

Neural Networks

Ingrid Russell
Department of Computer Science
University of Hartford
West Hartford, CT 06117
irussell@hartford.edu

Introduction

The power and usefulness of artificial neural networks have been demonstrated in several applications including speech synthesis, diagnostic problems, medicine, business and finance, robotic control, signal processing, computer vision and many other problems that fall under the category of pattern recognition. For some application areas, neural models show promise in achieving human-like performance over more traditional artificial intelligence techniques.

What, then, are neural networks? And what can they be used for? Although von-Neumann-architecture computers are much faster than humans in numerical computation, humans are still far better at carrying out low-level tasks such as speech and image recognition. This is due in part to the massive parallelism employed by the brain, which makes it easier to solve problems with simultaneous constraints. It is with this type of problem that traditional artificial intelligence techniques have had limited success. The field of neural networks, however, looks at a variety of models with a structure roughly analogous to that of a set of neurons in the human brain.

The branch of artificial intelligence called neural networks dates back to the 1940s, when McCulloch and Pitts [1943] developed the first neural model. This was followed in 1962 by the *perceptron* model, devised by Rosenblatt, which generated much interest because of its ability to solve some simple pattern classification problems. This interest started to fade in 1969 when Minsky and Papert [1969] provided mathematical proofs of the limitations of the perceptron and pointed out its weakness in computation. In particular, it is incapable of solving the classic exclusive-or (XOR) problem, which will be discussed later. Such drawbacks led to the temporary decline of the field of neural networks.

The last decade, however, has seen renewed interest in neural networks, both among researchers and in areas of application. The development of more-powerful networks, better training algorithms, and improved hardware have all contributed to the revival of the field. Neural-network paradigms in recent years include the Boltzmann machine, Hopfield's network, Kohonen's network, Rumelhart's competitive learning model, Fukushima's model, and Carpenter and Grossberg's Adaptive Resonance Theory model [Wasserman 1989; Freeman and Skapura 1991]. The field has generated interest from researchers in such diverse areas as engineering, computer science, psychology, neuroscience, physics, and mathematics. We describe several of the more important neural models, followed by a discussion of some of the available hardware and software used to implement these models, and a sampling of applications.

Definition

Inspired by the structure of the brain, a neural network consists of a set of highly interconnected entities, called *nodes* or *units*. Each unit is designed to mimic its biological counterpart, the neuron. Each accepts a weighted set of inputs and responds with an output. Figure 1 presents a picture of one unit in a neural network.

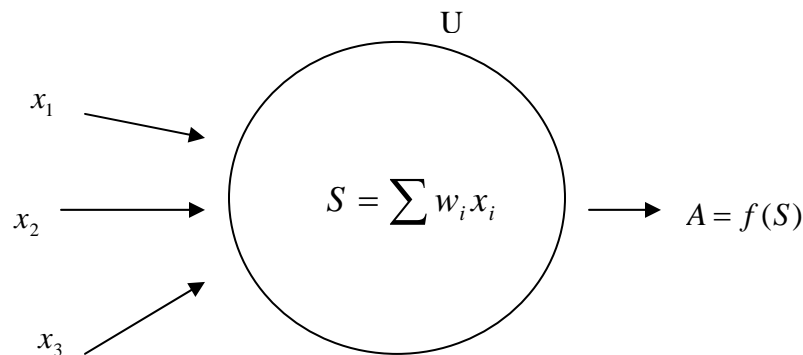


Figure 1. A single unit in a neural network.

Let $\vec{X} = (x_1, x_2, \dots, x_n)$, where the x_i are real numbers, represent the set of inputs presented to the unit U . Each input has an associated weight that represents the strength

of that particular connection. Let $\vec{W} = (w_1, w_2, \dots, w_n)$, with w_i real, represent the weight vector corresponding to the input vector \vec{X} . Applied to U , these weighted inputs produce a net sum at U given by

$$S = \sum w_i x_i = \vec{W} \cdot \vec{V}.$$

Learning rules, which we will discuss later, will allow the weights to be modified dynamically.

The state of a unit U is represented by a numerical value A , the *activation value* of U . An activation function f determines the new activation value of a unit from the net sum to the unit and the current activation value. In the simplest case, f is a function of only the net sum, so $A = f(S)$. The output at unit U is in turn a function of A , usually taken to be just A .

