NEURAL NETWORKS IN APL

By Manuel Alfonseca
IBM Madrid Scientific Center
Paseo de la Castellana, 4
28046 Madrid (SPAIN)

ABSTRACT

Neural networks are fairly straightforward to program in a matrix oriented language such as APL. The only general improvement that would benefit them would be the implementation of sparse matrices. Small networks can be trained quite easily using the standard procedures (back propagation, etc).

INTRODUCTION

A neural network (also called a "connectionist system") is a set of elementary units, called neurons, mutually related by means of connections.

Each neuron has a certain number of inputs and a single output. The output, however, can divide itself to provide connections (inputs) to many other neurons. In addition, each neuron has an associated number (its threshold). Each connection also has an associated number (its weight).

The response of a neuron is a procedure that computes the output of the neuron as a function of its inputs, the weights of the input connections and the threshold of the neuron. Usually, the response of a neuron can be expressed in the following way:

$$f((\Sigma\ w_i\ x_i\ ) - \Theta) \qquad \text{(Equation 1)}$$

where x_i is the set of inputs to the neuron, w_i are the respective weights of the input connections, Θ is the neuron threshold and f is the response function.

In typical neural networks, connections are such that the neurons can be divided in a certain number of layers. Neurons in the first layer (the input layer) have inputs that do not come from other neurons, but from outside the neural network (from the environment). Neurons in the last layer (the output layer) have outputs that do not go to other neurons, but to the environment. There may be zero to any number of intermediate layers (also called "hidden layers").

A neural network where at least a neuron sends a connection to another neuron in a preceding layer is a neural network with feedback. An interesting family of neuron networks with feedback is called "Hopfield neural networks" (see reference 1).

A neural network with just two layers (one input layer and one output layer) and no feedback between them is called a "perceptron". In an important paper (reference 2) Minsky and Papert proved that it is impossible to generate the "exclusive OR" operation with a perceptron. This paper effectively put an end to all research in neural networks for several years. Current research usually uses neural networks with at least one intermediate layer.

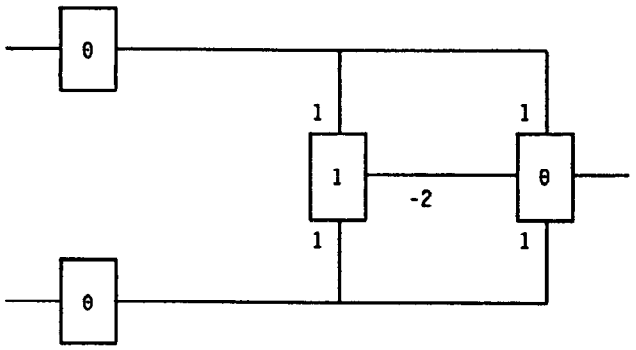
MATRIX-VECTOR REPRESENTATION OF A NEURAL NETWORK

In general, any neuron in a neural network can provide an input (a connection) to any other neuron. Therefore, the network structure can be represented by a square n by n matrix, where n is the number of neurons in the network and the element i,j in the matrix is the weight of the connection between neuron i and neuron j. Nonexistent connections can be represented as connections of zero weight (since equation 1 is not affected by those null connections).

The connection matrix represents the structure of the networks. To include all the available information we need an additional vector with the thresholds of all the neurons in the network, given, of course, in the same order as in the matrix rows and columns.

Example

The following is an example of a neural network that computes the "exclusive OR" operation:



All the neurons in the network have the following response function:

$$(\Sigma\, w_i\, x_j) > \Theta$$

The preceding neural network has three layers: two input neurons, one hidden neuron and one output neuron.

The structure of this neural network can be represented by the following matrix:

```
0 0  1  1
0 0  1  1
0 0  0 -2
0 0  0  0
```

while the threshold vector is equal to:

```
0 0  1  0
```

COMPUTATION OF THE NETWORK OUTPUT

Let us assume that we have a neural network represented by a matrix and a threshold vector. How do we compute the output of the network as a result of receiving certain inputs?

