An Investigation into
Simulation of Neural
Networks for
Pattern Recognition

Graduate Project

by

Roland Dominguez

January 29, 1989

Dr. Herbert R. Haynes
chairman

Mr. David C. Perkins
member

Dr. Roy S. Ellzey
member
Once the input and output layers of neurons were presented with data, there remained the problem of how to store or learn these patterns for future recall. What was needed was a simple, efficient function to map the input vector to the output vector and then store the information for this mapping. Since this implementation of an ANN was to run on a microcomputer, the learning algorithm chosen was of extreme importance. For this reason, a simple one-step algorithm was chosen [1]. over more thorough and complex algorithms [8, 9].

In this learning algorithm we form a mapping from the input layer of neurons to the output layer. This is also referred to as forming an "association" between two patterns [1, 2, 5, 6, 7, 8, 9]. This is first accomplished by transforming the input and output vectors into "bi-polar vectors" [1, 11]. Using this idea, the original input and output vectors are transformed into vectors where the only values are 1 and -1 instead of the original 1 and 0 that were input from the data file. One example of constructing a bi-polar vector is: given some arbitrary binary pattern in a vector such as \([0 1 1 0 1 0 0 1]\), merely replace the 0's with -1's, giving \([-1 1 1 -1 -1 -1 -1 -1]\). According to Kosko [1], "It can be shown that BAM (bi-directional associative memory) correlation encoding improves if bi-polar vectors and matrices are used instead of binary vectors and matrices" [1].
Since we are forming a mapping of one symbolic vector onto another where the input layer is fully interconnected to the second layer, we have a symbolic matrix which can be referred to as Memory where each entry $m_{ij}$ indicates the strength of the connection between neuron $i$ of one layer to neuron $j$ of the next layer [1, 2, 5, 6, 7, 8, 9] as shown in Fig. 1.

**Input layer of neurons**

\[
\begin{array}{cccccc}
  v_{11} & v_{12} & v_{13} & \ldots & v_{1n} \\
  m_{21} & m_{22} & m_{23} & \ldots & m_{2n} \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  m_{n1} & \ldots & \ldots & \ldots & m_{nn}
\end{array}
\]

**Fig. 1** This figure illustrates the interconnection between each layer of neurons. Each row of the matrix contains the strengths of the connections between each neuron of the output layer and all the input neurons to which it is connected. Likewise each column contains the strengths of the connections between each neuron of the input layer and all the output neurons to which it is connected.

Since this ANN implementation used bi-polar vectors to build memory, entries in Memory were either positive, negative, or zero. These entries in memory represented the strengths of the connections between the neurons of the input and output layers and are classified as excitatory, inhibitory, or neutral, respectively. These classes are indicative of the type
of influence that one neuron will have on another. An excitatory (positive) connection influences a neuron to fire (change its output to a 1), an inhibitory (negative) connection influences a neuron not to fire (change its output to a 0), and a neutral connection (0) does nothing to influence a neuron (remains the same) [1, 2, 8].

To form a matrix from the two bi-polar vectors we must follow the laws of linear algebra. To form a matrix of size \( N \times N \), where \( N \) is the number of elements in each layer of neurons, in each row vector, we take the transposition of the input vector of neurons times the row vector for the output layer of neurons [1, 10]. This gives us a matrix where each row is simply the output vector multiplied by each element of the transposition of the vector of the input layer of neurons. Thus each row of the matrix Memory is simply the \( n \)'th element of the transposition vector of the input layer multiplied by the row vector of the output layer and is only different from the original values in sign, since each value of the input vector is either 1 or -1, due to the use of bi-polar vectors [1, 2, 8] (fig. 2).
fig. 2 This figure illustrates how a memory matrix is constructed. Each element of the \( n \times 1 \) vector is multiplied by the \( 1 \times n \) vector to give an \( n \times n \) matrix. It is this \( n \times n \) matrix that contains the strengths of the connections between each neuron of one layer to all neurons of the corresponding layer.

