International Journal on Recent and Innovation Trends in Computing and Communication
Volume: 4 Issue: 2
ISSN: 2321-8169
323 - 327

Implementation of Single Layer Perceptron Model using MATLAB

Poonam Gupta
Department of Mathematics, Hindu Girls College , Sonipat
Poonammittal2207@gmail.com

Parveen Mehta
Department of Computer Science, Hindu Girls College , Sonipat
Parveen_amb25@rediffmail.com

Abstract :- ANN consists of hundreds of single units, artificial neurons or processing elements . Neurons are connected with weights, which constitute the neural structure and are organized in layers. Perceptron is single layer artificial neuron network and it works with continuous or binary inputs. In the modern sense perceptron is an algorithm for learning a binary classifier. In ANN the inputs are applied via series of weights and Actual output are compared to the target outputs. Then to adjust the weihts, learning rule is used and bias the network so that actual output move closer to the target output .The perceptron learning rules comes under the category of supervised learning. In this Paper , implementation of single layer perceptron model using single perceptron learning rule through MAT LAB is discussed.

Keyword*: Perceptron , ANN, Sigmoid, Step function , FFNN.*

*****

I. Introduction:

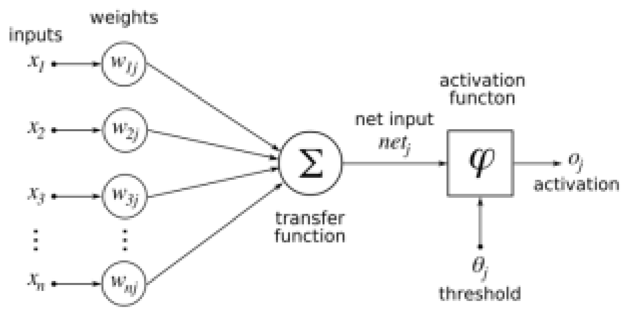
Artificial neural networks are computational models which are inspired by biological neural networks . They are used to approximate functions thatare generally unknown. In ANN, Artificial neurons are elementary units.Artificial neuron on receiving inputs summing them and produces an output. Activation function (Transfer function) is used to pass the sum. Activation Function may be of the type – Sigmoid function, Step function and piecewise linear functions. Usually they are monotonically increasing,continuous, differentiable and bounded.

Single layer perceptron network is simplest kind of neural network, consisting of single layer of output nodes . In this network, inputs are directly given to outputs via a series of weights. Due to this it can be considered as the simplest kind of feed forward network.

In 1943, Mc Culloch and Pitts [1] produced a model of the neuron which is still used today in artificial neural networking. Mc Culloch created computational models based on mathematical algorithms called threshold logic which splits the enquiry into two distinct approaches. One approach focused on biological processes in the brain and the other focused on application of neural network to artificial intelligence.

II. Activation Function

The artificial neurons receives one or more inputs and sums them to produce an output(Activation).Usually the sums of each node are weighted and Activation function is used to transform the activation level of a neuron into an output signal. There are many types of activation functions such as linear function , step function , sigmoid function , Ramp function ,Gaussian function , hyperbolic tangent .



Step Function

A **step function** is a **function** is likely used by the original Perceptron.This function produces two scalar output values depending on the threshold(ϑ). If input sum is above a certain threshold the output is 1 and if input sum is below a certain threshold the output is 0 .

$$f(netj)=\begin{cases} 1 \text{ if } netj \geq \theta \\ 0 \text{ if } netj < \theta \end{cases}$$

III. Artificial Neuron Learning(Training)

Training is the act of presenting the network with some sample data and modifying the weights to better approximate the desired function . There are two main types of training - Supervised Training and unsupervised training . In supervised learning both the inputs and outputs are provided . Then neural network processes the input and calculate an error based on its desired output and actual output. The weights are modified to reduce the difference(error) between the actual and desired outputs

A **feedforward neural network** (FFNN) is the simplest type of artificialneuralnetwork. FFNN consists of three layers – input layer , hidden layer and output layer.