A neural network is composed of such units and weighted unidirectional connections between them. In some neural nets, the number of units may be in the thousands. The output of one unit typically becomes an input for another. There may also be units with external inputs and/or outputs. Figure 2 shows one example of a possible neural network structure.

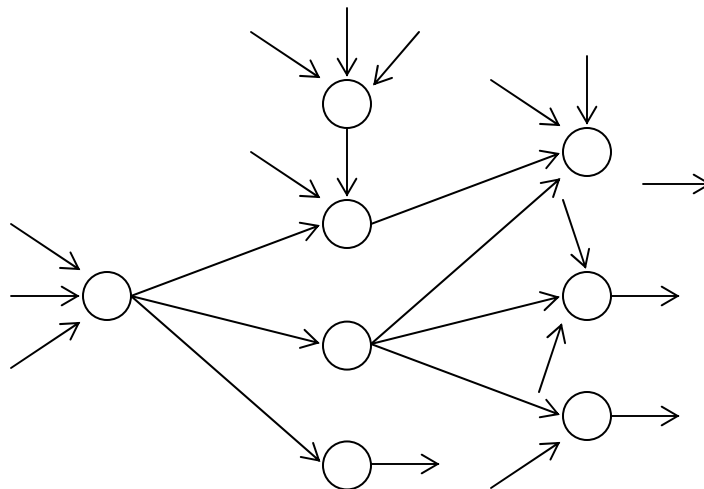


Figure 2. An example of a neural network structure.

For a *simple linear network*, the activation function is a linear function, so that

$$f(cS) = cf(S),$$

$$f(S_1 + S_2) = f(S_1) + f(S_2)$$

Another common form for an activation function is a *threshold function*: the activation value is 1 if the net sum S is greater than a given constant T , and is 0 otherwise.

Single-Layer Linear Networks

A single-layer neural network consists of a set of units organized in a layer. Each unit U_i receives a weighted input x_j with weight w_{ji} . Figure 3 shows a single-layer linear model with m inputs and n outputs.

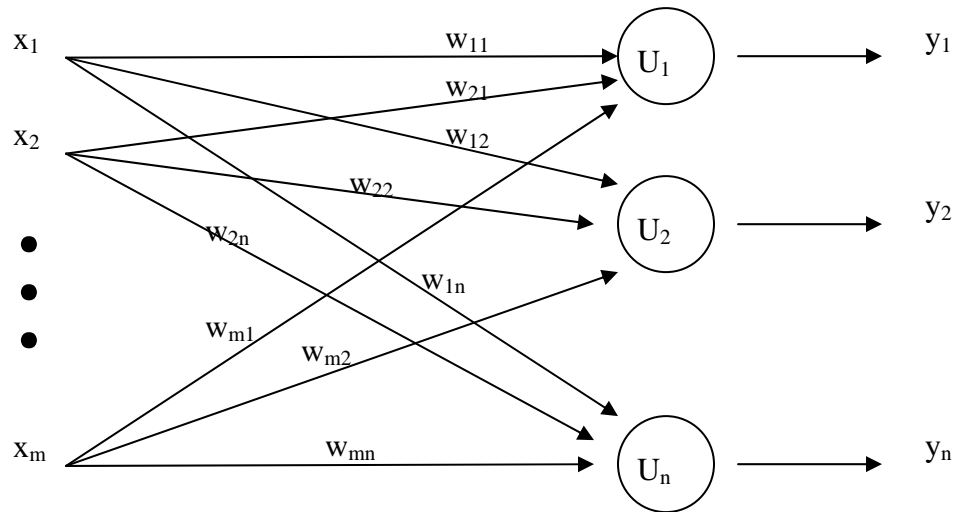


Figure 3. A single-layer linear model.

Let $\vec{X} = (x_1, x_2, \dots, x_m)$ be the input vector and let the activation function f be simply, so that the activation value is just the net sum to a unit. The $m \times n$ weight matrix is

$$W = \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \dots & w_{mn} \end{pmatrix}$$

Thus the output y_k at unit U_k is

$$y_k = (w_{1k}, w_{2k}, \dots, w_{mk}) \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix}$$

So the output vector $\vec{Y} = (y_1, y_2, \dots, y_n)^T$ is given by

$$\vec{Y}_{n \times 1} = W_{m \times n}^T \vec{X}_{m \times 1}$$

Learning Rules

A simple linear network, with its fixed weights, is limited in the range of output vectors it can associate with input vectors. For example, consider the set of input vectors (x_1, x_2) , where each x_i is either 0 or 1. No simple linear network can produce outputs as shown in Table 1, for which the output is the boolean exclusive-or (XOR) of the inputs. (You can easily show that the two weights w_1 and w_2 would have to satisfy three inconsistent linear equations.) Implementing the XOR function is a classic problem in neural networks, as it is a subproblem of other more complicated problems.

