

Character Recognition and Learning with Neural Networks*

Ingrid Russell[†], Zdravko Markov, Todd Neller

June 3, 2005

1. Introduction

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.

Neural networks have been applied to a wide variety of areas including speech synthesis, character recognition, diagnostic problems, medicine, business and finance, robotic control, signal processing, computer vision and many other problems that fall under the category of pattern recognition. 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, character recognition, speech recognition, and many other problems that fall under the category of pattern recognition.

For some application areas including patter recognition, neural models show promise in achieving human-like performance over more traditional artificial intelligence techniques. This project will introduce students to the basic neural networks concepts and to neural models and learning. Students develop a basic character recognition system based on a neural network model and will implement and experiment with various neural learning algorithms.

* Introductory material on neural networks included in this project are adapted from Russell [1993]

[†] Corresponding author: irussell@hartford.edu, University of Hartford, Department of Computer Science, West Hartford CT 06117

2. Project Overview

This project introduces students to the basic neural network concepts and to neural models and learning. Students will implement basic types of neural networks as well as some important approaches to learning in order to solve a number of typical pattern recognition problems. The goal of the project is to provide students with knowledge and skills to understand implement and use basic neural network models. This will allow students to understand the role that neural networks play in the more general context of AI techniques and tools. In particular, the learning objectives of the project are the following:

- Learning the basics of single-layer neural networks and perceptron type models and their use as pattern associators.
- Understanding the limitations of single-layer networks and introducing the basic types of multilayer networks as well as the backpropagation training algorithm.
- Learning the mathematical foundations of the neural network computation.
- Better understanding of the nature of problems that neural networks can solve.
- Gaining experience in implementing neural network algorithms for solving basic pattern recognition problems.
- Better understanding of the differences between the neural network approach to solving AI problems and those based on classical symbolic knowledge representation, Search and Learning.

3. Definition

Artificial neural networks are computer systems that attempt to simulate the structure of the brain in order to achieve parallelism. 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.

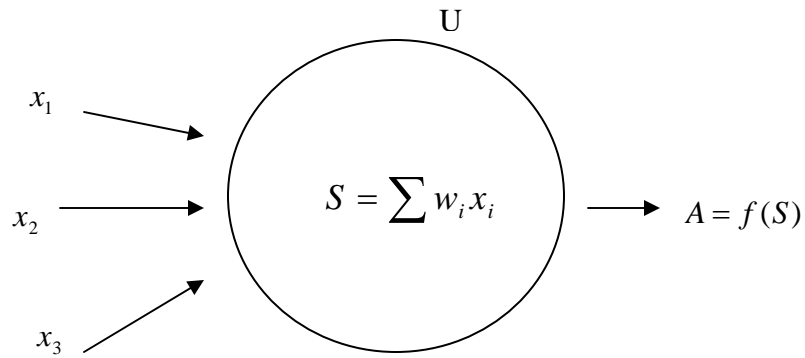


Figure 1. A single unit in a neural network.

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.

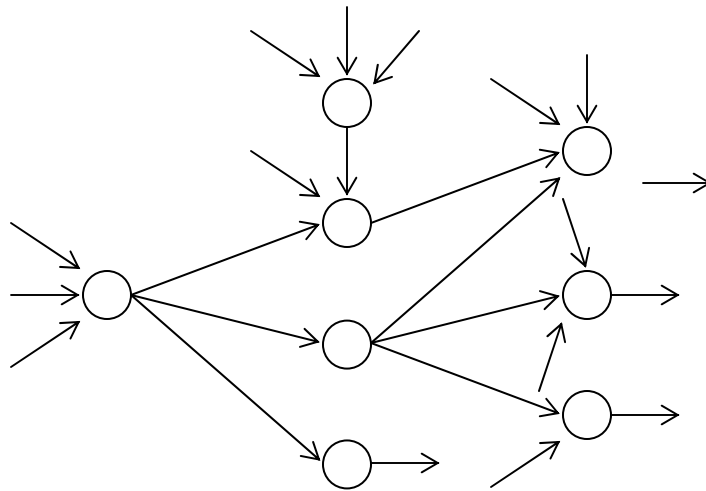


Figure 2. An example of a neural network structure.

