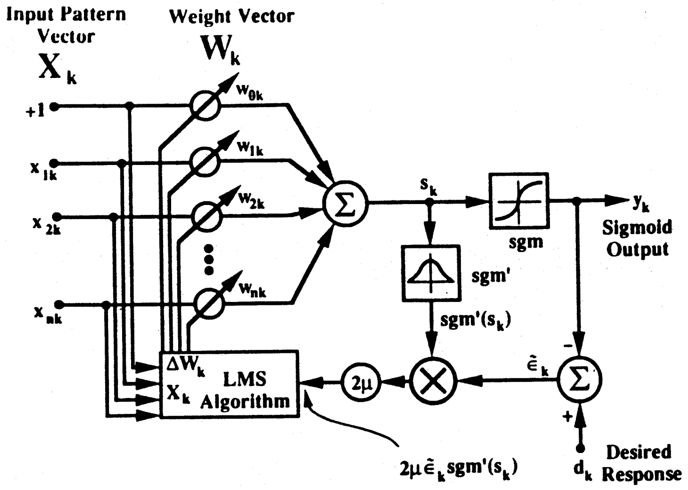

Figure 4: Implementation of backpropagation for the Sigmoid Adaline element.

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 gradient of the sum square error in weight space.

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

$$\varepsilon_k^2 = \sum_{i=1}^{N_y} \epsilon_{ik}^2. \tag{13}$$

In the network example shown in Fig. 5, this is given by

$$\varepsilon_k^2 = (d_{1k} - y_{1k})^2 + (d_{2k} - y_{2k})^2. \tag{14}$$

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 $\mathbf{Y}$, 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 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 values. The algorithm will not work properly with multilayer networks if the initial weights are either zero or poorly chosen nonzero values.

We can see the calculations needed by 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. The solid lines in this diagram represent forward signal paths through the network, and the dotted lines

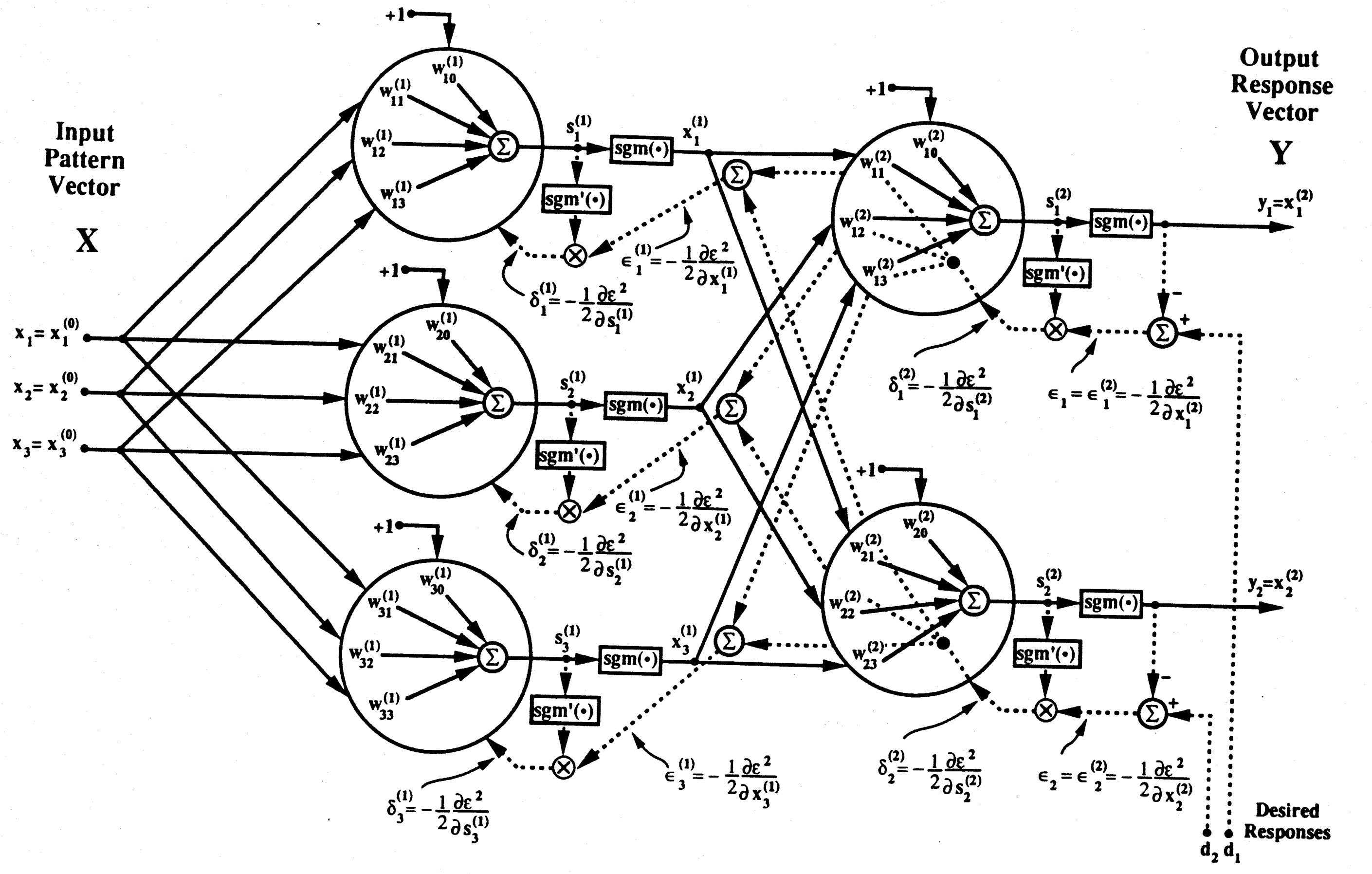
Input Pattern Vector
X
Output Response Vector
Y
Desired Responses
$x_1 = x_1^{(0)}$
$x_2 = x_2^{(0)}$
$x_3 = x_3^{(0)}$
$y_1 = x_1^{(2)}$
$y_2 = x_2^{(2)}$
sgm(•)
sgm'(•)
$\delta_1^{(1)} = -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial s_1^{(1)}}$
$\epsilon_1^{(1)} = -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial x_1^{(1)}}$
$\delta_1^{(2)} = -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial s_1^{(2)}}$
$\epsilon_1 = \epsilon_1^{(2)} = -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial x_1^{(2)}}$
d_2 d_1