This is only for one set of patterns. To learn multiple sets of patterns, we must perform this operation for each pair to be learned. Since different sets of patterns produce different matrices it would appear that only one association could be stored over one set of neurons, but this is not the case. In fact, an ANN has the ability to store multiple associations on one set of connections (Memory). To store more than one matrix of connections for a pair of patterns, we merely add the result (a matrix) of the outer product of the transposition of the input vector times the output vector of neurons to Memory. Thus for the association from pattern 1 to pattern 2, we have bi-polar vectors \( \mathbf{v}_1 \) and \( \mathbf{v}_2 \). Their outer product or "association" is given by \( \mathbf{v}_1 \mathbf{v}_2^T \times \mathbf{v}_2 = \mathbf{M} \), where \( \mathbf{v}_1 \mathbf{v}_2^T \) is the transposition of the input row vector, \( \mathbf{v}_2 \) is the output row vector and \( \mathbf{M} \) is the \( n \times n \) matrix that stores the strengths of the connections between the input and output layer for this
association. More explicitly, for a series of patterns \((p_1, p_3)\),
where each pattern is stored as a bi-polar vector, to store more
than one set of associations we have

\[(p_1^T \ast p_2) + (p_3^T \ast p_4) + (p_5^T \ast p_6) + \ldots\]

which gives

\[M_1 + M_2 + M_3 + \ldots\]

At this point we have now encoded a series of associations
into a symbolic matrix of connection strengths between two layers
of artificial neurons. Now the problem changes to one of using
this matrix Memory to recall associations, or to "remember" a
pair of patterns when presented with either the original pair or
one that closely resembles the original pair. It is obvious that
to recall some object we must first have some input with which to
work. Thus the input layer is given a pattern. Since we have
formed an association to some output layer we also need to
initialize the output layer of neurons with some pattern \([1, 8]\).

The Memory matrix stores pairs of patterns by using
distributed representation. This means that instead of using
individual storage locations for individual items, one storage
matrix (Memory) is used to place all of our information.
Inside this matrix each cell is used to store one "bit" of
information, either a 1 or -1. Since we are storing multiple
matrices in Memory, each cell of Memory can be affected by either
decreasing or increasing in value; thus the representation of the
association built between a pair of patterns is spread across an
enire matrix instead of just one variable. By spreading this representation across a large surface it is possible to store more than one association in the same storage area (Memory).

To extract the original input and output patterns from a storage space that contains information for other associations, the neurons of both the input and output layer must be initialized with some pattern. Each neuron will contain one bit of the pattern (1 or 0). Since each bit of information represents only a part of the whole pattern, this bit of information must be checked to see whether it is a valid part of memory of some pattern that has been previously learned. The way to determine whether a bit is a valid part of memory is to sum the product of each bit times the strength of the connection from its node to those to which it is connected. This final sum is used to determine whether or not the original bit is valid. Since this process is performed for each bit of information in each neuron of each layer, each bit of information represents a possible final value for a neuron. The idea is to try to use these bits of information that represent a pattern to reach a pair of patterns on both layers of nodes that closely resemble the original patterns learned. This is how an extraction of an association occurs. Much more research is currently taking place to fully understand this phenomena [8].

Using this idea, we step into a process where the ANN works toward finding some unchanging state of the input layer and the
Abstract

The problem solved by this project is that of pattern recognition. This consists of storing a set of patterns in one programmer-defined storage area and then recalling any one of the stored patterns by presenting either one of the original patterns or a pattern which contains only part of the original to an algorithm which extracts the original pattern from the storage area. This task is solved by simulating a small network of neurons, more commonly known as an artificial neural network (ANN). Using algorithms that perform linear algebra operations on one- and two-dimensional arrays, two layers of neurons are simulated and trained to learn a set of patterns and then recall those patterns.