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.

4. 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.

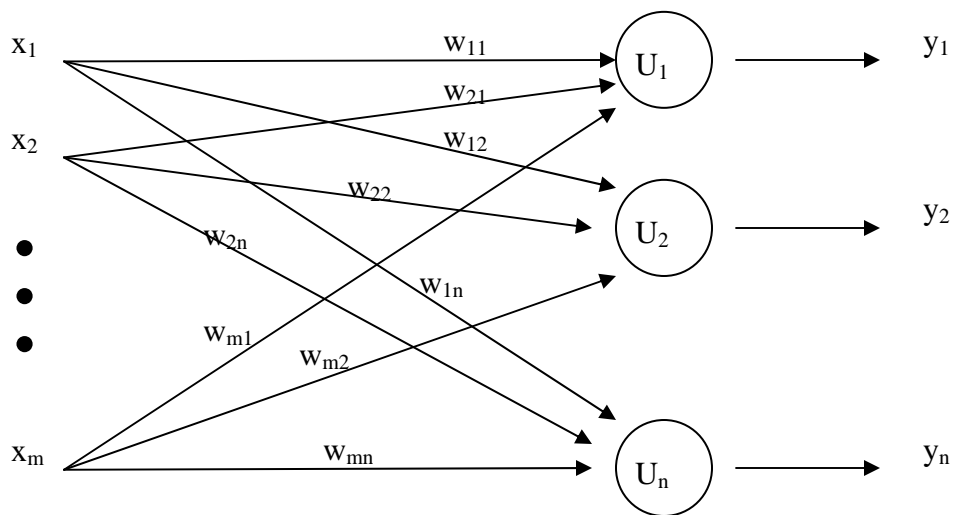


Figure 3. A single-layer linear model.

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

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

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

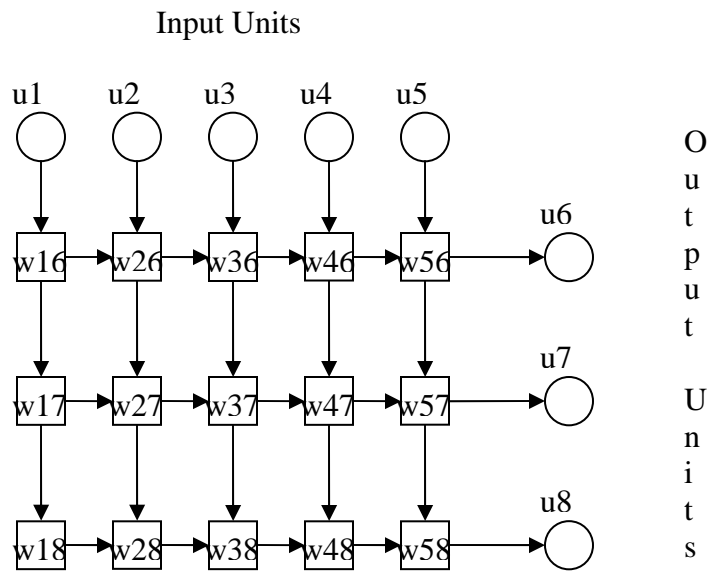
$$\vec{Y}_{n \times 1} = W_{m \times n}^T \vec{X}_{m \times 1}$$

4.1 Deliverable 1: A Single Layer Neural Network Model

A pattern associator is a single-layer network capable of learning associations between input patterns and output patterns. Each unit in the input layer is connected via weighted connections to each unit in the output layer. The activation function of an output unit u_j is given by:

$$a_j = \sum_i w_{ij} * a_i$$

where a_i is the activation of u_i and w_{ij} is the weight of the connection from u_i to u_j . Often times a threshold function is used to transform a real output into the desired binary output. A pattern associator with five input units and three output units is presented below.