We will represent the inputs as a vector of values which we will extend to the same length as the number of neurons in the network. This extension is easy. It is enough to assume that all the neurons have exactly one input, and assigning zero as the input value of those neurons that in actual fact did not have any input.

In the case of the previous example, there are two input neurons (corresponding to rows 1 and 2 of our matrix). Rows 3 and 4, which are not input neurons, should always have an input value of zero. Therefore, the input vector for the example will be of the form:

```
x x 0 0
```

where x can be replaced by either zero or one. There are, therefore, four possible input vectors in the example:

```
0 0 0 0
0 1 0 0
1 0 0 0
1 1 0 0
```

The output of the network can be computed by means of the following APL2 function:

```
[0]  Z←NETWORK COMPUTE INPUT;A;CONEC;
       THRESHOLD
[1]   (CONEC THRESHOLD)←NETWORK
[2]   Z←INPUT
[3]  L:A←Z
[4]   Z←(A+.×CONEC)>THRESHOLD
[5]   Z←INPUT+Z
[6]   →(~A≡Z)/L
```

The left argument is the definition of the network, and is a a general vector of two elements. The first is the connectivity matrix, the second is the threshold vector.

It will be noticed that the response function, applied to the whole neural network, reduces in this case to an inner product and a comparison.

The preceding function has a loop. This is due to the fact that each inner product propagates the effect of the input to the next accessible layer. The loop, which proceeds until the network stabilizes, will eliminate the transient stages and provide us with the steady-state result. In a neural network without feedback, the loop will be executed at most n times, where n is the number of layers in the network, usually equal to 3.

Let us assume that the matrix of the example network is contained in variable MATRIX, while the threshold vector is in variable TH. The following APL2 instructions compute the results of the network corresponding to all the possible inputs:

```
      IN←(0 0 0 0) (0 1 0 0) (1 0 0 0)
         (1 1 0 0)
      (⊂MATRIX TH) COMPUTE"IN
0 0 0 0  0 1 0 1  1 0 0 1  1 1 1 0
```

It will be noticed that the fourth element of the result is the exclusive OR of the first two elements, which are the inputs. The COMPUTE function also provides us with additional information, since it includes the final state of all the neurons in the network.

MATRIX REPRESENTATION OF A NEURAL NETWORK

If the response of all the neurons in a network is of the form indicated by equation 1, the network will be equivalent to another network, where all the neurons in the original network are present, with the same connections and weights, but with zero threshold, and where an additional input neuron has been added, whose output is always 1, and which is connected to every neuron in the network by means of a connection whose weight is equal to minus the threshold of the target network in the original network. The proof of this assertion is obvious, just by looking at equation 1.

Therefore, if the indicated hypothesis holds (which is usually the case) a neural network with n neurons and arbitrary thresholds can be considered equivalent to another neural network with n+1 neurons, all of them with zero threshold. Therefore, there exists a single matrix representation of the total behavior of any neural network if the response of its neurons can be represented by means of equation 1.

The neural network in the above example can be fully represented by the following matrix:

```
 0 0 0 -1  0
 0 0 0  1  1
 0 0 0  1  1
 0 0 0  0 -2
 0 0 0  0  0
```

where the first row is the threshold neuron, the weights of whose connections are the negatives of the thresholds of the remaining neurons in the network. This neuron should always fire. Rows 2 and 3 correspond to the input neurons, row 4 to the hidden neuron and row 5 to the output neuron.

