Binary classification of two gaussian clusters using perceptron

Sai Keerthy. K, Department of Electrical Engineering, Arizona State University, skakarl2@asu.edu

***Abstract*— In this project, training data is formed by two Gaussian clusters with variance 1 and centers (0,0) and (2.5,0) respectively. Then binary classification using perceptron is done by training the neural network with adapt function. It is followed by discussion about differences of adapt and train function.**

***Index Terms*—data, perceptron, training, classification.**

I. Introduction

In this project, training a single layer perceptron using adapt function is done for certain number of iterations. Then we test the network by creating random data set with same Gaussian distribution. We then plot a confusion matrix for train dataset and test dataset and analyze it.

II. Approach and Algorithm

To create input data for the network, we form two Gaussian clusters each of 1000 data points using randn function with variance 1, centers (0,0) and (2.5,0) respectively. The target variable consists of ones and zeros, with "+" for cluster with center (0,0) and "o" for cluster with center (2.5,0). Then the perceptron network is trained. The perceptron uses a hard-limit transfer function which gives us an output of 1 if its input is greater than 0 and an output of 0 if its input is less than 0 as shown in fig.1. We then plot the linear decision boundary. Then we generate 1000 test points for each cluster. Then, plot the confusion matrix for both training and test data which shows the number of points successful and failed in classification. The equation of the decision line is linear and of the form x'*w +b where x is the input vector, w is the weights vector and b is the bias vector.

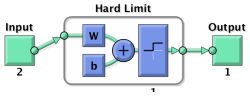

Fig. 1. A Perceptron with hard limit transfer function

III. RESULTS

As the given problem is not linearly separable, the perceptron doesn't converge and all the vectors will never be classified correctly. Here, we are using 7 iterations which is considered by calculating the mse() for different iterations and taking the least number for which mse() is reasonably less. As we are using adapt function, each time the input is sent in a sequence and the network is updated 2000 times (where 2000 is the number of input data points) for each iteration changing the values of weights and bias where as if the train function is used, it corrects the network based on sum of all individual corrections. By observing the confusion matrix of training data from fig.2, 42.7 % of data is correctly classified as 0 whereas the remaining 7.3% of data is wrongly classified as 1 when it should be 0. Similarly, we can see that 45.5% of data is correctly classifies as 1 and remaining 4.5% is wrongly classified as 0. From fig2. we can observe we got an accuracy of 89.0% for test data.

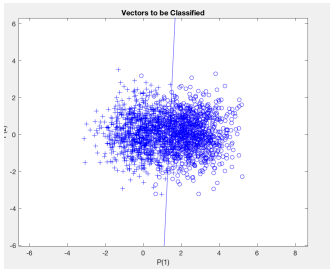

Fig 1: Two Gaussian distributions data points with linear decision boundary for binary classification

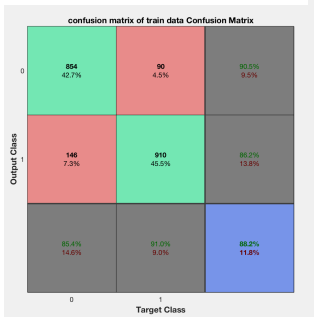

Fig. 2: Confusion matrix for training data

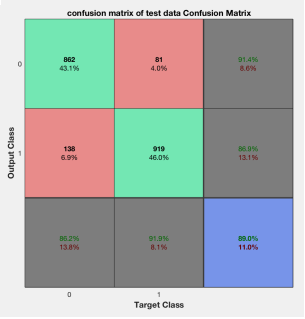

Fig. 3: Confusion matrix for test data

References

[1] Adapt neural network to data as it is simulated - MATLAB adapt - MathWorks United Kingdom", *Mathworks.com*. [Online]. Available: https://www.mathworks.com/help/nnet/ref/adapt.html.

[2] Perceptron Neural Networks - MATLAB & Simulink - MathWorks United Kingdom", *Mathworks.com*, 2017. [Online]. Available: https://www.mathworks.com/help/nnet/ug/perceptron-neural-networks.html.

[3] Linearly Non-separable Vectors - MATLAB & Simulink Example - MathWorks United Kingdom", *Mathworks.com*, 2017. [Online]. Available: https://www.mathworks.com/help/nnet/examples/linearly-non-separable-vectors.html.

MATLAB CODE

```
clear all
close all
% for repeatability
rng(5);
% Creating training data points
% gaussian cluster with center (0,0) and variance 1
k1 = randn(1000,2);
% gaussian cluster with center (2.5,0) and variance 1
k2 = [randn(1000,1)+2.5 randn(1000,1)];
% input data
X = [k1(:,1)' k2(:,1)';k1(:,2)' k2(:,2)'];
% target data
T = [ ones(1,1000) zeros(1,1000)];
plotpv(X,T);
% perceptron network
net = perceptron;
hold on
% drawing a line with weight IW and bias b
linehandle = plotpc(net.IW{1},net.b{1});
for a = 1:7
    % neural network training and adjusting the
decision boundary
   [net,Y,E] = adapt(net,X,T);
   linehandle = plotpc(net.IW{1},net.b{1},linehandle);
   drawnow;
end;
figure;
% plotting the confusion matrix of training data
plotconfusion(T,Y,'confusion matrix of train data')
view(net)
% for repeatability
rng(1);
% creating test data points
k11 = randn(1000,2);
k22 = [randn(1000,1)+2.5 randn(1000,1)];
x = [k11(:,1)' k22(:,1)';k11(:,2)' k22(:,2)'];
y = net(x);
% plotting the confusion matrix of test data
figure;
plotconfusion(T,y, 'confusion matrix of test data')
```