Table 1.
Inputs and outputs for a neural net that implements the boolean exclusives (XOR) function.

x_1	x_2	output y
0	0	0
0	1	1
1	0	1
1	1	0

Hence, in addition to the network topology, an important component of most neural networks is a *learning rule*. A learning rule allows the network to adjust its connection weights in order to associate given input vectors with corresponding output vectors. During training periods, the input vectors are repeatedly presented, and the weights are adjusted according to the learning rule, until the network learns the desired

associations, i.e., until $\vec{Y} = W^T \vec{X}$. It is this ability to learn that is one of the most attractive features of neural networks.

A single-layer model usually uses either the *Hebb rule* or the *delta rule*. In the Hebb rule, the change δw_{ij} in the weights is calculated as follows. Let $\vec{X}(x_1, \dots, x_m)$, $\vec{Y}(y_1, \dots, y_n)^T$ be the input and output vectors that we wish to associate. In each training iteration, the weights are adjusted by

$$\delta w_{ij} = e x_i y_j,$$

where e is a constant called the *learning rate*, usually taken to be the reciprocal of the number of training vectors presented. During the training period, a number of such iterations can be made, letting the (\vec{X}, \vec{Y}) pairs vary over the associations to be learned. A network using the Hebb rule is guaranteed (by mathematical proof) to be able to learn associations for which the set of input vectors are orthogonal. [McClelland and Rumelhart et al. 1986]. A disadvantage of the Hebb rule is that if the input vectors are not mutually orthogonal, interference may occur and the network may not be able to learn the associations.

The delta rule was developed to address the deficiencies of the Hebb rule. Under the delta rule, the change in weight is

$$\delta w_{ij} = r x_i (t_j - y_j)$$

Where

r is the learning rate,

t_j is the target output, and

y_j is the actual output at unit U_j .

The delta rule changes the weight vector in a way that minimizes the *error*, the difference between the target output and the actual output. It can be shown mathematically that the delta rule provides a very efficient way to modify the initial weight vector toward the optimal one (the one that corresponds to minimum error) [McClelland and Rumelhart et al. 1986]. It is possible for a network to learn more associations with the delta rule than with the Hebb rule. McClelland and Rumelhart et al.

prove that a network using the delta rule can learn associations whenever the inputs are linearly independent [1986].

Threshold Networks

Much early work in neural networks involved the *perceptron*. Devised by Rosenblatt, a perceptron is a single-layer network with an activation function given by

$$f(S) = \begin{cases} 1 & \text{if } S > T \\ 0 & \text{otherwise} \end{cases}$$

where T is some constant. Because it uses a threshold function, such a network is called a *threshold network*.

But even though it uses a nonlinear activation function, the perceptron still cannot implement the XOR function. That is, a perceptron is not capable of responding with an output of 1 whenever it is presented with input vectors (0,1) or (1,0), and responding with output 0 otherwise.

The impossibility proof is easy. There would have to be a weight vector for which $\vec{W} = (w_{11}, w_{12})$ for which the scalar product net sum

$$S = \vec{W} \cdot \vec{X} = w_{11}x_1 + w_{21}x_2$$

leads to an output of 1 for input (0,1) or (1,0), and 0 otherwise (see Table 2).

Table 2.
Inputs, net sum, and desired output for a perceptron that implements the boolean exclusives (XOR) function.

x_1	x_2	S	desired output
0	0	0	0
0	1	w_2	1
1	0	w_1	1
1	1	$w_1 + w_2$	0

Now, the line with equation $w_{11}x_1 + w_{21}x_2 = T$ divides the x_1x_2 -plane into two regions, as illustrated in **Figure 4**. Input vectors that produce a net sum S greater than T lie on one side of the line, while those with net sum less than T lie on the other side. For the network to represent the XOR function, the inputs (1,1) and (0, 0), with sums (w_1+w_2)

