Neural Networks in Structural Engineering

R. D. VANLUCHENE and ROUFEI SUN
Montana State Univesity

In the past few years literature on computational civil engineering has concentrated primarily on artificial intelligence (AI) applications involving expert system technology. This article discusses a different AI approach involving neural networks. Unlike their expert system counterparts, neural networks can be trained based on observed information. These systems exhibit a learning and memory capability similar to that of the human brain, a fact due to their simplified modeling of the brain's biological function. This article presents an introduction to neural network technology as it applies to structural engineering applications. Differing network types are discussed. A back-propagation learning algorithm is presented. The article concludes with a demonstration of the potential of the neural network approach. The demonstration involves three structural engineering problems. The first problem involves pattern recognition; the second, a simple concrete beam design; and the third, a rectangular plate analysis. The pattern recognition problem demonstrates a solution which would otherwise be difficult to code in a conventional program. The concrete beam problem indicates that typical design decisions can be made by neural networks. The last problem demonstrates that numerically complex solutions can be estimated almost instantaneously with a neural network.

INTRODUCTION

Since physicist John Hopfield published his paper on neural networks and physical systems with emergent collective computational abilities [5], interest in neural networks has grown exponentially. Because of the technology's enormous potential, an increased number of researchers have concentrated their efforts on neural networks. Several articles have summarized neural networks and their potential [2, 10, 11]. Many impressive results have been produced [4].

Neural networks have been used to successfully solve the "traveling salesman problem," finding the shortest route through a group of cities [12]. This problem has also been solved by optimization techniques. Another application of a neural network, the airline marketing tactician, is used to maximize airline profits and has attracted the attention of several major airlines. Neural networks are presently being developed to convert written text to speech and to recognize human speech and translate it into written text. In the area of image recognition, an associative memory network can identify a partially obscured face from a memory of 500 different faces. Another neural network that has been created can identify hand-printed characters with 95% accuracy. For the problems solved above, a basic algorithmic computer program would be extremely difficult to develop.

With so many breakthroughs in AI, what are neural networks? Neural networks are computing systems that simulate the structure of the biological neural network of the human brain. Based on research in neurophysiology, a human brain has approximately 100 billion neural cells interconnected in a complex manner constituting a large scale network. A typical biological neuron model is shown in Figure 1. When a biological neuron is excited by electrical inputs, it sends out pulses through its axon. The soma sums the electrical potentials between its dendrites and utilizes these to output a voltage spike along its axon. The voltage output may be modeled via a sigmoid function. This output voltage is biologically transmitted to a part of the body associated with the particular neuron, for example, a muscle in an arm. The muscle will then move accordingly.

Neural networks are highly simplified models of the human neural system [3]. Figure 2 shows a mimic model of neural nets. Here V_1, V_2, and V_3 represent the output voltage of neurons J_1, J_2, and J_3, respectively, through connection axons. Neuron I sums V_1, V_2, V_3 according to the synaptic connections (weight) W_{ij} (between upper-

Address correspondence to: R. D. VanLuchene, Department of Civil Engineering, Montana Satate University, Bozeman, MT 59717.