1. Construct by hand a pattern associator using a threshold function to compute the OR problem as represented in the table below:

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

What weight vector did you use? What threshold function did you use?

2. Is it possible to construct a pattern associator that computes the XOR problem as represented in the table below? Explain your results in detail.

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

In the above examples you saw how a network can be designed once the weights of the connections are known. One of the more important features of neural networks is their

ability to self-modify and learn. Learning means the ability of the network to modify its weights in order to learn input/output associations.

5. Learning in a Neural Network

Learning is essential to most of these neural network architectures and hence the choice of a learning algorithm is a central issue in network development. What is really meant by saying that a processing element learns? Learning implies that a processing unit is capable of changing its input/output behavior as a result of changes in the environment. Since the activation rule is usually fixed when the network is constructed and since the input/output vector cannot be changed, to change the input/output behavior the weights corresponding to that input vector need to be adjusted. A method is thus needed by which, at least during a training stage, weights can be modified in response to the input/output process. A number of such learning rules are available for neural network models. In a neural network, learning can be supervised, in which the network is provided with the correct answer for the output during training, or unsupervised, in which no external teacher is present.

6. 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 exclusive (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].

6.1 Deliverable 2: Modeling a Single-Layer Network Using Hebbian Learning

One of the most appealing characteristics of a pattern associator is the fact that it can generate what it learns about one pattern to other similar input patterns. Pattern associators have been widely used in distributed memory modeling.

The pattern associator is one of the more basic single-layer networks. Its architecture consists of two sets of units, the input units and the output units. Each input unit connects to each output unit via weighted connections. Connections are only allowed from input units to output units. The effect of a unit u_i in the input layer on a unit u_j in the output layer is determined by the product of the activation a_i of u_i and the weight of the connection from u_i to u_j . The activation a_j of a unit u_j in the output layer is given by:

$$a_j = \sum_i w_{ij} * a_i ,$$

A pattern associator can be trained to respond with a certain output pattern when presented with an input pattern. The connection weights can be adjusted in order to change the input/output behavior. However, one of the most interesting properties of these models is their ability to self-modify and learn. The learning rule is what specifies how a network changes its weights for a given input/output association. The most commonly used learning rules with pattern associators are the Hebb rule and the Delta rule.

As mentioned above the Hebb learning rule states that the change in weight w_{ij} from u_i to u_j is given by: $\Delta w_{ij} = r * a_i * a_j$,

Where r is the learning rate, and a_i, a_j are the activation of the units u_i and u_j respectively. In the exercise below, you will illustrate how the Hebb learning rule can be used to adjust the weights so that a pattern associator can learn input/output associations.

1. Train a single-layer neural network with the AND problem as represented in the table below:

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

using a threshold value of 0.8 and a learning rate of 0.5, how many iterations do you need for the network to be trained?

2. Consider the following single layer input-output pattern associator with three input units u_1, u_2, u_3 and one output unit u_4 . Use the Hebb learning rule to design a network to model the pattern input/output pattern below.

		Input Units			Output
		u1	u2	u3	u4
V1	Pattern #1	+1	-1	-1	-1
V2	Pattern #2	-1	+1	-1	-1
V3	Pattern #3	-1	-1	+1	+1
V4	Pattern #4	+1	+1	+1	+1

(a) Using $W = (0 \ 0 \ 0)$ as the initial weight vector and 0.25 as learning rate and after one iteration (i.e., after all four input/output pattern pair is presented once) present a table whose rows represent the four patterns and whose columns represent: u1 input, u2 input, u3 input, expected output, actual output, and the three corresponding weight values. What is the weight vector after the presentation of each input vector? What is the value

of the weight vector after one iteration? What is the value of $W.V_i$ for each i ? Can this pattern associator learn this input/output pattern? Explain.

(b) Write a program that simulates and tests a pattern associator that uses the Hebb learning rule to learn input/output associations. The program should allow the user to train the network to associate a given input/output pattern and also to test the performance of the network. The input to your program should include the number of input units, number of output units, number of patterns, learning rate, number of iterations desired, input/output associations.