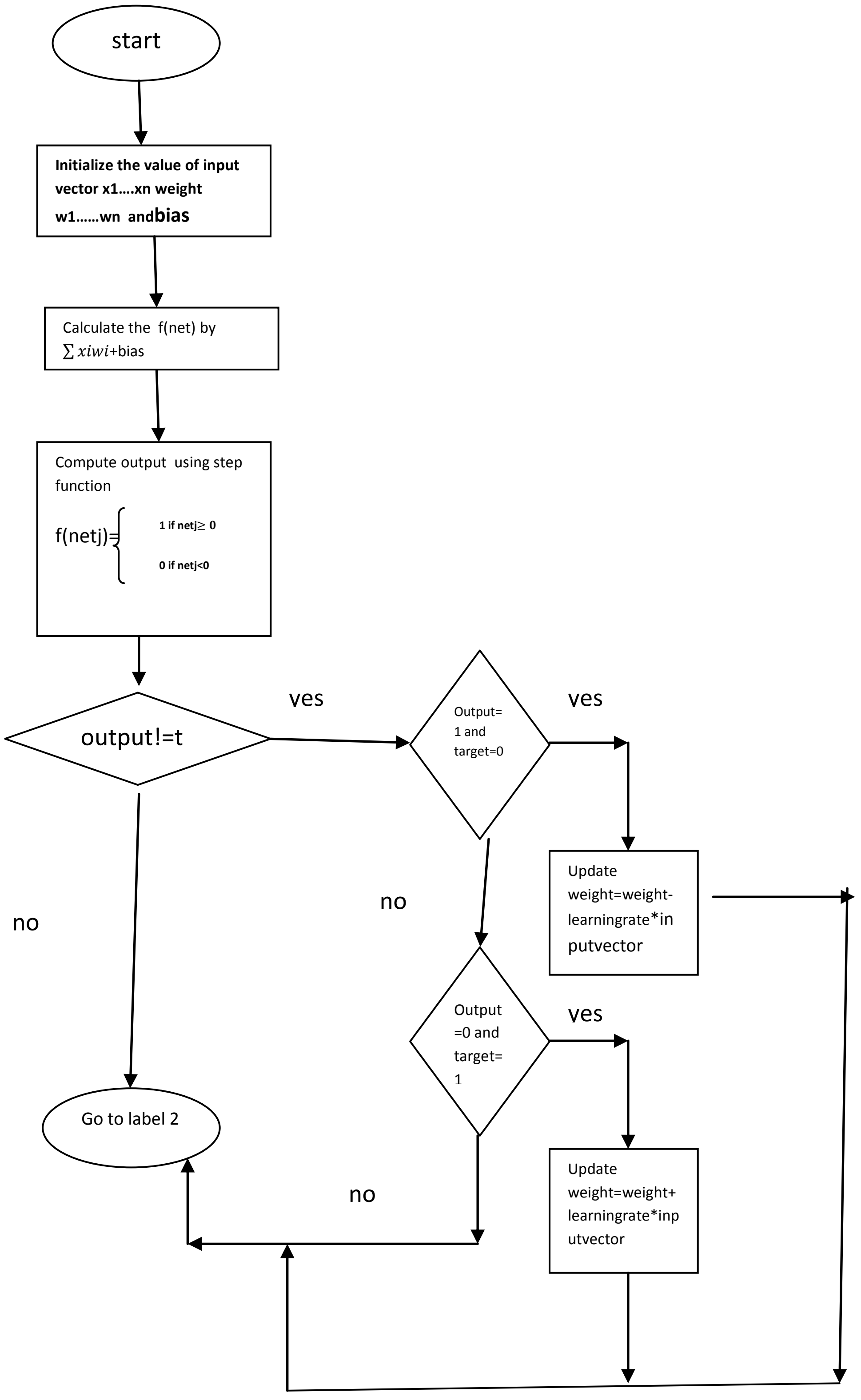
In this network, the information moves in only one direction, forward, from the input layer, through the hidden layers (if any) and to the output layer and the connection between the units don't form a cycle or loop . FFNN with monotonically increasing differentiable function can approximate any continuous function with one hidden layer , provided the hidden layer has enough hidden neurons.

