Classification of Gaussian Clusters Using Perceptron

Sai Prajwal Kotamraju
Department of Electrical Engineering
Arizona State University, Tempe, 85281, AZ
e-mail: skotamra@asu.edu

Abstract— **Data classification has been one of the important applications of a Neural Network. This paper aims at analyzing the results of classification of two 2-D Gaussian clusters centered at (0,0) and (2.5,0) with unit variance using a single Perceptron neural network. Training and Test accuracies for different activation functions have been tabulated and a comparison has been made between the performance of a Single Perceptron network and a Two-Layer Neural Network with one hidden layer of depth 10. It has been observed that a MLNN outperforms a single perceptron used with the best activation function.**

Keywords—Data Classification; Perceptron; Multi-Layer-Neural-Network; Activation Function;

I. Introduction

The idea of data classification using neural networks has been initiated long ago by the invent of Perceptron in 1957 by Frank Rosenblatt [1]. Although the network was able to classify simple data clusters which can be separated by a linear boundary, it failed to classify several trivial problems (such as Exclusive OR (XOR) classification problem) where a simple linear boundary can never separate the input clusters completely. This has been indicated in 1969 by Marvin Minsky and Seymour Papert in a monograph titled *Perceptrons*. However, adding a few more layers into the simple Perceptron architecture helped in eliminating a few limitations of the Perceptron [2].

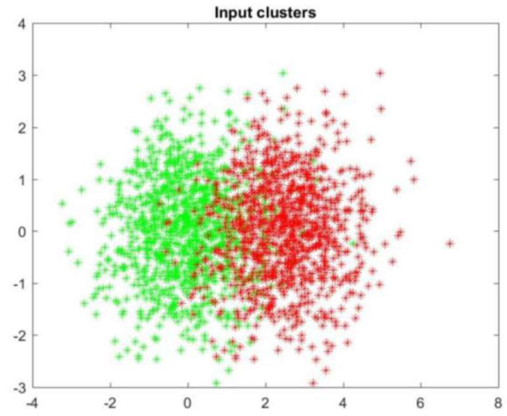

Fig. 1. Two Gaussian clusters of unit variance, each of 1000 data points, centered at (0,0) and (2.5,0) respectively.

In this paper, we aim at classifying two 2-D Gaussian clusters centered at (0,0) and (2.5,0) respectively, each having a unit variance (shown in Fig. 1), using a Simple Perceptron first and then using a MLNN created by stacking up a hidden layer of depth 10 to the former ANN. The green cluster has been assigned with a target value of *-1* while the red cluster is assigned with a target value of *+1*.

II. Functionality Of Perceptron

Perceptron is one of the earliest and simplest Neural Network architectures, proposed in 1957 [1]. It is based on an artificial neuron called the Linear Threshold Unit (LTU). Perceptron first calculates the weighted sum of inputs and then applies an activation function to the result obtained.

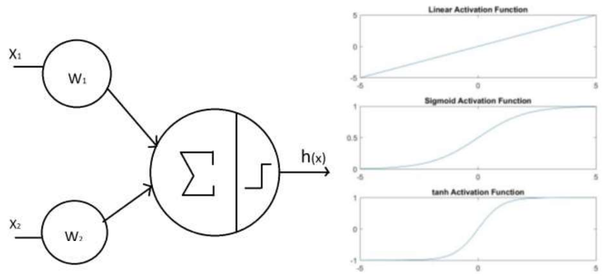

Fig. 2. Left- Simple Perceptron Neural Network with two inputs X_1 and X_2 with corresponding weights W_1 and W_2. Output of the neuron is h(x) for the given inputs and weight values; Right- Different activation functions used.

$$h(\boldsymbol{x}) = g(\boldsymbol{w}^T.\boldsymbol{x}) \tag{1}$$

In the above equation, $g(.)$ is the activation function while **w** and **x** represent weight and input vectors respectively. The three main activation functions used in general are: 1. Linear Activation Function, 2. Sigmoid Activation Function, and 3. Hyperbolic Tangent Activation Function.

Based on the difference between predicted and actual outputs for a given input, the weights get updated until the condition for error is met.

$$W_i^{(next\ step)} = W_i + \eta(h(\boldsymbol{x}) - t)X_i \tag{2}$$

In the equation (2), W_i is the weight associated with i^{th} input X_i and η is the learning rate. $h(\boldsymbol{x})$ is the output of the neuron for the current training instance $\boldsymbol{x}$ while t is the desired target value for the current training instance $\boldsymbol{x}$.