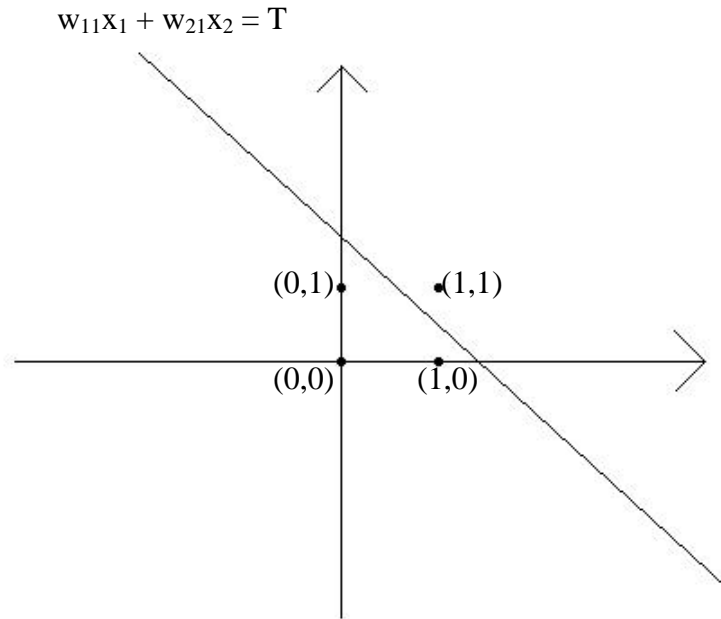


Figure 4. The graph of $w_{11}x_1 + w_{21}x_2 = T$.

and 0, must produce outputs on one side, while the inputs $(1, 0)$ and $(0, 1)$, with sums w_1 and w_2 , must produce outputs on the other side. But if $w_1 > T$ and $w_2 > T$, then $w_1 + w_2 > T$; and similarly for $<$. So a perceptron cannot represent the XOR function.

In fact, there are many other functions that cannot be represented by a single-layer network with fixed weights. While such limitations were the cause of a temporary decline of interest in the perceptron and in neural networks in general, the perceptron laid foundations for much of the later work in neural networks. The limitations of single-layer networks can, in fact, be overcome by adding more layers; as we will see in the following section, there is a multilayer threshold system that can represent the XOR function.

Multilayer Networks

A multilayer network has two or more layers of units, with the output from one layer serving as input to the next. The layers with no external output connections are referred to as *hidden* layers (Figure 5).

However, any multilayer system with fixed weights that has a linear activation function is equivalent to a single-layer linear system. Take, for example, the case of a two-layer linear system. The input vector to the first layer is \vec{X} , the output $\vec{Y} = W_1\vec{X}$ of the first layer is given as input to the second layer, and the second layer produces output $\vec{Z} = W_2\vec{Y}$.

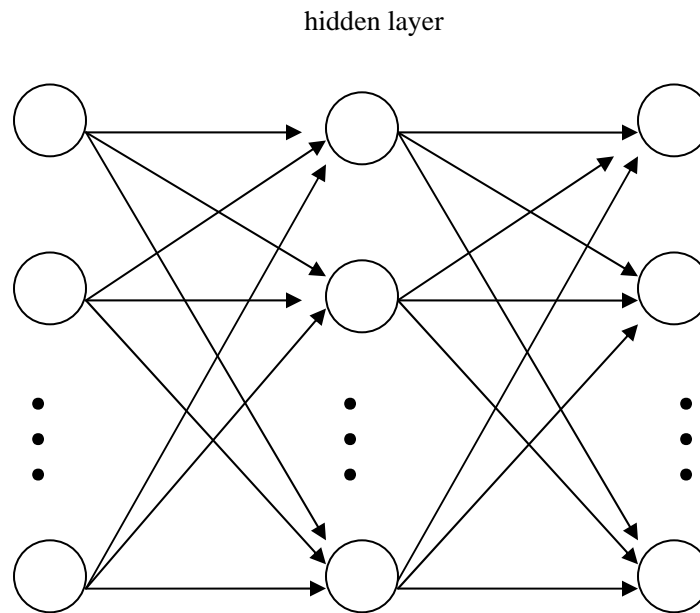


Figure 5. A multilayer network.