IV. Single Layer Perceptron :

The simplest kind of neural network is a single layer perceptron network which consist of one or more artificial neurons in parallel and it can be considered the simplest kind of feedforward network as the inputs are given directly to the outputs via a series of weights

The sum of the products of the weights and the inputs is calculated in each node and calculated value is compared with the threshold value. If the calculated value is above the some threshold value typically 0, the neuron fires and takes the value typically 1(called activated value) otherwise it takes thevalue typically -1(called deactivated value). Neurons with this kind of activation function are also called *artificial neurons* or *linear threshold units*.

Single-unit perceptrons are only capable of learning linearlyseparable patternsAlthough a single threshold unit is quite limited in its computational power, it has been shown that networks of parallel threshold units can approximate any continuous function from a compact interval of the real numbers into the interval [-1,1]. This result can be found in Peter Auer, Harald Burgsteiner and Wolfgang Maass "A learning rule for very simple universal approximators consisting of a single layer of perceptrons"[7].


start
Initialize the value of input vector x1….xn weight w1……wn andbias
Calculate the f(net) by ∑xiwi+bias
Compute output using step function
f(netj)= 1 if netj≥ 0
0 if netj<0
output!=t
yes
Output=1 and target=0
yes
no
no
Update weight=weight-learningrate*inputvector
Output =0 and target=1
yes
Go to label 2
no
Update weight=weight+learningrate*inputvector

V. Perceptron learning Algorithm

Step1
Create a perceptron with (n+1) input neurons x0, x1…….xn. where b is the bias input let z be the computed(actual) output neurons , target output inialize [t0, t1….tn]
Step 2
Iniatlize weight w=(w0, w1, w2,…….wn)
To random weights.
Step3
Iterate through the inpute patterns xj of the training set using the weight set i.e compute the weighted sum of inputs net = $\sum xiwi$+b for each input pattern I .

Step 4

Compute output yj using step function z=f(net)=

$$\begin{cases} 1 \text{ if net} \geq \theta \\ 0 \text{ if net} < \theta \end{cases}$$

$$\theta \; is \;\; threshold \; value$$

Step 5
Compare the computed output with the target output t for each input pattern . if all the computed output matches with target output , then output display the weights and exit .
Step6
otherwise update the weights as given below z_i if the computed output is 1 but should have been 0 , then

$$w_i = w_i - \alpha x_i$$

wherei= 0 ,1,2,3,4,5…
If the computed output z_i is 0 and target output should bbe 1 then

$$w_i = w_i + \alpha x_i$$

wherei= 0 ,1,2,3,4,5…

step 7

go to step 3 and repeat

Implementation of single neuron layer suing mat lab

Equation used in mat lab

i. Calculating the output
As in mat lab it is easy to multiple one vector to other as like variable multiply
Input vector Multiple by the weight vector and add bias
sigma=bias + x(j,i)*w';

ii. Calculate the update weight lr=eta*x(j,i)
w(i)=w(i)-lr; w(i)=w(i)+lr;

Control structure for comparison used

```
if sigma(i)> theta
z(i)=1;
elseif sigma(i)<=theta
z(i)=0;
end
```

For loop used for iteration

```
for j= 1:2
        sigma=0;
fori=1 : 4
        sigma=bias + x(j,i)*w';
end
end
```

Displacement Function Are Used To Print Output On The Screen

```
disp('final calculation');
disp(sigma);
```

MAT LAB CODE

```
x=[0 0 1 1; 0 1 0 1];
%input variable pass
d=[1 1 0 0];
% target output
w=[-20 3 3 -5];
%initialize weight for per input
z=[0 0 0 0];
% vector to store the calculated value of the sigma input*weight + bias
bias=0.2;
%iniatlize of bias to store the value
%calculate the values total value ;
for j= 1:2
sigma=0;
fori=1 : 4
sigma=bias + x(j,i)*w';
end
end
disp('final calculation');
disp(sigma);
 % set the theta value for step function
theta=0.3;
fori=1:4
if sigma(i)> theta
z(i)=1;
elseif sigma(i)<=theta
z(i)=0;
```

```
end
end
disp('Finaloutput of the computed net value');
disp(z);
disp('oldweight');
disp(w);
```