III. Experimental Results

A. *Single Perceptron with different Activations*

In this experiment, the training, and test-set accuracies have been calculated using different Activation Functions. From equation (2), it can be noted that the updated weight value is dependent upon desired target, inputs, and $h(\boldsymbol{x})$. The range and the values of $h(\boldsymbol{x})$ are dependent upon the activation function being used. Hence, by changing the Activation Function, there would be a change in the updated weights.

A random seed of 42 has been used to create two training clusters while a seed of 55 has been used to create two test clusters. The target values assigned are -1 and +1 respectively for the clusters centered at (0,0) and (2.5,0). The training and test set accuracies for different activation functions have been formulated in the table below.

TABLE I. Activation Functions VS Accuracies

Activation Function	**Training Accuracy**	**Test Accuracy**
Hard Limit	83.55%	83.80%
Sigmoid	82.95%	83.55%
Elliot Sigmoid	72.60%	73.15%
Hyperbolic Tangent	56.90%	56.7%

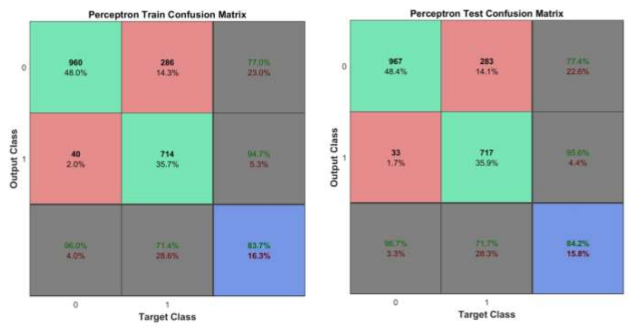

Fig. 3. Confusion Matrices for training and test sets with *hardlim* as the activation functions. Note that the target value *-1* has been changed to *0* as the plotconfusion(.) command doesn't accept negative target values.

B. *Multi-Layer NN VS Single Perceptron Neural Network*

As mentioned in [2], adding layers to the perceptron helps in eliminating a few of its' limitations. This is because of the non-linear boundary created by adding an additional hidden layer. In this experiment, a Multi-Layer Neural Network is created with 1 hidden layer of depth 10 in-between input and output layers. All the neurons in the hidden layers are assigned with *tanh(.)* activation function while the output layer neuron is activated with sigmoid activation function. This is compared with the Single Layer Perceptron with *sigmoid* as the activation function.

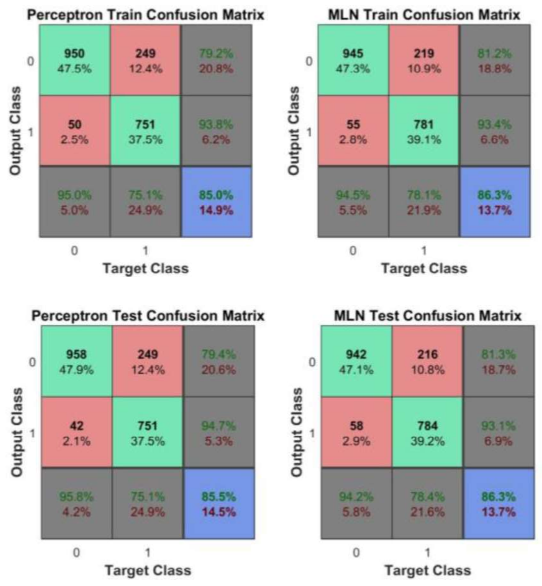

Fig. 4. Confusion Matrices for training and test sets for Single Perceptron and MLNN. Note that the accuracies of MLNN are more than the Single Perceptron.

IV. Conclusion

From this study, the following points can be concluded.

- A single perceptron can never classify with 100% accuracy in this case. This is because there is no linear boundary which could separate both the clusters completely.
- As both the clusters have unit variance, 89.43% of data points of first cluster fall on the left of (1.25,0) and 89.43% of data points of second cluster fall on to the right of (1.25,0). Rest of them fall on the opposite sides. So, our maximum accuracy cannot exceed 89.43% from a Perceptron network.
- Both *hardlim* as well as *sigmoid* activation functions resulted in a maximum accuracy around 83% for a single perceptron while the Multi-Layer-NN resulted in an accuracy around 87%. Hence, by increasing the number of layers, classification accuracy can be increased as there is scope for creation of a non-linear boundary.