Hence

$$\vec{Z} = W_2(W_1\vec{X}) = (W_2W_1)\vec{X}$$

Consequently, the system is equivalent to a single-layer network with weight matrix $W = W_2W_1$. By induction, a linear system with any number n of layers is equivalent to a single-layer linear system whose weight matrix is the product of the n intermediate weight matrices.

On the other hand, a multilayer system that is not linear can provide more computational capability than a single-layer system. For instance, the problems encountered by the perceptron can be overcome with the addition of hidden layers; **Figure 6** demonstrates how a multilayer system can represent the XOR function. The threshold is set to zero, and consequently a unit responds if its activation is greater than zero.

The weight matrices for the two layers are

$$W_1 = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \quad W_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

We thus get

$$\begin{aligned} W_1^T \begin{pmatrix} 1 \\ 0 \end{pmatrix} &= \begin{pmatrix} 1 \\ 0 \end{pmatrix}, & W_2^T \begin{pmatrix} 1 \\ 0 \end{pmatrix} &= 1, \\ W_1^T \begin{pmatrix} 0 \\ 1 \end{pmatrix} &= \begin{pmatrix} 0 \\ 1 \end{pmatrix}, & W_2^T \begin{pmatrix} 0 \\ 1 \end{pmatrix} &= 1, \\ W_1^T \begin{pmatrix} 1 \\ 1 \end{pmatrix} &= \begin{pmatrix} 0 \\ 0 \end{pmatrix}, & W_2^T \begin{pmatrix} 0 \\ 0 \end{pmatrix} &= 0, \\ W_1^T \begin{pmatrix} 0 \\ 0 \end{pmatrix} &= \begin{pmatrix} 0 \\ 0 \end{pmatrix}, & W_2^T \begin{pmatrix} 0 \\ 0 \end{pmatrix} &= 0. \end{aligned}$$

With input vector (1,0) or (0,1), the output produced at the outer layer is 1; otherwise it is 0.

Multilayer networks have proven to be very powerful. In fact, any boolean function can be implemented by such a network [McClelland and Rumelhart 1988].

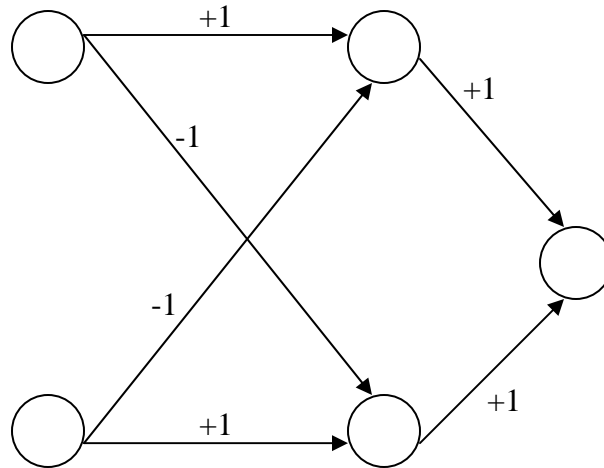


Figure 6. A multilayer system representation of the XOR function.

Multilayer networks have proven to be very powerful. In fact, any boolean function can be implemented by such a network [McClelland and Rumelhart 1988].

Multilayer Networks with Learning

No learning algorithm had been available for multilayer networks until Rumelhart, Hinton, and Williams introduced the *backpropagation training algorithm*, also referred to as the *generalized delta rule* [1988]. At the output layer, the output vector is compared to the expected output. If the difference is zero, no changes are made to the weights of the connections. If the difference is not zero, the error is calculated from the delta rule and is propagated back through the network. The idea, similar to that of the delta rule, is to adjust the weights to minimize the difference between the real output and the expected output. Such networks can learn arbitrary associations by using differentiable activation functions. A theoretical foundation of backpropagation can be found in McClelland and Rumelhart et al. [1986] and in Rumelhart et al. [1988].

One drawback of backpropagation is its slow rate of learning, making it less than ideal for real-time use. In spite of some drawbacks, backpropagation has been a widely used algorithm, particularly in pattern recognition problems.

All the models discussed so far use supervised learning, i.e., the network is provided the expected output and trained to respond correctly. Other neural network models employ unsupervised learning schemes. Unsupervised learning implies the absence of a trainer and no knowledge beforehand of what the output should be for any given input. The network acts as a regularity detector and tries to discover structure in the patterns presented to it. Such networks include competitive learning, for which there are four major models [Wasserman 1989; Freeman and Skapura 1991; McClelland and Rumelhart et al. 1986].