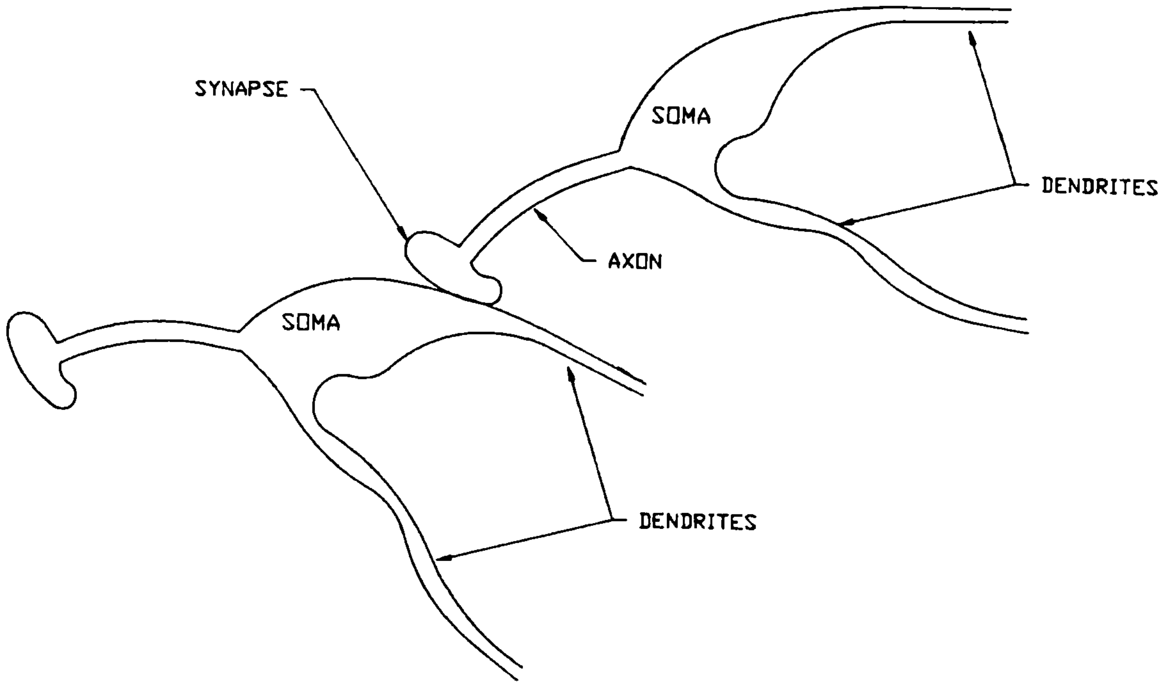


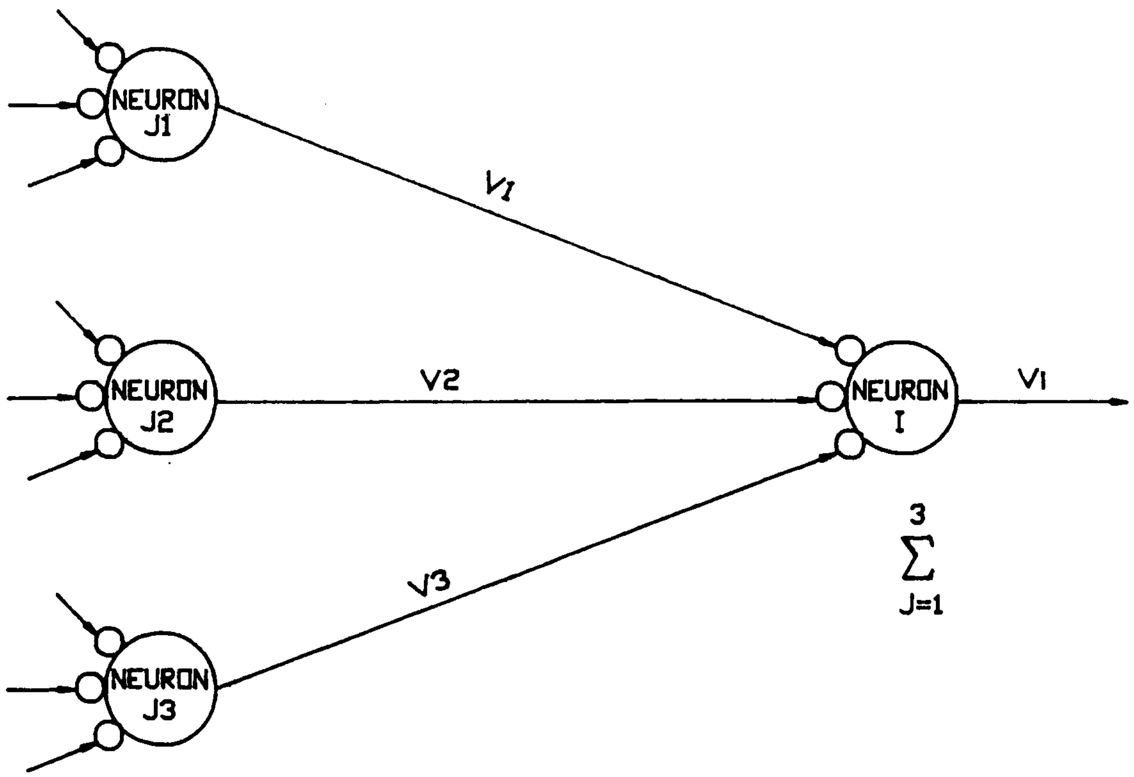
FIGURE 1. Biological neuron model.

layer neuron *i* and lower-layer neuron *j*) and outputs at a magnitude based on a function similar to the sine function evaluated between 0 and $\pi/2$ radians.

Figure 3 shows the basic types of artificial neural networks. The types range from the simple single neuron to complex multilayer nets. There is also a correspondence in the difficulty of problems that can be solved by these network types, that is from simple to complex. The ways in which neurons in these networks may be connected are shown in Figure 4.

One possible method for artificial neural networks to simulate biological neural networks is through the fol-

FIGURE 2. Mimic model.



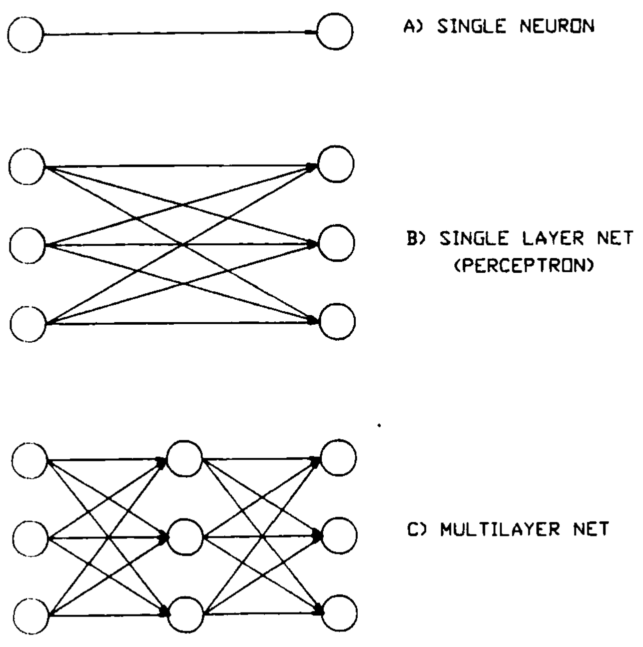


FIGURE 3. Neural network types.

lowing four-step process [6]:

1. An input is received at layer j based on the weight W_{ij} times the output of the connected neuron in layer i. The value of W_{ij} indicates the relationship between the two connected neurons. The process of establishing the connection weights is called training and involves a learning process for the network.
2. A summation which is carried out combines all the inputs together (see Equation 3 below).
3. An output value is calculated by using a threshold function or transfer function. Common threshold functions include sigmoid functions, linear functions, step functions, trigonometric functions, and Gaussian functions.