In the case of the example, there are two input neurons (corresponding to rows 2 and 3 of our matrix). Neuron 1 (the threshold neuron) should always receive an input of 1, to make sure it fires. Rows 4 and 5, which are not input neurons, should always have an input value of zero. Therefore, the input vector for the example will be of the form:

```
1 x x 0 0
```

where x can be replaced by either zero or one. There are, therefore, four possible input vectors in the example:

```
1 0 0 0 0
1 0 1 0 0
1 1 0 0 0
1 1 1 0 0
```

The output of the network can be computed by means of the following, slightly simpler APL2 function:

```
[0]  Z←CONEC COMPUTE1 INPUT;A
[1]   Z←INPUT
[2]  L:A←Z
[3]   Z←(A+.×CONEC)>0
[4]   Z←INPUT+Z
[5]   →(~A≡Z)/L
```

Let us assume that the modified matrix of the example network is contained in variable MATRIX1. The following APL2 instructions compute the results of the network corresponding to all the possible inputs:

```
      IN←(1 0 0 0 0) (1 0 1 0 0) (1 1 0 0 0)
         (1 1 1 0 0)
      (⊂MATRIX1) COMPUTE1"IN
1 0 0 0 0  1 0 1 0 1  1 1 0 0 1  1 1 1 1 0
```

compatible with the result obtained with the matrix-vector representation.

ANALOGIC NEURONS

The neurons in the previous examples were digital, since their output can only be zero or one. Analogic neurons can produce other outputs, such as any number in the [0,1] interval. For example, a commonly used response function for neural networks is:

$$1/(1+e^{-\Sigma w_i x_i})$$

with appropriate corrections when the value obtained is too near one or zero. The following APL2 function computes the result of a neural network composed of neurons with this response function. The neural network is assumed to be fully represented by a single connectivity matrix.

```
[0]  Z←CONEC COMPUTE2 INPUT;A
[1]   Z←INPUT
[2]  L:A←Z
[3]   Z←÷1+*-A+.×CONEC
[4]   Z[(Z<0.2)/⍳⍴Z]←0
[5]   Z[(Z>0.8)/⍳⍴Z]←1
[6]   Z←INPUT+Z
[7]   →(~A≡Z)/L
```

LEARNING

We say that a neural network "learns" when it modifies its behavior in such a way that its response to a certain set of inputs adapts to another set of predefined "desired outputs".

There are different learning procedures that modify the weights of the connections of the neural network in such a way that the outputs get closer and closer to the desired values. These techniques require a teaching period during which the following happens:

1. One or several inputs are applied to the network.
2. The corresponding outputs are computed.
3. The outputs are compared to the desired outputs.
4. The weights of the connections are modified so that the outputs gets closer to the desired outputs.

The above process is repeated until the network behavior is acceptable.

One of learning procedures most used in neural networks is called "back propagation", because the weight corrections are applied to those output neurons in the last layer that differ from the desired value, and then the correction is propagated to the preceding layers. The following APL2 program executes a version of back propagation:

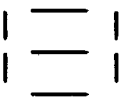
```
[0]  BKPROP1 I;INPUT;OUTPUT;O;OUT;E;d;NT;NO;
             ER;N;E1
[1]   E←0.02
[2]   NT←1++/N←LAYERS
[3]  L:'Input value: ',⍕IN[I;]
[4]   INPUT←1,IN[I;],(N[2]+N[3])⍴0
[5]   'Output value: ',⍕OU[I;]
[6]   OUTPUT←OU[I;]
[7]  L1:O←(-N[3])↑OUT←CONEC COMPUTE INPUT
[8]   ER←0.5×+/(E1←O-OUTPUT)*2
[9]   →(ER<1E¯10)/0
[10]  d←E×OUT×.×E1
[11]  NO←1+N[1]+N[2]+⍳N[3]
[12]  d←d×CONEC[;NO]≠0
[13]  CONEC[;NO]←CONEC[;NO]-d
[14]  E1←(-NT)↑E1
[15]  NO←(∨/d≠0)/⍳↑⍴d
[16]  d←E×OUT×.×(CONEC+.×E1)[NO]
[17]  d←d×CONEC[;NO]≠0
[18]  CONEC[;NO]←CONEC[;NO]-d
[19]  →L1
```

This program makes use of several global variables: CONEC is the matrix of the neural network. LAYERS is a vector that contains the number of neurons in each layer. IN is a matrix of possible inputs. Finally, OU is the desired set output values.

The program assumes that the number of layers in the network is three (this is the usual number). Some modifications would have to be done to apply a similar procedure to a perceptron or to a network with four or more layers.