Software and Hardware Implementation

It is relatively easy to write a program to simulate one of the networks described in the preceding sections (see, e.g., Dewdney [1992]); and a number of commercial software packages are available, including some for microcomputers. Many programs feature a neural-network development system that supports several different neural types, to allow the user to build, train, and test networks for different applications. Reid and Zeichick provide a description of 50 commercial neural-network products, as well as pricing information and the addresses of suppliers [1992].

The training of a neural network through software simulation demands intensive mathematical computation, often leading to excessive training times on ordinary general-purpose processors. A neural network accelerator board, such as the NeuroBoard developed to support the NeuroShell package, can provide high-speed performance. NeuroBoard's speed is up to 100 times that of a 20 MHz 30386 chip with a math co-processor.

Another alternative is a chip that implements neural networks in hardware; both analog and digital implementations are available. Carver Mead at UCLA, a leading researcher in analog neural-net chips, has developed an artificial retina [1989]. Two companies lead in commercialized neural network chip development: Intel, with its 80170 ETANN (Electronically Trainable Artificial Neural Network) chip, and Neural

Semiconductor, with its DNNA (Digital Neural Network Architecture) chip. These chips, however, do not have the capabilities of on-chip learning. In both cases, the chip is interfaced with a software simulation package, based on backpropagation, which is used for training and adjustment of weights; the adjusted weights are then transferred to the chip [Caudill 1991]. The first chips with on-chip training capability should be available soon.

Applications

Neural networks have been applied to a wide variety of different areas including speech synthesis, pattern recognition, diagnostic problems, medical illnesses, robotic control and computer vision.

Neural networks have been shown to be particularly useful in solving problems where traditional artificial intelligence techniques involving symbolic methods have failed or proved inefficient. Such networks have shown promise in problems involving low-level tasks that are computationally intensive, including vision, speech recognition, and many other problems that fall under the category of pattern recognition. Neural networks, with their massive parallelism, can provide the computing power needed for these problems. A major shortcoming of neural networks lies in the long training times that they require, particularly when many layers are used. Hardware advances should diminish these limitations, and neural-network-based systems will become greater complements to conventional computing systems.

Researchers at Ford Motor Company are developing a neural-network system that diagnoses engine malfunctions. While an experienced technician can analyze engine malfunction given a set of data, it is extremely complicated to design a rule-based expert system to do the same diagnosis. Marko et al. [1990] trained a neural net to diagnose engine malfunction, given a number of different faulty states of an engine such as open plug, broken manifold, etc. The trained network had a high rate of correct diagnoses. Neural nets have also been used in the banking industry, for example, in the evaluation of credit card applications.

Most neural network applications, however, have been concentrated in the area of pattern recognition, where traditional algorithmic approaches have been ineffective. Such

nets have been used for classifying a given input into one of a number of categories and have demonstrated success, even with noisy input, when compared to other more conventional techniques.

Since the 1970s, work has been done on monitoring the Space Shuttle Main Engine (SSME), involving the development of an Integrated Diagnostic System (IDS). The IDS is a hierarchical multilevel system, which integrates various fault detection algorithms to provide a monitoring system that works for all stages of operation of the SSME. Three fault-detection algorithms have been used, depending on the SSME sensor data. These employ statistical methods that have a high computational complexity and a low degree of reliability, particularly in the presence of noise. Systems based on neural networks offer promise for a fast and reliable real-time system to help overcome these difficulties, as is seen in the work of Dietz et al. [1989]. This work involves the development of a fault diagnostic system for the SSME that is based on three-layer backpropagation networks. Neural networks in this application allow for better performance and for the diagnosis to be accomplished in real time. Furthermore, because of the parallel structure of neural networks, better performance is realized by parallel algorithms running on parallel architectures.

At Boeing Aircraft Company, researchers have been developing a neural network to identify aircraft parts that have already been designed and manufactured, in efforts to help them with the production of new parts. Given a new design, the system attempts to identify a previously designed part that resembles the new one. If one is found, it may be able to be modified to conform to the new specifications, thus saving time and money in the manufacturing process.

Neural networks have also been used in biomedical research, which often involves the analysis and classification of an experiment's outcomes. Traditional techniques include the linear discriminant function and the analysis of covariance. The outcome of the experiment is in some cases dependent on a number of variables, with the dependence usually a nonlinear function that is not known. Such problems can, in many cases, be managed by neural networks.