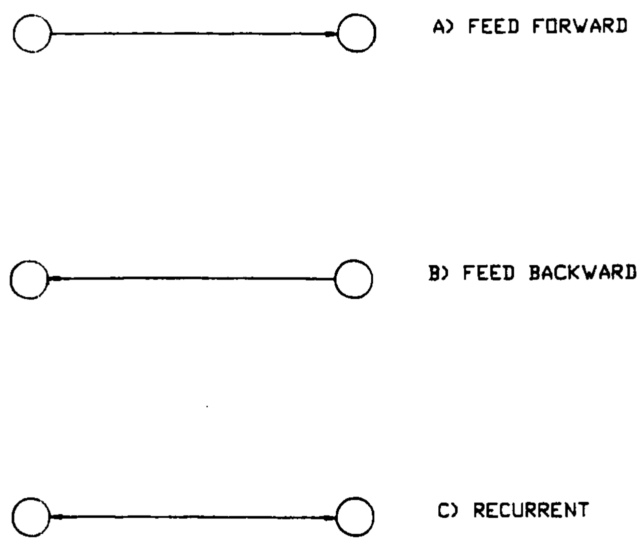


FIGURE 4. Interconnection types.

4. The output value is sent to the connected units in the next layer. This output will then be multiplied by a weighting factor and serve as the input for the next layer of neurons.

SINGLE-LAYER NETWORKS

There are several important neural network models, such as Hopfield net, Hamming net, Carpenter/Grossberg net, single-layer perceptron, multilayer network, etc. [7]. As an example, the single-layer Hopfield and Hamming nets are normally used with a binary input and output. They can be used as associative memories to solve problems such as pattern recognition. Binary nets can also be used to solve optimization problems. The weights in a Hopfield net are set using the given example patterns in a training stage. Input for a known pattern is introduced to the network. Output of each neuron is fed back to all other neurons via the connection weights. The output pattern is compared with the known pattern, and weights are adjusted if necessary. This process of feeding back neuron output then iterates until the outputs remain nearly unchanged over two iterations [12]. The Hamming net differs from the Hopfield net in that it employs a subnet, which is used to optimize minimum errors.

In the Hopfield and Hamming nets, training is supervised; that is, correct solutions are provided and used to establish connection weights. The Carpenter/Grossberg net, however, uses unsupervised training. Correct solutions are not provided to the network. These nets are used primarily to form clusters valuable for pattern recognition. Single-layer binary networks such as the Hopfield, Hamming, and Carpenter/Grossberg nets work well for problems involving binary input and output. A typical problem of this type is pattern recognition of black and white images. The single-layer perceptron network, on the other hand, can be used with multivalue input and output in addition to binary data. Recent work indicates that the single-layer perceptron can be used to simulate structural steel design experience [1].

A serious disadvantage of the single-layer network is that complex decisions may not be possible. It has been shown that the decision region of a single-layer network is bounded by a hyperplane, whereas a two-layer network may have open or closed convex decision regions [7]. The three-layer network decision region is arbitrary and the complexity is limited only by the number of nodes in the net. To summarize, single-layer networks may not have the decision-making complexity necessary for solving a broad range of civil engineering problems.

MULTILAYER NETWORKS

Probably the most important and powerful nets are multilayer networks that have been developed from single-layer networks. Most multilayer networks have a more sophisticated learning algorithm called back propagation. This method involves sending the input forward through the network and then comparing the output with the required training pattern output. If differences exist, a set of changes are applied to the weighting factors in a backward propagating manner. This method can be used to solve a number of different problems such as machine learning.

Training a multilayer net is done in two steps: feed forward and back propagation. Feed forward involves sending the weighted output of the lower-layer neurons to the upper-layer neurons. Back propagation propagates the error between the actual output and the desired output back to adjust the interconnection weights W_{ij}. The change of weights for training pattern P is given by [11]:

$$\Delta_p W_{ij} = \eta^*(t_{pi} - o_{pi})^* X_{pj} = \eta * \delta_{pi}^* X_{pj}, \tag{1}$$

where $\Delta_p W_{ij}$ is the change to be considered to W_{ij} for pattern P. The quantity η is the learning ratio and is used to dampen out predicted changes in weighting factors. For the examples presented in this article, a value of 0.95 for the learning ratio improves convergence during the training phase. $\delta_{pi} = t_{pi} - o_{pi}$ is the difference between the desired output t_{pi} and actual output o_{pi} for pattern P at neuron i. X_{pj} is the input from neuron j. The main problem here is to find the minimum error between the actual output and the desired output. Defining the total error for training pattern P as E_p, the "hill-climbing" or gradient descent method may be used to minimize E_p [11]. Equation (2) provides an expression for E_p [11].