A Learning Example

The preceding back-propagation program will be used to teach a neuron network to distinguish the numbers in a digital matrix similar to the following:



Each number will be obtained by lighting some of the slots in the preceding matrix, but not the others. For instance, number 8 will be obtained by lighting all the slots. Number 3 by lighting all the slots except the two vertical ones to the left. And so on.

The network will have seven input neurons in the first layer. Their input will be one if the corresponding slot is lighted, zero otherwise.

There will be ten output neurons in the third layer. Each output neuron will be trained to recognize one and only one number.

The hidden layer is unnecessary to solve this problem, which falls under the reach of perceptrons. However, since the back propagation program we are going to use requires three layers, we shall put a single neuron in the intermediate layer.

The following program generates the network:

```
[0]  Y NETWORK N;M;Z
[1]  ⍝ Generate a network with N[1] input
       elements,
[2]  ⍝ N[2] hidden elements and N[3] output
       elements
[3]   LAYERS←N
[4]   N[1]←N[1]+1
[5]   M←+/N
[6]   Z←((M,N[1])⍴0),((N[1],N[2])⍴Y),[⎕IO]
        ((N[2]+N[3]),N[2])⍴0
[7]   Z←Z,(((N[1]+N[2]),N[3])⍴Y),[⎕IO]
        (N[3],N[3])⍴0
[8]   Z[⎕IO;]←-Z[⎕IO;]
[9]   CONEC←Z
```

where Y is the initial value to be assigned to all the connections. The program generates the two global

variables LAYERS and CONEC. In our case we will execute

```
      1 NETWORK 7 1 10
```

The following program teaches this network to recognize the definitions of the ten digits in the digital matrix:

```
[0]  TEACHNRO;IN;OU;I;II
[1]   IN←2 7⍴1 1 1 0 1 1 1 0 0 1 0 0 1 0
[2]   IN←IN,[1]2 7⍴1 0 1 1 1 0 1 1 0 1 1
                 0 1 1
[3]   IN←IN,[1]3 7⍴0 1 1 1 0 1 0 1 1 0 1
                 0 1 1 1 1 0 1 1 1 1
[4]   IN←IN,[1]3 7⍴1 0 1 0 0 1 0 1 1 1 1
                 1 1 1 1 1 1 1 0 1 1
[5]   OU←10 10⍴1,10⍴0
[6]  L0:I←1+II←0
[7]  L:→(OU[I;]≡(-LAYERS[3])↑CONEC COMPUTE
        1,(+/LAYERS)↑IN[I;])/L1
[8]   BKPROP1 I
[9]   II←II+1
[10] L1:I←I+1
[11]  →(I≤↑⍴IN)/L
[12]  →(II≠0)/L0
```

The teaching algorithm is very simple. The back-propagation method is applied to the ten inputs one by one. At the end of the process, the first inputs are not guaranteed, since the changes to the network to learn the last numbers may have modified conditions for the first ones. Therefore, the learning procedure is repeated until all of the numbers are simultaneously learned.

The teaching process is finished in 2 minutes and 25 seconds, executing under APL2/PC (32 bit system) on an IBM PS/2 model 80 with the math co-processor. One hundred percent accuracy is obtained.

CONCLUSION

The preceding set of examples make it clear that emulating neural networks in APL2 is very simple. Very general networks (with or without feedback) can be emulated with a five line function. Learning procedures can also be implemented easily. Different methods can be tested and compared with a huge productivity.

The only extension to APL2 that could improve the treatment of neural networks would be the implementation of sparse matrices. This is due to the fact that the matrices representing neural networks are usually very large and most of their values are equal to zero (since neurons in the same layer do not have common connections).

REFERENCES

1. Hopfield, J.J. *Neural networks and physical systems with emergent collective computational abilities.* Proc. Nat. Acad. Sci. USA, Vol. 79, 2554-2558, 1982.
2. Minsky, M., Papert, S. *Perceptrons: an introduction to computational geometry.* MIT Press, Cambridge, Mass., 1965.