(c) Use your program to train the input/output associations given above.

(i) Study the effects of the following factors: learning rate, straight versus permuted training (randomly selecting the order), and the number of training iterations.

(ii) What combination of the above factors provides reasonable performance? Use this combination for training the network.

(iii) Use the best combination you arrived at in (b) to test the network's performance using the training patterns.

(iv) Test the network's ability to generalize what it learned about the training patterns to other similar patterns. One way to do that is to test the performance of the network when a linear combination of the training patterns is presented. Explain your conclusions.

Below are some such test data:

(a)	0	0	-2
(b)	-2	0	0
(c)	0	0	2
(d)	3	-1	-3

(d). Train your network as you did above but this time with the following input/output associations:

		Input Units			Output
		u1	u2	u3	u4
V1	Pattern #1	+1	+1	0	2
V2	Pattern #2	+1	-1	+1	1
V3	Pattern #3	0	+1	+2	3

(i) Test the performance of the network with the training patterns. Explain the results you come up with.

(ii) Compare the results of (d) with the results in (c). How do you explain the difference in the results?

7. 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)

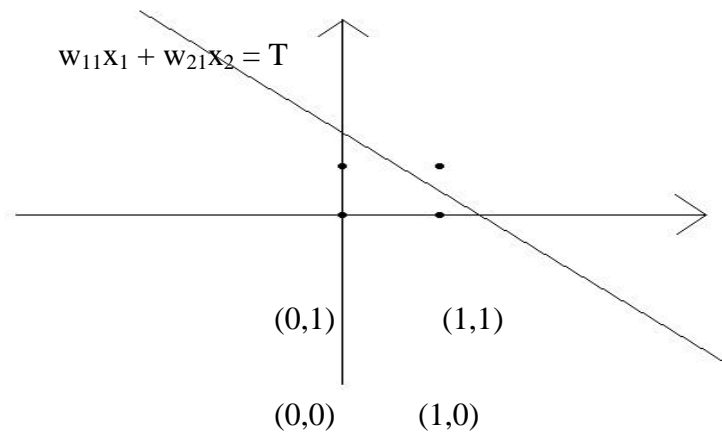


Figure 4. The graph of $w_{11}x_1 + w_{21}x_2 = T$.

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.

7.1 Deliverable 3: Modeling a Single-Layer Network using the Delta Rule

1. Revise the program in deliverable II so there is an option to train using the delta learning algorithm described above. Test the performance of the network using the training pattern in exercise #4 of deliverable I. Compare the results here with 4(a) of deliverable I. How do you explain the difference in the results?

7.2 Deliverable 4: Implementing a Basic Character Recognition System

1. Using the delta learning rule, a zero initial weight vector, and a threshold value of 0.5, implement a single layer network that can be trained to recognize the letters 'A' and 'B'. Letters should be represented by a 5x5 binary matrix and hence the input layer should consist of 25 input units. How many output units will you need? How many iterations are needed for the network to recognize the two letters. Show the value of the weight matrix as well as the output values after each iteration.

2. Write and test a program that implements the character recognition system in exercise #1. The program should read in the threshold value and the number of iterations to be used. What test data did you use? Report on the results of testing and classification of the input patterns used as well as the classification results.

8. 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 $\vec{Z} = W_2 \vec{Y}$.