$$E_p = 1/2 \sum \delta_{pi}^2. \tag{2}$$

Training begins by initializing weights. This may be done randomly or by assigning values input by the user. Each pattern is then fed forward to get the output based on the equation:

$$X_{pi} = \sum W_{ij}^* X_{pj} . \tag{3}$$

X_{pi} is the value for the neuron in the upper layer, X_{pj} is the value for the neuron in the lower layer, and W_{ij} is the connection weight between neuron i and neuron j. By applying a sigmoid threshold or transfer function of the form:

$$f(X_{pi}) = \frac{1}{1 + e^{-(X_{pi} - \theta_i)}}, \tag{4}$$

the output value of the neuron may be calculated. This output value is sent on to the next layer of neurons. At the last (output) layer, for pattern P, $o_{pi} = X_{pi} = f(X_{pi})$. The quantity θ_i in Equation (4) is termed the solution bias or noise and its calculation is described below. The bias acts as a function shifting term which can improve overall network accuracy.

Back propagation starts from the last (output) layer. The adjustment of W_{ij} is $\Delta p W_{ij}$ as given by Equation (1). The quantity δ_{pi} may be derived by using the partial derivative of the error E_p with respect to the network output o_{pi} of Equation (3). In the output layer, $\delta_{pi} = f'(X_{pi})^*(t_{pi} - o_{pi})$, and in other layers, $\delta_{pi} = f'(X_{pi}) \sum \delta_{pk}^* W_{ki}$. After the weights in the lowest layer have been adjusted, the next pattern is inputted and the process is repeated. The whole training procedure is repeated until weights converge and the error of outputs is reduced to an acceptable value.

The back propagation discussed above involves finding the minimum error of each pattern, E_p. Another method may be used whereby the minimum error of all patterns, $\sum E_p$, is found. The change in W_{ij} is made after one complete cycle of pattern presentations. If there are a number of patterns for training, this method will require more computer memory [9].

The critical problem in training is convergence. Research results indicate that convergence may be accelerated if a momentum smoothing term is added [2]:

$$W_{ij}(t + 1) = W_{ij}(t) + \eta^* \delta_{pi}^* X_{pj} + \alpha^* \Delta_p W_{ij}(t), \tag{5}$$

where α is a momentum smoothing factor. Factors η and α influence convergence. In addition, our work indicates that convergence is faster if initial weights are selected randomly (using a FORTRAN random number generator) in the range of $-1 < W_{ij} < +1$ rather than the usual range of $0 < W_{ij} < +1$. Because of this random initialization process, networks trained with the same data on two different days will not produce exactly the same results. Our experience indicates that the daily differences for the examples mentioned below, however, are quite small.

Another important factor influencing convergence is the bias or noise factor, θ_i. There are two ways to obtain θ_i. One is to simply assign it to a small value and not change it during the training. The other is to initially select small random values for θ_i and change them during training. The process for changing the biases is based on training using the same gradient descent method used for establishing the weights. The method involves simply imagining that θ_i is the weight from a neuron that is always turned on [11]. Bias corrections involve an equation very similar to Equation (1) above. In the system developed for the examples which follow, both bias techniques are allowed. It is hard to say which method is more effective. Our work indicates that for some problems, assigning θ_i and holding it constant is preferable. For other problems, however, varying the bias may be more effective.

NNICE DEVELOPMENT AND SAMPLE PROBLEMS

A test program, NNICE (Neural Network In Civil Engineering), based on the algorithm of back-propagation training has been developed. NNICE is written in FOR-

TRAN (1977 Standard) due to the widespread acceptance and use of the language. NNICE allows selection of the number of layers used, the number of input neurons, the number of neurons on interior layers (hidden neurons), the number of output neurons, the means for selecting θ_i, and the smoothing factor α. The learning ratio, η, has been held constant at 0.95. NNICE is presently being run on a Digital Equipment C8 VAX minicomputer and an IBM-compatible personal computer. The goal for NNICE is to develop a package that may be widely used to solve many problems in civil engineering.

NNICE has been tested on three widely varied problems. One is a simple beam load location problem, another the cross-section selection of reinforced concrete beams, and the third involves analysis of a simply supported plate. These problems were chosen as typical of structural engineering problems. They were also chosen to determine if NNICE may be capable of learning to solve engineering problems in which analytical solutions exist.

LOAD LOCATION PROBLEM

The load location problem involved bending moments in a simply supported beam with a concentrated load. The beam length was 1 unit and a concentrated load with value of 4.05 was applied at discrete points along its length. For calculation purposes, the beam was divided into nine parts, giving 10 section output points at 0/9, 1/9, 2/9, . . . 9/9 from the left end. A three-layer network was used and had two hidden layers, 10 input neurons (one for the moment at each of the interval points), 10 neurons at each hidden layer, and one output neuron representing the fractional location of the load (from the left end) producing the input moments. The input data were moments at each interval point along the beam with the unit load applied at points 0.2, 0.4, 0.6, and 0.8 units from the left end. These data are shown in Table 1. The output data were the interval points indicating where the load was placed to produce the input pattern.