Figure 5: Example two-layer backpropagation network architecture.

represent the separate backward paths used in calculation of the square error derivatives δ. We see that the δ's in the output layer are computed just as they are for the Sigmoid Adaline element. Hidden layer calculations, however, are more complicated. The procedure for finding the value of $\delta^{(\ell)}$, the value of δ 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 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 $\epsilon^{(\ell)}$, which, in turn, is multiplied by $sgm'(s^{(\ell)})$, the derivative of the given Adaline's sigmoid function at its current operating point.

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

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu(-\hat{\nabla}_k) = \mathbf{W}_k + 2\mu\delta_k\mathbf{X}_k. \quad (15)$$

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

This brief tutorial is a compressed version of reference [5], which derives backpropagation simply but in much more detail and gives a number of practical applications.

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] 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.

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

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

Above Threshold

International Neural Network Society

Volume 1 ● Number 4 ● December 1992

Call for Papers

DEADLINE: JANUARY 15, 1993

INNS WORLD CONGRESS ON NEURAL NETWORKS

PORTLAND, OREGON, U.S.A.
JULY 11-15, 1993

See pages 4 and 5

In This Issue

1992 INNS Election Results

All the ballots have been counted in the 1992 INNS election. Nearly 900 members (29%, or 897, of INNS's membership) took the opportunity to vote for President-Elect and eight new Board of Governors members. In the President-Elect race, Walter Freeman of the University of California-Berkeley, won over Clifford Lau of the United States Office of Naval Research. Dr. Freeman will serve as President-Elect in 1993 and become the President of INNS in 1994.

Five board members return to office for another three-year term. They are: Shun-ichi Amari, University of Tokyo; James Anderson, Brown University; Gail Carpenter, Boston University; Stephen Grossberg, Boston University; and Christoph von der Malsburg, Ruhr-University Bochum. New board members for 1993 are Leon Cooper, Brown University; John Taylor, King's College-London; and Lotfi Zadeh, University of California-Berkeley. A special feature on the 1993 INNS Officers and Board of Governors begins on page 6.

Small Business Innovation Gets U.S. Congress's Nod

WASHINGTON, DC—Before leaving Washington this fall, the U.S. Congress renewed and expanded the Small Business Innovation Research (SBIR) program (P.L. 102-564), one of the most significant and politically popular sources of federal support for small businesses involved in applied research and development (R&D). Neural networks are typical of the kinds of projects supported by the SBIR, and demonstrations of successful neural network technology have been instrumental in inspiring the widespread support that exists for SBIR in Congress.

Established in 1982, the SBIR program requires federal agencies that spend more than $100 million annually on extramural research or R&D to devote a percentage of those funds for grants or contracts to small businesses conducting innovative applied research on topics relevant to agency missions. In essence, the SBIR provides seed money for higher-risk research that will not attract private investors, with the goal of getting emerging technologies into the marketplace sooner.

SEE **SBIR** ON PAGE 8