DEVELOPMENT OF A PERCEPTRON BASED ON ARTIFICIAL NEURAL NETWORK

Muhammad Nawaz Brohi
Siraj Muhammad Pandhiani

ABSTRACT

This paper presents a perceptorn which is an Artificial Neural Network (ANN) and also discusses the algorithm and basic learning rule for designing ANN which is based on Original Neural Network Architecture (ONNA) by using C++ for demonstration how to train a perceptron model.

Keywords: Perceptron; Neural Network; Training.

1.0 INTRODUCTION

Most of the neural network architectures, that were proposed and studied during the 1950s and 1960s were simple, single layered networks. During this period, early research was carried out by Warren McCulloch and Walter Pits (1943) on biological neuronal computations, by Frank Rosenblatt on basic artificial neural computers 1962 and by Bernard Widrow 1960 on adaptive neurons for engineering applications. Though other systems and architectures developed in that era like ADAline or MADAline were of significant importance, but the model given by Frank Rosenblatt, a psychologist at Cornell Aeronautical Laboratories was of the highest importance, due to the following reasons:

1. The perceptron learning algorithm is more powerful than the Hebb rule, considered de-facto standard at that time.

2. It can be developed in a fashion that the system either can be learnt by itself or with few changes, can be developed to get training from the user of surrounding environment.

1.1 The Original Perceptron

The original perceptron had three layers of neurons - sensory units, association units and a response unit - forming an exact model of a retina. The input to the perceptron was an array of light sensors arranged in a rectangular grid. The sensors were randomly connected (in a localized sense) to a layer of association units, which in turn were connected to a layer of response unit for given input patterns or class of patterns.

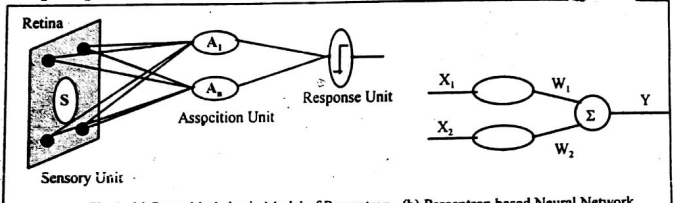

Fig-1: (a) Rosenblat's basic Model of Perceptron . (b) Perceptron based Neural Network

A simple perceptron is illustrated in Figure 1(a). The sensory, units act like transducers responding to physical signals such as light, pressure or heat. The association units receive random, localized and fixed connections from the sensory units. The association units send variable connections to the response units that act as the output units. Figure 1(b) explains the model of simple neural network based on original perceptron which also consists of three units: input, processing and output units. The input unit receives an input in the form of a vector. The inputs are then vectorally multiplied with corresponding weight values in processing unit. The output unit adds up the total response of the network.[2]

1.2 The Output Function

Looking at Figure 1(b), the output 'Y' is zero if the weighted sum inputs are equal or less than zero and equal

to the weighted sum (linear function) if the inputs are greater than zero. In other versions of the perceptron, the outputs is a step function with binary outputs values (0 or 1) or with threshold or bipolar values (-1 or 1).

2.0 BASIC PERCEPTRON ALGORITHMS

The original perceptron was essentially a single layer, feed forward, threshold logic unit. An equivalent is shown in Figure 2.

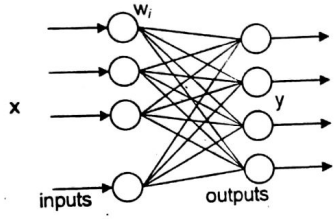

Figure 2. Single layer Perceptron

The inputs, outputs and training patterns are binary values (0 or 1) and the single layer of adjustable weights are real valued. The weights are on the connections leading to the output layer units. They are represented as a matrix of values W. Although there are different versions of the perceptron and corresponding learning rules, the basic rule is to alter the weights only when an error exists between the computed or actual network output and the correct or desired output. In that case, weights on active lines (inputs = 1) are incremented by a small amount when they should be 1 but are actually 0, and decremented by a small amount when they are 1 but should be 0. The amount of increment or decrement may be fixed value or may be proportional to the product of the error and input activation.

The perceptron learning theorem states that a

perceptron separates linearly separable pattern sets in finite iterations. Rosenblatt first proved perceptron learning theorem.