Training using the four patterns above took about 40 seconds on an IBM compatible personal computer. Considering the capabilities of the resulting network, this training effort seems highly justified. Eight input patterns were then used to test the network. The bias was "trained" by NNICE for this problem. The results are shown in Table 2. An important observation is that although patterns 5 through 8 were not used for training, the results were very close to the exact solution except for pattern 8. While it is not implied that this network should be used to replace conventional simple beam methods, it does demonstrate the pattern recognition capabilities of neural networks.

TABLE 2 RESULTS FOR SIMPLE BEAM NETWORK

	Actual Output	*Desired Output*	*Percent Error (%)*
Pattern 1	0.207	0.200	3.5
Pattern 2	0.398	0.400	0.5
Pattern 3	0.607	0.600	1.2
Pattern 4	0.794	0.800	0.8
Pattern 5	0.271	0.300	9.7
Pattern 6	0.503	0.500	0.6
Pattern 7	0.746	0.700	6.6
Pattern 8	0.629	0.900	30.1

CONCRETE BEAM PROBLEM

The concrete beam problem involved inputting the bending moment M_u, the reinforcing steel strength f_y, the concrete compressive strength f'_c, the reinforcing ratio ρ, and the rectangular section's width to depth ratio b/d. Output consisted of the required member depth d. This is an elementary problem in reinforced concrete design [8].

Test results of the concrete beam selection problem were impressive. A network with five input neurons, two hidden layers of six neurons each, and one output neuron was used. Training took place with 21 randomly chosen patterns. Testing involved 31 patterns, which included 10 that were not part of the training patterns. It took approximately 2.5 hours for training on the personal computer. A bias value of −0.3 was chosen for all neurons. Other bias values were used, including trained values. For this problem, the value of −0.3 seemed to yield the most accurate network. The results of the test patterns are shown in Table 3.

TABLE 1 TRAINING DATA FOR SIMPLE BEAM PROBLEM

	Input (Moment)										*Output*
Location	*0*	*1/9*	*2/9*	*3/9*	*4/9*	*5/9*	*6/9*	*7/9*	*8/9*	*1*	*(Location)*
Pattern 1	0	0.40	0.35	0.30	0.25	0.20	0.15	0.10	0.05	0	0.2
Pattern 2	0	0.30	0.60	0.90	0.75	0.60	0.45	0.30	0.15	0	0.4
Pattern 3	0	0.20	0.40	0.60	0.80	1.00	0.75	0.50	0.25	0	0.6
Pattern 4	0	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.35	0	0.8

TABLE 3 CONCRETE BEAM DESIGN RESULTS[a]

	Input					Desired Output	Actual Output	Percent Error
	$M_u/10^4$ ft-kips	$f_y/10^2$ ksi	$f'_c/10$ ksi	p	b/d/10	$d/10^2$ "	$d/10^2$ "	%
Pattern 1	0.0063	0.4	0.300	0.019	0.0750	0.120	0.119	0.8
Pattern 2	0.0112	0.4	0.300	0.015	0.0690	0.160	0.160	0.0
Pattern 3	0.0183	0.4	0.300	0.012	0.0700	0.200	0.209	4.5
Pattern 4	0.0929	0.4	0.375	0.013	0.0560	0.360	0.360	0.0
Pattern 5	0.0736	0.4	0.375	0.023	0.0570	0.280	0.278	0.7
Pattern 6	0.0185	0.4	0.375	0.027	0.1000	0.140	0.130	7.1
Pattern 7	0.0719	0.6	0.300	0.006	0.0875	0.320	0.317	0.9
Pattern 8	0.0397	0.6	0.300	0.011	0.0670	0.240	0.244	1.7
Pattern 9	0.0572	0.6	0.300	0.013	0.0294	0.340	0.365	6.8
Pattern 10	0.0282	0.6	0.375	0.007	0.1200	0.200	0.192	4.0
Pattern 11	0.0288	0.6	0.375	0.010	0.2570	0.140	0.143	2.1
Pattern 12	0.0476	0.6	0.375	0.014	0.0820	0.220	0.225	2.3
Pattern 13	0.0689	0.6	0.400	0.005	0.0500	0.400	0.394	1.5
Pattern 14	0.0180	0.6	0.400	0.009	0.0455	0.220	0.214	2.7
Pattern 15	0.1016	0.6	0.400	0.016	0.0500	0.320	0.329	2.8
Pattern 16	0.0138	0.6	0.500	0.008	0.1200	0.150	0.157	4.7
Pattern 17	0.0160	0.6	0.500	0.012	0.0560	0.180	0.180	0.0
Pattern 18	0.1753	0.6	0.500	0.017	0.0470	0.380	0.378	0.5
Pattern 19	0.0371	0.6	0.600	0.008	0.0400	0.300	0.301	0.3
Pattern 20	0.0340	0.6	0.600	0.014	0.2140	0.140	0.141	0.7
Pattern 21	0.0640	0.6	0.600	0.020	0.0580	0.240	0.235	2.0
The untrained testing patterns follow:								
Pattern 22	0.0172	0.4	0.300	0.017	0.0670	0.180	0.171	5.0
Pattern 23	0.0115	0.4	0.375	0.020	0.2200	0.100	0.142	42.0
Pattern 24	0.0571	0.6	0.300	0.010	0.0530	0.300	0.324	8.0
Pattern 25	0.2373	0.6	0.375	0.015	0.0440	0.450	0.483	7.3
Pattern 26	0.0323	0.6	0.400	0.014	0.1730	0.150	0.151	0.8
Pattern 27	0.0906	0.6	0.500	0.019	0.0375	0.320	0.309	3.4
Pattern 28	0.0083	0.6	0.600	0.011	0.1800	0.100	0.123	23.0
Pattern 29	0.0145	0.6	0.400	0.011	0.0560	0.180	0.173	3.9
Pattern 30	0.1138	0.6	0.500	0.009	0.0240	0.500	0.464	7.2
Pattern 31	0.0904	0.6	0.600	0.018	0.0570	0.280	0.277	1.1