Stubbs [1990] presents three biomedical applications in which neural networks have been used, one of which involves drug design. Non-steroidal anti-inflammatory

drugs (NOSAIIDs) are a commonly prescribed class of drugs, which in some cases may cause adverse reactions. The rate of adverse reactions (ADR) is about 10%, with 1% of these involving serious cases and 0.1% being fatal [Stubbs 1990]. A three-layer backpropagation neural network was developed to predict the frequency of serious ADR cases for 17 particular NOSAIIDs, using four inputs, each representing a particular property of the drugs. The predicted rates given by the model matched within 5% the observed rates, a much better performance than by other techniques. Such a neural network might be used to predict the ADR rate for new drugs, as well as to determine the properties that tend to make for "safe" drugs.

Conclusion

In the early days of neural networks, some overly optimistic hopes for success were not always realized, causing a temporary setback to research. Today, though, a solid basis of theory and applications is being formed; and the field has begun to flourish. For some tasks, neural networks will never replace conventional methods; but for a growing list of applications, the neural architecture will provide either an alternative or a complement to these other techniques.

References

- Carpenter, G., and S. Grossberg. 1988. The ART of Adaptive Pattern Recognition by a Self-organizing Neural Network. *IEEE Computer* 21: 77-88.
- Caudill, M. 1990. Using neural nets: Diagnostic expert nets. *AI Expert* 5 (9) (September 1990): 43-47.
- , 1991. Embedded neural networks. *AI Expert* 6 (4) (April 1991): 40-45.
- Denning, Peter J. 1992. The science of computing: Neural networks. *American Scientist* 80: 426-429.
- Dewdney, A.K. 1992. Computer recreations: Programming a neural net. *Algorithm: Recreational Computing* 3 (4) (October-December 1992): 11-15.
- Dietz, W., E. Kiech, and M. Ali. 1989. Jet and rocket engine fault diagnosis in real time. *Journal of Neural Network Computing* (Summer 1989): 5-18.
- Freeman, J., and D. Skapura. 1991. *Neural Networks*. Reading MA: Addison-Wesley.

Fukushima, K. 1988. A neural network for visual pattern recognition. *IEEE Computer* 21 (3) (March 1988): 65-75.

Kohonen, T. 1988. *Self-Organization and Associative Memory*. New York: Springer-Verlag.

Marko, K., J. Dossdall, and J. Murphy. 1990. Automotive control system diagnosis using neural nets for rapid pattern classification of large data sets. In *Proceedings of the International Joint Conference on Neural Networks* I-33-I-38. Piscataway, NJ: IEEE Service Center.

McClelland, J., D. Rumelhart, and the PDP Research Group. 1986. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations*. Cambridge, MA: MIT Press.

McClelland, J., and D. Rumelhart. 1988. *Explorations in Parallel Distributed Processing*. Cambridge, MA: MIT Press.

McCulloch, W., and W. Pitts. 1943. A logical calculus of the ideas imminent in nervous activity. *Bulletin of Mathematical Biophysics* 5: 115-33.

Mead, C. 1989. *Analog VLSI and Neural Systems*. Reading MA: Addison-Wesley.

Minsky, M., and S. Papert. 1969. *Perceptrons*. Cambridge, MA: MIT Press.

Reid, K., and A. Zeichick. 1992. Neural network products resource guide. *AI Expert* 7 (6) (June 1992): 50-56.

Rumelhart, D., G. Hinton, and R. Williams. 1988. Learning internal representations by error propagation. In *Neurocomputing*, edited by J. Anderson and E. Rosenfeld, 675-695. Cambridge, MA: MIT Press.

Russell, I. 1991. Self-organization and adaptive resonance theory networks. In *Proceedings of the Fourth Annual Neural Networks and Parallel Processing Systems Conference*, edited by Samir I. Sayegh, 227-234. Indianapolis, IN: Indiana University-Purdue University.

Shea, P., and V. Lin. 1989. Detection of explosives in checked airline baggage using an artificial neural system. *International Journal of Neural Networks* 1 (4) (October 1989): 249-253.

Stubbs, D. 1990. Three applications of neurocomputing in biomedical research. *Neurocomputing* 2: 61-66.

Wasserman, P. 1989. *Neural Network Computing*. New York: Van Nostrand Reinhold.