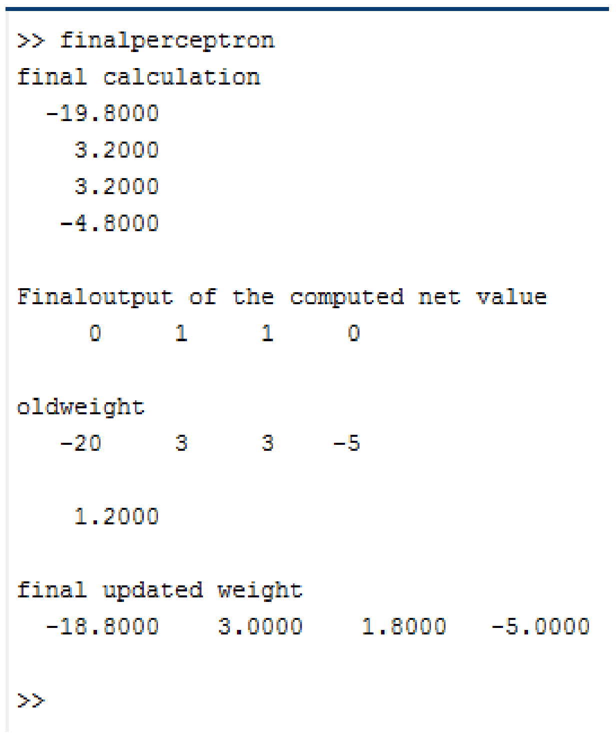
% updation to minimize the error

```
eta=1.2; % learning rate ;
for j= 1:4
lr=0;
fori=1 : 2
lr= x(i,j)*eta;
end
end
disp(lr);

fori=1:4
if and(z(i)==1 , d(i)==0)
w(i)=w(i)-lr;
elseif and(z(i)==0, d(i)==1)
w(i)=w(i)+lr;
end
end
```

%final weight

```
disp('final updated weight');
disp(w);
```



```
>> finalperceptron
final calculation
  -19.8000
    3.2000
    3.2000
   -4.8000

Finaloutput of the computed net value
     0     1     1     0

oldweight
   -20     3     3    -5

    1.2000

final updated weight
  -18.8000    3.0000    1.8000   -5.0000

>>
```

VI. Conclusion

The perceptron learning rule was originally developed by Frank Rosenblatt in the late 1950's. The perceptron learning rule provides a simple algorithm for training a perceptron neural network. In present paper we have considered the various steps in single layer perceptron model and only one epoch is used to update the weight so that error between actual output and target output can be minimized. However single layer perceptron network are not computationally complete. Much more work needs to be done to apply this research to multilayer network.

References

[1] Mc Culloch , Warren ;Walter Pitts(1943), "A logical calculus of ideas immanent in Nervous Activity" . Bulletin of Mathematical Biophysics, 5(4):115-1133.

[2] Ken Aizawa(2004), "Mc Culloch , Warren Sturgis" In : Dictionary of the Philosophy of Mind . Retrived May 17, 2008 .

[3] Freund, Y; Schapire, R.E.(1999). "Large margin classification using the perceptron algorithm"(PDF). Machine Learning . 37(3) :277-296.

[4] https://en.wikipedia.org/wiki/Types_of_artificial_neural_networks

[5] http://www2.econ.iastate.edu/tesfatsi/NeuralNetworks.CheungCannonNotes.pdf

[6] https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Activation_Functions

[7] Auer, Peter; Harald Burgsteiner; Wolfgang Maass (2008). "A learning rule for very simple universal approximators consisting of a single layer of perceptrons" (PDF). *Neural Networks*. **21** (5): 786–795. doi:10.1016/j.neunet.2007.12.036. PMID 182 49524.

[8] https://en.wikipedia.org/wiki/Feedforward_neural_network.