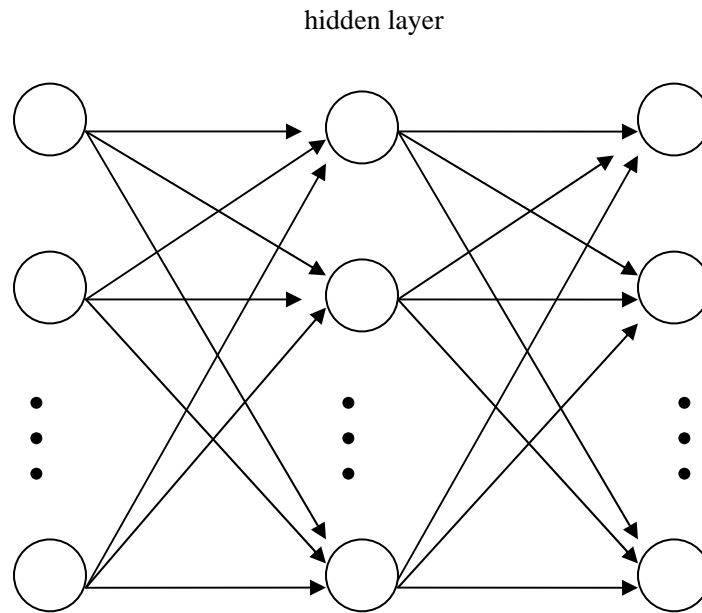


Figure 5. A multilayer network.

Hence

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

Consequently, the system is equivalent to a single-layer network with weight matrix $W = W_2 W_1$. 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

$$W_1^T \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad W_2^T \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 1,$$

$$W_1^T \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad W_2^T \begin{pmatrix} 0 \\ 1 \end{pmatrix} = 1,$$

$$W_1^T \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad W_2^T \begin{pmatrix} 0 \\ 0 \end{pmatrix} = 0,$$

$$W_1^T \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad W_2^T \begin{pmatrix} 0 \\ 0 \end{pmatrix} = 0.$$

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].

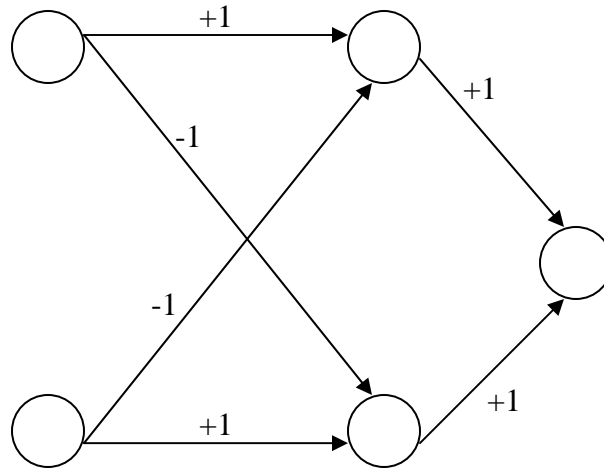


Figure 6. A multilayer system representation of the XOR function.

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

8.1 Deliverable 5: Computing the XOR Problem using a Multilayer Network

1. Design a multilayer network with one hidden layer that can compute the XOR problem. What is the corresponding weight vector and what threshold value did you use?

9. 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. Artificial neural networks are computer systems that

attempt to simulate the structure of the brain in order to achieve parallelism. 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].

9.1 Deliverable 6: A Final Report

Write a report commenting on the relation of the approaches used in this project to the areas of search and knowledge representation and reasoning (KR&R). In particular, answer the following questions:

- Which search techniques have been used in the project?
- If no search has been used explicitly, what is the relation of the approaches used in the project to the area of search? (any search components used or project approaches applicable in the area of search).
- Which KR&R techniques have been used in the project?
- If no KR&R has been used explicitly, what is the relation of the approaches used in the project to the area of KR&R? (any KR&R components used or project approaches applicable in the area of KR&R).

10. Extra Credit – A Character Recognition System

Revise the program in deliverable #4 so it implements a character recognition system for all 26 letters in the alphabet. Your system should implement a multilayer

network with one hidden layer and you should use the backpropagation learning algorithm.

References and Readings

Freeman, J., and D. Skapura. 1991. *Neural Networks*. Reading MA: Addison-Wesley.

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.

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

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. 1992. A Neural Network Simulation Project. In *The Journal of Artificial Intelligence in Education. Vol 3 No 1*.

Russell, I., 1993. Neural Networks. In *The UMAP Journal. Vol 14, No 1*.

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