[a] 1 ft = 30.48 cm; 1 kip = 4.45 KN; 1" = 2.54 cm; 1 ksi = 6.89 MPa.

SIMPLY SUPPORTED PLATE PROBLEM

The final problem involved predicting the location and magnitude of maximum moment in a simply supported rectangular plate subjected to a concentrated load. The input included four quantities: the dimension of the plate in both directions, and the *x-y* coordinates of the point of application of a unit concentrated loading. Output consisted of six quantities: the maximum moment and *x-y* location of the maximum moment for bending in both directions. The net consisted of four input neurons, two hidden layers with six neurons each, and an output layer with six neurons. Training patterns were obtained from a standard finite-element analysis. The bias value found to yield the most accurate network was a constant value of -0.2 for all neurons. Each training pattern required approximately 2 minutes of central processing unit (CPU) time on a VAX 8550 minicomputer to determine using a standard finite-element analysis program. It is noted that several other classic methods could have been used for determining the training patterns; however, the finite-element method was chosen for convenience.

Table 4 shows the training patterns used and the predicted output once the network was completely trained. Training took approximately 32 hours of CPU time on the personal computer. Table 5 lists the results of the patterns that were not trained with the network. Several conclusions can be drawn from this example. First, a considerable amount of CPU time was required for training, indicating more study should be made regarding the training method. Second, as with the examples above, the network predicted output with an accuracy that is acceptable in most design scenarios. Third, it should be noted that once the net was trained, the time required to output results for a given set of input was nearly instantaneous on a personal computer. This is a dramatic improvement in computational efficiency over the VAX finite-element solution. Similar improvements, although possibly not as dramatic, would exist when compared with classic plate solutions. This indicates that a network of this type may have considerable potential for solving time-consuming problems.

The plate network was trained for aspect ratios typical of two-way behavior, that is between 0.5 and 2.0. As can be seen in Tables 4 and 5, the actual output matched