Most ANN modeling is implemented on high-end computers such as: supercomputers, workstations, and mini-computers. This ANN model was implemented using a microcomputer for the following reasons: to show that any set of algorithms can be implemented on almost any computer, and to show that this type of research can be made available to a much larger proportion of the population (since microcomputers outnumber higher end computers).
# Table of Contents

**Abstract** .......................................................... 2

**Introduction** ...................................................... 4

**Implementation of the Program** .......................... 14

**Summary and Conclusions** ................................. 34

**Bibliography** .................................................... 43

**Appendix** .......................................................... 45
Introduction

Many different classes of ANN models have been developed. One class of ANNs performs pattern association. This ANN is a pattern associator. The goal is to construct an association between patterns input to one layer of nodes and patterns input to a second layer of nodes. The final objective is to find a group of connections between each node of one layer to all nodes of the second layer and vice-versa, so that when a particular pattern is presented to the first layer of neurons, the associated pattern will appear across the second set [8]. For this ANN, patterns must be learned in pairs so that one pattern will be associated with another pattern. To do this, we present some pattern denoted by "A" to the first layer of neurons and then present a second pattern denoted by "a" to the second layer of neurons. Using some algorithm, we form a set of connections across a storage area, which symbolizes memory. At the end of this process we have an association between a pair of patterns denoted by (A, a).

Another goal of this project was to keep the program simple enough so that the source code would be easy enough for anyone to understand and to keep the resource requirements at a minimum in order to implement this project on a microcomputer.

To solve this problem simply, a small two-layer network of artificial neurons was simulated. One layer was treated as the
input layer and the second layer was treated as the output layer. Both layers were then presented two patterns from which to build a mapping from the input to the output layer. This particular type of ANN is referred to as "bi-directional associative Memory" by Kosko [7], which is a variation on some of the neural networks implemented by Hopfield [2, 8, 10] and Hebb [9].

The patterns to be learned were letters of the alphabet constructed in a 5 X 9 array of binary digits. For example the letter "A" was given by:

```
1 2 3 4 5

1 0 0 0 0 0
2 0 0 1 0 0
3 0 1 0 1 0
4 1 0 0 0 1
5 1 1 1 1 1
6 1 0 0 0 1
7 1 0 0 0 1
8 1 0 0 0 1
9 1 0 0 0 1
```

The ones form the pattern to be learned, while the zeros form the background field upon which the character is imposed.

This type of pattern representation is used in many areas of ANN research [1, 2, 4, 5, 6, 7, 8, 9, 11]. Since the input and output vectors consist of arrays symbolizing vectors, the data was read and stored in one-dimensional arrays. Thus the original 5 X 9 input pattern was stored in a 45-element array. Row 1 of the pattern was stored in positions 1 - 5 of the array, row 2 was stored in positions 6 - 10, and so forth. The input to the output layer of neurons was treated in the same manner.
Once the input and output layers of neurons were presented with data, there remained the problem of how to store or learn these patterns for future recall. What was needed was a simple, efficient function to map the input vector to the output vector and then store the information for this mapping. Since this implementation of an ANN was to run on a microcomputer, the learning algorithm chosen was of extreme importance. For this reason, a simple one-step algorithm was chosen [6], over more thorough and complex algorithms [2, 8].

In this learning algorithm we form a mapping from the input layer of neurons to the output layer. This is also referred to as forming an "association" between two patterns [1, 4, 5, 6, 7, 8, 9]. This is first accomplished by transforming the input and output vectors into "bi-polar vectors" [4, 7]. Using this idea, the original input and output vectors are transformed into vectors where the only values are 1 and -1 instead of the original 1 and 0 that were input from the data file. One example of constructing a bi-polar vector is: given some arbitrary binary pattern in a vector such as \([0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1]\), merely replace the 0's with -1's, giving \([-1 \ 1 \ 1 \ -1 \ 1 \ -1 \ -1 \ 1]\). According to Kosko [7], "It can be shown that BAM (bi-directional associative memory) correlation encoding improves if bi-polar vectors and matrices are used instead of binary vectors and matrices" [7].
Since we are forming a mapping of one symbolic vector onto another where the input layer is fully interconnected to the second layer, we have a symbolic matrix which can be referred to as Memory where each entry \( m_{ij} \) indicates the strength of the connection between neuron \( i \) of one layer to neuron \( j \) of the next layer [1, 4, 5, 6, 7, 8, 9] as shown in fig. 1.