2.1 The Perceptron Basic Learning Rule is given as,

Step 0:
Initialize inputs, weights, and bias values.
Initialize learning rate α

Step 1:
While Stop condition is false, repeat steps 2-6

Step 2:
For each training pair, in the matrix of input variables, repeat
steps 3-5

Step 3:
Set activation of Input units ($X_i = S_i$)

Step 4:
Compute response of output unit.

$$y_{in} = b + \sum_j x_j + w_j$$

$$y = \begin{cases} 1 \rightarrow y_{in} > \theta \\ 0 \rightarrow -\theta \le y_{in} \le \theta \\ -1 \rightarrow y_{in} < -\theta \end{cases}$$

where θ is a threshold value.

Step 5:
Update Weights and bias if an error occurred for this pattern.

If $y \neq t$,
$w_i(new) = w_i(old) + \alpha\, t\, x_i$
$b(new) = b(old) + \alpha\, t$
else
$w_i(new) = w_i(old)$
$b(new) = b(old)$

Step 6:
Test the stop condition (if no weights changed in step 2 then stop else continue.

2.2 Perceptron Convergence Theorem

One of the most important achievements due to Rosenblatt is a convergence theorem for the above learning algorithm. The theorem states that if there is a set of weights that correctly classify the training patterns, then the learning algorithm will find one such 'W*' in a finite number of iterations. The assumptions here are the fact that at least one such set of weights exists and the number of training patterns is finite.

2.2.1 Theorem:

"If there is a weight vector W* such that f(x (p). W*) = t(p) for all p, then for any starting vector W, the percetron learning rule will converge to a weight vector, that gives the correct response for all training patterns, and it will do so in a finite number of steps".

The proof of this theorem is stated in [2.0 & 3.0].

3.0 LIMITATIONS OF PERCEPTRON

The single layer perceptron has several limitations, discussed below:

1. The output value of a perceptron can accept only one of the two (0 or 1) or (-1 or 1) value due to hard limiting transfer function.

2. The basic perceptron network can find solutions to problems in which a single plan can be drawn through the space, such that all of the items with a label 1 are on one side, all of the items with the label 0 are on the other side. In other words, we can say that a single layer perceptron can work only on *linear functions.*

3.1 Remedies

Minsky reported this as a very basic flaw in the perceptron theory and regarded this model as a failure. He was an influential personality and on his report, all the funds and aid to Neural Nets were stopped for over a

period of 20 years. Later, it was proved that a multi-layer perceptron could solve the binary two-dimensional problems or non-linear problems like the famous XOR problem.

4.0 MULTI LAYER PERCEPTRONS

A multi layer perceptron is shown in Figure 3.

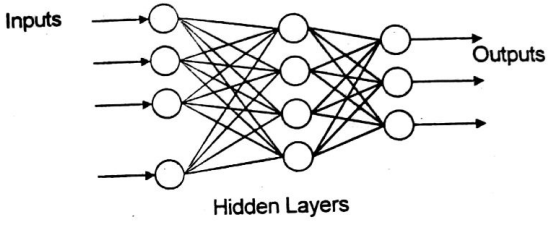

Figure 3. A general Multi layer Perceptron

The above figure shows a feed forward, fully connected hierachical network consisting of an input layer, one or more middle or hidden layers and an output layer. The internal layers are called "hidden", because they only receive internal inputs (inputs from other processing units) and produce internal outputs. Consequently they are hidden from outside world.

Many variations of the basic multi layer models have been proposed and study over the years show that these models are very useful in applications of pattern recognition, with addition to back propagation algorithms.

Neural Networks with multiple neuronal fields can often out estimate neural networks with only two neuronal fields. Minsky and Papert (1996) proved this for

two-layer perceptron threshold networks for some problems. Hornik and White confirmed this with their recent representation theorems.

5.0 APPLICATIONS OF PERCEPTRON

5.1 Logic Functions

Consider the AND function with binary input and bipolar target. In this case a single layer perceptron model can be trained within 8 - 9 training sets to give the correct outputs. Similarly the OR function is trained within 4 - 5 training sets. However the complex functions like XOR need multi level perceptron model to get the desired outputs. Usually this can be easily solved with a two layer model.