TABLE 4 SIMPLY SUPPORTED PLATE TRAINING DATA RESULTS

Plate		Load		X-Bending						Y-Bending					
				Desired			Actual			Desired			Actual		
X	Y	X	Y	M	X	Y	M	X	Y	M	X	Y	M	X	Y
1.00	1.00	0.50	0.50	0.32	0.50	0.50	0.32	0.49	0.50	0.32	0.50	0.50	0.33	0.49	0.50
1.00	1.00	0.60	0.15	0.24	0.57	0.17	0.24	0.58	0.17	0.26	0.57	0.17	0.27	0.58	0.17
1.00	1.00	0.80	0.25	0.27	0.77	0.28	0.25	0.77	0.28	0.26	0.77	0.28	0.26	0.77	0.28
1.00	1.00	0.20	0.75	0.27	0.22	0.73	0.27	0.22	0.72	0.26	0.22	0.73	0.25	0.22	0.73
1.00	1.00	0.20	0.20	0.25	0.22	0.22	0.25	0.22	0.22	0.25	0.22	0.22	0.27	0.22	0.22
0.90	1.00	0.45	0.50	0.33	0.45	0.50	0.32	0.45	0.51	0.32	0.45	0.50	0.32	0.45	0.51
0.90	1.00	0.14	0.60	0.26	0.16	0.57	0.25	0.16	0.57	0.23	0.16	0.57	0.23	0.16	0.57
0.90	1.00	0.45	0.75	0.29	0.45	0.73	0.31	0.45	0.73	0.30	0.45	0.73	0.31	0.45	0.73
0.90	1.00	0.68	0.20	0.26	0.65	0.22	0.26	0.66	0.23	0.26	0.65	0.22	0.27	0.66	0.23
0.90	1.00	0.72	0.90	0.20	0.70	0.88	0.20	0.70	0.87	0.21	0.70	0.88	0.21	0.70	0.87
1.00	0.80	0.50	0.40	0.31	0.50	0.40	0.31	0.51	0.40	0.33	0.50	0.40	0.34	0.51	0.39
1.00	0.80	0.60	0.12	0.22	0.57	0.14	0.22	0.57	0.14	0.26	0.57	0.14	0.26	0.57	0.14
1.00	0.80	0.90	0.64	0.21	0.88	0.62	0.21	0.88	0.62	0.20	0.88	0.62	0.22	0.88	0.62
1.00	0.80	0.20	0.60	0.26	0.22	0.58	0.26	0.23	0.58	0.27	0.22	0.58	0.26	0.23	0.58
1.00	0.80	0.75	0.40	0.30	0.73	0.40	0.29	0.73	0.40	0.30	0.73	0.40	0.30	0.73	0.40
0.70	1.00	0.14	0.20	0.26	0.11	0.22	0.24	0.12	0.22	0.23	0.11	0.22	0.23	0.12	0.23
0.70	1.00	0.10	0.60	0.25	0.12	0.57	0.25	0.12	0.57	0.21	0.12	0.57	0.22	0.12	0.58
0.70	1.00	0.17	0.80	0.27	0.19	0.77	0.28	0.20	0.77	0.25	0.19	0.77	0.24	0.20	0.77
0.70	1.00	0.52	0.20	0.27	0.51	0.22	0.26	0.51	0.22	0.25	0.51	0.22	0.25	0.51	0.22
0.70	1.00	0.35	0.75	0.30	0.35	0.73	0.30	0.35	0.72	0.29	0.35	0.73	0.28	0.35	0.73
1.00	0.60	0.20	0.12	0.22	0.22	0.14	0.22	0.23	0.14	0.25	0.22	0.14	0.25	0.23	0.14
0.60	1.00	0.09	0.60	0.24	0.10	0.57	0.25	0.10	0.59	0.20	0.10	0.60	0.21	0.10	0.60
1.00	0.60	0.80	0.15	0.24	0.77	0.17	0.25	0.78	0.17	0.27	0.77	0.17	0.26	0.78	0.17
0.60	1.00	0.45	0.20	0.27	0.44	0.22	0.27	0.43	0.22	0.24	0.44	0.22	0.25	0.43	0.23
0.60	1.00	0.30	0.50	0.32	0.30	0.50	0.31	0.30	0.50	0.28	0.30	0.50	0.29	0.30	0.50
1.00	0.50	0.50	0.25	0.26	0.50	0.25	0.26	0.50	0.26	0.32	0.50	0.25	0.30	0.50	0.25
0.50	1.00	0.13	0.80	0.26	0.14	0.77	0.27	0.14	0.77	0.23	0.14	0.77	0.21	0.14	0.77
1.00	0.50	0.75	0.25	0.26	0.73	0.25	0.28	0.72	0.25	0.30	0.73	0.25	0.29	0.72	0.25
0.50	1.00	0.40	0.90	0.23	0.22	0.36	0.24	0.23	0.36	0.26	0.22	0.36	0.25	0.23	0.36
1.00	0.50	0.60	0.08	0.18	0.60	0.09	0.19	0.60	0.08	0.23	0.60	0.09	0.23	0.60	0.08

the desired output reasonably well. The plate network was subsequently trained with more patterns of larger aspect ratios, indicative of one-way plate behavior. When test patterns indicative of one-way plate behavior were entered, the error between desired and predicted output increased. It appears that the single network has a difficulty representing both two-way and one-way behavior. This seems similar to a human attempting to be an expert for several fields of study, the result being someone who is less knowledgeable in all areas.

TABLE 5 SIMPLY SUPPORTED PLATE TEST PATTERN RESULTS

Plate		Load		X-Bending						Y-Bending					
				Desired			Actual			Desired			Actual		
X	Y	X	Y	M	X	Y	M	X	Y	M	X	Y	M	X	Y
1.00	1.00	0.75	0.50	0.30	0.73	0.50	0.24	0.71	0.60	0.29	0.73	0.50	0.25	0.71	0.60
1.00	1.00	0.90	0.80	0.21	0.88	0.77	0.15	0.89	0.80	0.19	0.88	0.77	0.15	0.89	0.80
0.90	1.00	0.18	0.20	0.25	0.20	0.22	0.25	0.19	0.23	0.25	0.20	0.22	0.26	0.19	0.24
0.90	1.00	0.22	0.80	0.26	0.25	0.73	0.28	0.26	0.80	0.26	0.25	0.73	0.26	0.26	0.81
1.00	0.80	0.60	0.12	0.22	0.57	0.14	0.22	0.57	0.14	0.26	0.57	0.14	0.26	0.57	0.14
1.00	0.80	0.80	0.20	0.26	0.77	0.22	0.26	0.78	0.21	0.27	0.77	0.22	0.27	0.78	0.21
0.70	1.00	0.35	0.50	0.33	0.35	0.50	0.32	0.36	0.50	0.30	0.35	0.50	0.31	0.36	0.51
0.70	1.00	0.56	0.90	0.21	0.54	0.88	0.22	0.48	0.81	0.21	0.54	0.88	0.25	0.48	0.81
0.60	1.00	0.30	0.75	0.30	0.30	0.73	0.29	0.27	0.68	0.28	0.30	0.73	0.26	0.27	0.68
1.00	0.60	0.90	0.48	0.20	0.88	0.47	0.27	0.80	0.36	0.21	0.88	0.47	0.28	0.80	0.36
0.50	1.00	0.10	0.20	0.25	0.11	0.22	0.25	0.07	0.20	0.21	0.11	0.22	0.23	0.07	0.20
1.00	0.50	0.20	0.38	0.23	0.22	0.36	0.20	0.23	0.38	0.26	0.22	0.36	0.23	0.23	0.38

A logical conclusion here is that for large aspect ratios, a different network, such as one trained for one-way behavior only, should be utilized. An expert system could be capable of determining which network is appropriate and then pass control to that net for the actual value determination. A complex intelligent program would, therefore, consist of a combination of expert system technology and neural network technology.