References

[1] Rosenblatt, Frank (1957), The Perceptron--a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory.

[2] Rumelhart, David E., Geoffrey E. Hinton, and R. J. Williams. "Learning Internal Representations by Error Propagation". David E. Rumelhart, James L. McClelland, and the PDP research group. (editors), Parallel distributed processing: Explorations in the microstructure of cognition, Volume 1: Foundation. MIT Press, 1986.

```
clear;

%% Data Generation
clc; var = 1; mu1 = [0 0]; % Mean
sigma1 = [var 0; 0 var]; %Co-Variance vector
m1 = 1000;           % Number of data points.
mu2 = [2.5 0]; sigma2 = [var 0; 0 var];
m2 = 1000;
% Generate sample points with the specified means and covariance matrices.
rng(42);
R1 = chol(sigma1); X1 = randn(m1, 2) * R1; X1 = X1 + repmat(mu1, size(X1, 1), 1);
rng(42);
R2 = chol(sigma2); X2 = randn(m2, 2) * R2; X2 = X2 + repmat(mu2, size(X2, 1), 1);
X = [X1; X2];
figure(1);
% Display a scatter plot of the two distributions.
hold off;
plot(X1(:, 1), X1(:, 2), 'g*');
hold on;
plot(X2(:, 1), X2(:, 2), 'r*');
set(gcf,'color','white') % White background for the figure.
title('Input clusters');
% First, create a [10,000 x 2] matrix 'gridX' of coordinates representing
% the input values over the grid.
gridSize = 100; u = linspace(-6, 6, gridSize); [A, B] = meshgrid(u, u); gridX = [A(:), B(:)];
x = transpose(X);
t = [zeros(1,1000)-1 ones(1,1000)];

%% Training Perceptron
net = perceptron;
net.layers{1}.transferFcn = 'hardlim'; net.performFcn = 'mse'; net = train(net,x,t);
%view(net)
y = net(x);
y(y<=0.5)=-1; y(y>0.5)=1; %For logsig
%y(y<=0) =-1; y(y>0) = 1; %For tansig and elliotsig
%y(y==0)=-1; %For hardlim

%% Training MLNN
net1 = feedforwardnet; net1.layers{1}.transferFcn = 'tansig'; net1.layers{2}.transferFcn = 'logsig';
net1 = train(net1,x,t);
%view(net1);
y1 = net1(x); y1(y1<=0.5)=-1; y1(y1>0.5)=1;

%% Calculation of Training Accuracies
c = (y==t); c1 = (y1==t); train_acc = sum(c)/2000; train_acc_1 = sum(c1)/2000;

%% Testing with Random data
clc;
rng(55);
X1_test = randn(m1, 2) * R1; X1_test = X1_test + repmat(mu1, size(X1_test, 1), 1);
rng(55);
X2_test = randn(m2, 2) * R2; X2_test = X2_test + repmat(mu2, size(X2_test, 1), 1);
x_test = transpose([X1_test; X2_test]);
y_test = net(x_test);
y_test(y_test<=0.5)=-1; y_test(y_test>0.5)=1; %For logsig
%y_test(y_test<=0.5)=-1; y_test(y_test>0.5)=1; %For tansig and elliotsig
%y_test(y_test==0)=-1; %For hardlim
y1_test = net1(x_test);
y1_test(y1_test<=0.5)=-1; y1_test(y1_test>0.5)=1;
c_test = (y_test == t);
c1_test = (y1_test == t);
test_acc = sum(c_test)/2000;
test_acc_1 = sum(c1_test)/2000;

%% Results
disp('Perceptron');
fprintf("Training accuracy - %f\n Test accuracy - %f\n",train_acc*100,test_acc*100);
disp('MLN');
fprintf("Training accuracy - %f\n Test accuracy - %f\n",train_acc_1*100,test_acc_1*100);
%%%%%%%% Confusion Matrices %%%%%%%%
t(t==-1) = 0; y(y==-1) = 0; y1(y1==-1)=0; y_test(y_test==-1)=0; y1_test(y1_test==-1)=0;
figure; plotconfusion(t,y,'Perceptron Train',t,y1,'MLN Train');
figure; plotconfusion(t,y_test,'Perceptron Test',t,y1_test,'MLN Test');
figure; plotpv(x,t); plotpc(net.iw{1,1},net.b{1}); title('Perceptron Classification');
```