5.2 Application Using C

The following code demonstrate how to train a perceptron model while using C/C++:

```
    float X0,X1,X2;          //Inputs
    float W0,W1,W2;          //Weights
    float Y,Y_in;            //Outputs

    float T;                 //Target Output
    float alpha = 1;         //Learning Rate
    float theta = 0.2;       //Thresh-hold
    int iteration = 0;
    int max_iterations = 50;

    int AND_TABLE [4][4] = { {1,1,1,  1},  // { {1,1,1,  1},
                             {1,1,0, -1},     // {1,1,0,  1},
                             {1,0,1, -1},     // {1,0,1,  1},
                             {1,0,0, -1} };  // {1,0,0, -1} }; for OR fx.

main ()
{

  cout << "ITERATIONS       INPUTS          OUTPUTS          WEIGHTS  \n";
  cout << "i,j              X0,X1,X2        Y_in,Y           W0,W1,W2 \n";
  cout << "-------------------------------------------------------------\n";

  W0=0; W1=0; W2=0;  // Initialize Weights to zero.

  int stop = 0;       // Set Stop condition to false.

  while (!stop)
  {
    float W0old,W1old,W2old;
    W0old=W0; W1old=W1; W2old=W2;

    for (int i=0; i<4; i++)
    {
      if,((choice=='A')||(choice=='a'))
      {
    X0=AND_TABLE[i][0];
    X1=AND_TABLE[i][1];
    X2=AND_TABLE[i][2];
    T =AND_TABLE[i][3];
      }

      // step 2 of learning...

      Y_in= W0 + X1*W1 + X2*W2;

      if (Y_in > theta) Y=1;
      if ((Y_in < theta)&&(Y_in > (theta*(-1))) ) Y=0;
      if (Y_in < (theta*(-1))) Y=-1;

      cout <<iteration<<"."<<i<<"\t\t"<<X0<<" "<<X1<<" "<<X2<<"\t\t"<<Y_in<<"
           "<<Y<<"\t\t"<<W0<<" "<<W1<<" "<<W2<<"\n";
```

```
// step 3

if (Y != T)
    {
    W0 = W0 + T     ; // W1 = W1 + T*X1*alpha
    W1 = W1 + T * X1; // W2 = W2 + T*X2 alpha
    W2 = W2 + T * X2; // W3 = W3 + T*X3*alpha
    //  b = b + t;        // b = b + alpha*T
      };
    };

    iteration++;
    if ( W0==W0old && W1==W1old && W2==W2old ) stop = 1;
    if ( iteration > max_iterations )
    {
      cout << " No solution found after "<<iteration<<" iterations";
      exit(1);
    }
 }
 cout<<"\n\nFINAL WEIGHTS: W0="<<W0<<"; W1="<<W1<<"; W2="<<W2<<"\n";
 cout<<"Number of iterations: "<<iteration;
```

CONCLUSION

The developed perceptron is an Artificial Neural Network model which is based on original Neural Network Architecture and is shown how it can be used to approach one of the same problem to which we apply original Neural Network. It is parallel and simulated with human brain. We have also seen how to train the network (Perceptron) with the given algorithm. The algorithm is very useful and common training method for our developed perceptron. Finally, we developed training software in C++ for the running and operation of a designed perceptron.

REFERENCES

1. Gluck, M.A., Parker, D.B., and Reifsnider, E., *"Some Biological Implications of a Differential Hebbian Learning Rule"*, Psychobiology, Vol.16, No.3, 1988, Pp.298-302.
2. Hornik, K., Stinchcombe, M., and White, H., *"Multilayer Fed Forward Networks are Universal Approximators"*, Neural Networks, Vol.2, No.5, 1989, Pp.359-366.
3. McCulloch, W.S., and Pits, W., *"A Logical Calculus of the Ideals Immanent in Nervous Activity"*, Bulletin of Mathematical Biophysics, Vol.5, 1943, Pp.115-133.
4. Minsky, M.L., and Papert, S., *"Perceptrons: An Introduction to Computational Geometry"*, M.I.T. Press, Cambridge, MA, 1969, (2nd ed.,) 1988.
5. Rosenblatt, F., *"Principle of Neurodynamics"*, Spartan Books, New York, 1962.
6. Widrow, B., and Hoff, M.E., Jr., *"Adaptive Switching Circuits"*, IRE Wescon Convention Record, Pt.4, September, 1960, Pp.96-104.