CONCLUSION/FUTURE STUDY

The purpose of this article was to summarize the characteristics and use of neural networks and their application to structural engineering problems. One basic method is outlined, detailed expressions for back-propagation learning in multilayer networks are presented, and the results of the three tests are given. The problems chosen do not have anything in common and represent a broad range of structural engineering problems. These tests demonstrate that neural networks may be successfully applied to solve many civil engineering problems. These networks are capable of simulating learning of the type of knowledge used by structural engineers. While it appears that neural networks may be built to solve almost any problem in which sufficient training data exist, their use should be limited to problems that are presently difficult or time-consuming to solve. For example, many finite-element solutions fall into the time-consuming category. Several problems in which algorithmic solutions are difficult to develop are summarized in the following paragraph.

The application of neural networks in civil engineering needs to be studied further. There are several areas where additional study is needed. Some of these areas include:

1. Combining software such as NNICE with traditional rule-based expert systems to give an expert system the power to treat new problems including automatic learning.
2. Studying the use of neural networks in solving civil engineering optimization problems.
3. Using programs like NNICE with pattern recognition capabilities to help identify code and design inadequacies. Past acceptable designs would be used for training purposes.
4. Continued study involving training methods to reduce required training time and improve developed system accuracy.

NNICE was developed using 1977 Standard FORTRAN. The program contains a "user friendly," window-based system for input. A screen example is shown in Figure 5. Source code along with IBM personal computer–compatible executable code is available by sending a postage-paid addressed diskette mailer and diskette to the primary author.

REFERENCES

1. Adeli, H. and Yeh, C., Perceptron Learning in Engineering Design. *Microcomputers in Civil Engineering* 4(4):247–256, 1989).
2. Caudill, M., Neural networks primer. *AI Expert* Dec ., Feb., Apr., Jun., & Aug. issues (1987–1988).
3. Firebaugh, M.W., *Artificial Intelligence—A Knowledge Based Approach*. Boyd and Fraser, Boston, MA 1988.
4. Hecht-Nielsen, R., Neurocomputing: picking the human brain. *IEEE Spectrum* 25(3):13–18 (1988).
5. Hopfield, J.J., Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences USA* 81:6871–6874 (1982).
6. Jones, W.P. and Hoskins, J., Back-Propagation—a generalized delta learning rule. *BYTE* 12(11):155–162 (Oct 1987).
7. Lippmann, R.P., An introduction to computing with neural nets. *IEEE Acoustics Speech and Signal Processing* 4(2):4–22 (1987).
8. MacGregor, J.G., *Reinforced Concrete Mechanics and Design*. Prentice-Hall, Englewood Cliffs, NJ, 1988.

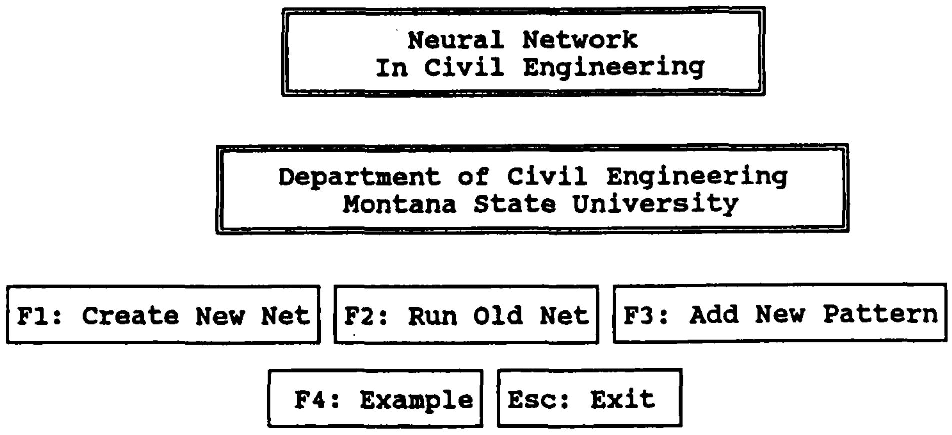


FIGURE 5. Sample NNICE Screen.

9. McClelland, J.L. and Rumelhart, D.E., *Explorations in Parallel Distributed Processing—A Handbook of Models, Programs, Exercises*. MIT Press, Cambridge, MA, 1988.
10. Minsky, M.L. and Papert, S.A. *Perceptrons* (expanded ed.). MIT Press, Cambridge, MA, 1988.
11. Rumelhart, D.E. and McClelland, J.L., *Parallel Distributed Processing—Explorations in the Microstructure of Cognition*, Vols. 1 and 2. MIT Press, Cambridge, MA, 1986.
12. Stanley, J. and Bak, E., *Introduction to Neural Networks*. California Scientific Software, Sierra Madre, CA, 1988.