### Input layer of neurons

\[
\begin{array}{cccc}
vl_1 & vl_2 & vl_3 & \ldots & vl_n \\
\hline
m_{11} & m_{12} & m_{13} & \ldots & m_{1n} \\
m_{21} & & & & \\
\vdots & & & & \\
m_{n1} & \ldots & \ldots & \ldots & m_{nn}
\end{array}
\]

**fig. 1** This figure illustrates the interconnection between each layer of neurons. Each row of the matrix contains the strengths of the connections between each neuron of the output layer and all the input neurons to which it is connected. Likewise each column contains the strengths of the connections between each neuron of the input layer and all the output neurons to which it is connected.

Since this ANN implementation used bi-polar vectors to build memory, entries in Memory were either positive, negative, or zero. These entries in memory represented the strengths of the connections between the neurons of the input and output layers and are classified as excitatory, inhibitory, or neutral, respectively. These classes are indicative of the type
of influence that one neuron will have on another. An excitatory (positive) connection influences a neuron to fire (change its output to a 1), an inhibitory (negative) connection influences a neuron not to fire (change its output to a 0), and a neutral connection (0) does nothing to influence a neuron (remains the same) [7, 8, 9].

To form a matrix from the two bi-polar vectors we must follow the laws of linear algebra. To form a matrix of size \( N \times N \), where \( N \) is the number of elements in each layer of neurons in each row vector, we take the transposition of the input vector of neurons times the row vector for the output layer of neurons [3, 7]. This gives us a matrix where each row is simply the output vector multiplied by each element of the transposition of the vector of the input layer of neurons. Thus each row of the matrix **Memory** is simply the \( n \)'th element of the transposition vector of the input layer multiplied by the row vector of the output layer and is only different from the original values in sign, since each value of the input vector is either 1 or -1, due to the use of bi-polar vectors [7, 8, 9] (fig. 2).
fig. 2 This figure illustrates how a memory matrix is constructed. Each element of the $n \times 1$ vector is multiplied by the $1 \times n$ vector to give an $n \times n$ matrix. It is this $n \times n$ matrix that contains the strengths of the connections between each neuron of one layer to all neurons of the corresponding layer.

This is only for one set of patterns. To learn multiple sets of patterns, we must perform this operation for each pair to be learned. Since different sets of patterns produce different matrices it would appear that only one association could be stored over one set of neurons, but this is not the case. In fact, an ANN has the ability to store multiple associations on one set of connections (Memory). To store more than one matrix of connections for a pair of patterns, we merely add the result (a matrix) of the outer product of the transposition of the input vector times the output vector of neurons to Memory. Thus for the association from pattern 1 to pattern 2, we have bi-polar vectors $v_1$ and $v_2$. Their outer product or "association" is given by $v_1^T \times v_2 = M$, where $v_1^T$ is the transposition of the input row vector, $v_2$ is the output row vector and $M$ is the $n \times n$ matrix that stores the strengths of the connections between the input and output layer for this
association. More explicitly, for a series of patterns \((p_1, p_2)\) where each pattern is stored as a bi-polar vector, to store more than one set of associations we have

\[
(p_1^T \ast p_2) + (p_3^T \ast p_4) + (p_5^T \ast p_6) + \ldots
\]

which gives

\[
M_1 + M_2 + M_3 + \ldots
\]

At this point we have now encoded a series of associations into a symbolic matrix of connection strengths between two layers of artificial neurons. Now the problem changes to one of using this matrix Memory to recall associations, or to "remember" a pair of patterns when presented with either the original pair or one that closely resembles the original pair. It is obvious that to recall some object we must first have some input with which to work. Thus the input layer is given a pattern. Since we have formed an association to some output layer we also need to initialize the output layer of neurons with some pattern [7, 9].

The Memory matrix stores pairs of patterns by using distributed representation. This means that instead of using individual storage locations for individual items, one storage matrix (Memory) is used to place all of our information. Inside this matrix each cell is used to store one "bit" of information, either a 1 or -1. Since we are storing multiple matrices in Memory, each cell of Memory can be affected by either decreasing or increasing in value; thus the representation of the association built between a pair of patterns is spread across an
entire matrix instead of just one variable. By spreading this representation across a large surface it is possible to store more than one association in the same storage area (Memory). To extract the original input and output patterns from a storage space that contains information for other associations, the neurons of both the input and output layer must be initialized with some pattern. Each neuron will contain one bit of the pattern (1 or 0). Since each bit of information represents only a part of the whole pattern, this bit of information must be checked to see whether it is a valid part of memory of some pattern that has been previously learned. The way to determine whether a bit is a valid part of memory is to sum the product of each bit times the strength of the connection from its node to those to which it is connected. This final sum is used to determine whether or not the original bit is valid. Since this process is performed for each bit of information in each neuron of each layer, each bit of information represents a possible final value for a neuron. The idea is to try to use these bits of information that represent a pattern to reach a pair of patterns on both layers of nodes that closely resemble the original patterns learned. This is how an extraction of an association occurs. Much more research is currently taking place to fully understand this phenomena [9].

Using this idea, we step into a process where the ANN works toward finding some unchanging state of the input layer and the
output layer of neurons. This state indicates that the ANN has reached a stable state and has "remembered" (successfully or unsuccessfully) an association between its input and output layers of neurons.

To accomplish this task we force the input pattern through the connections (Memory) to the output layer and then from the output layer back through the connections to the input layer until no changes occur in either layer. Since each neuron of the output layer is connected to each neuron of the input layer and vice versa, each neuron of the output layer takes into account the entire input pattern before an update of that output neuron takes place. The update of an output neuron is taken by summing the product of each value of the input layer times the strength of the connection between that output neuron and each input, as shown in figure 3.

![Input and Memory Matrix](image)

**fig 3.** This figure shows how the sum for the update of a neuron in the output layer is computed. As shown above the sum is taken by summing the product of each input value times the corresponding row values of the memory matrix for the output neuron being updated.

Mathematically, the value of output neuron $i$ is $\sum_{j=1}^{n} M_{i,j} \text{input}_j$. 
where $m_{i,j}$ are entries in the Memory matrix that indicate the strength of the connection between each input neuron and the output neuron, and each $i$ is the value of each input neuron. The subscript $i$ indicates which output neuron is currently being updated [6, 7, 8, 9]. It should be noted that this takes place for each output neuron of the output layer of neurons. This completes the update of the layer of output neurons.

To update the input layer of neurons, we use the same series of steps that were used to update the output neurons except that the transposition of Memory is used, since each column of the matrix represents the strength of the neurons for all output neurons to each input neuron (fig. 4). Thus the formula for the input neuron update for neuron $i$ is $\sum_{j=1}^{n} m_{i,j} \cdot \text{output}_j$.

\[
\begin{bmatrix}
0_1 & 0_2 & \ldots & 0_n
\end{bmatrix}
\begin{bmatrix}
\cdots & m_{11} & \cdots \\
\cdots & m_{21} & \cdots \\
\cdots & \cdots & \cdots \\
\cdots & m_{n1} & \cdots \\
\end{bmatrix}
= 0_1m_{11} + 0_2m_{12} + \ldots + 0_nm_{1n}
\]

$1 \times n$ $n \times n$

fig 4. This figure shows how the sum for the update of a neuron in the input layer is computed. As shown above the sum is taken by summing the product of each output value times the corresponding column values of the memory matrix for the input neuron being updated.

These two update procedures occur over and over until either no changes occur in any of the neurons of both layers or the update procedure takes place more than some predetermined number of times.