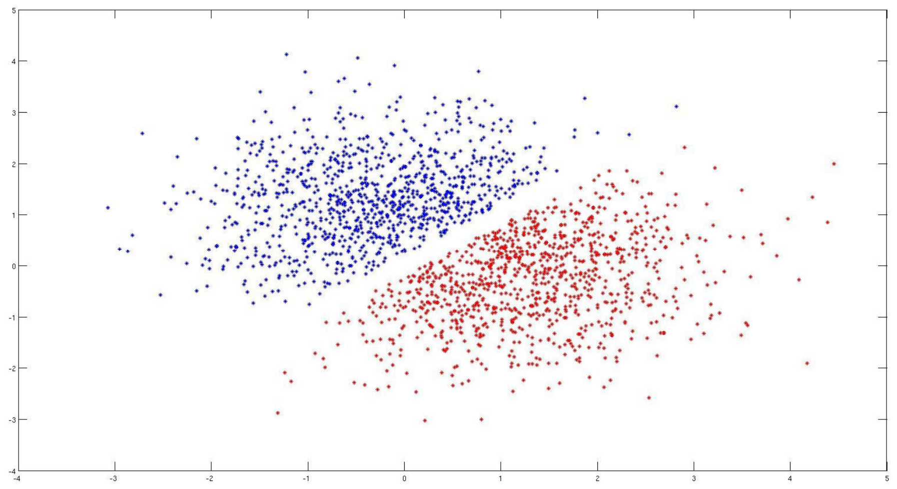
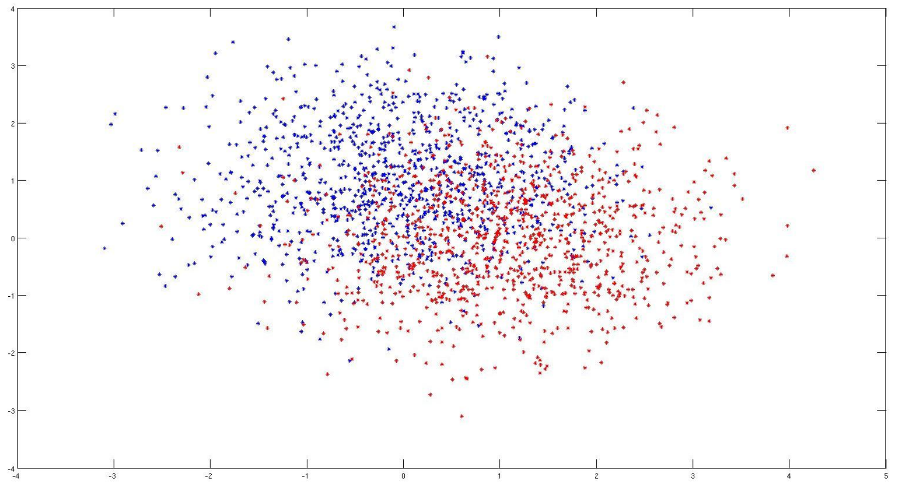
Single Perceptron Neuron

a.



Perception Test data Blue-Class1 Red-Class2



PDR Training data Blue-Class1 Red-Class2

The following can be seen as the plot or the given training data set. This is clear from the graph that the classes are linearly separable. That means when plotting the data on a graph, there is a hyper plane that clearly cuts through the classes and gives the boundary. In this types of cases, there is almost no error in drawing the boundary and there wouldn't be any outliers near the boundary as it has a good margin.

b. In the PDR testing. Again the data is plotted on the graph and it was observed that the classes lieoer each other and there can be no proper boundary see. They are very much mixed up, therefore it is difficult to plot a particular boundary that can divide them properly. Even if it is tried, there would be a huge error in division thus rendering the trial futile. From this testing it can be learnt that the data is better left alone and any fidgeting with it would lead to a huge error and thus failure of the neuron.

Epoch	Learning Rate (ŋ)	Training Class 1 errors	Training Class 2 errors	Test Class 1 Errors	Test Class 2 Errors
1	0.15	3	4	313	168
2	0.15	1	2	377	136
3	0.15	2	0	228	232
4	0.15	0	0	228	232

c. This experiment is done by running epochs using different types of weights and learning. But the most important thing that I have learnt through this experiment is that when changing the learning rate, I could see any direct change in the boundary. But the time take for the boundary to form changed with the rate of learning that is when the learning rate is high such as 0.4 the boundary was formed within 3 epochs. But when the learning rate was reduced to 0.1 the boundary wasn't formed until the 9th epoch. So from this I would like to analyze that the learning rate had no apparent effect on the quality of the boundary.

n	Errors
0.1	470
0.3	470
0.5	470
0.7	470
0.9	470
1	470

n:learning rate

Mathematically,

Using weight initial weights, the output can be given as v for the perceptron the activation function is $v = \sum(w.x) + w0$, w0is weight of bias

$y = f(v) = \{+1\ v >= 0, -1\ v < 0\}$.

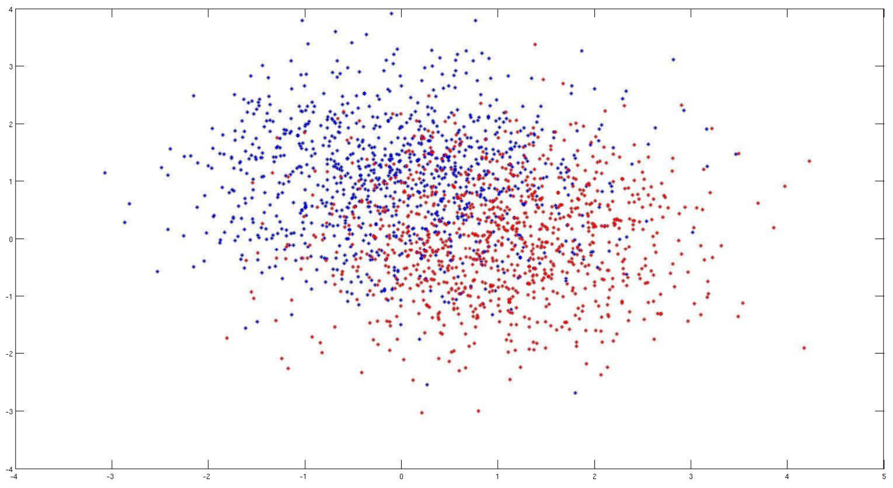
Now y is used to compare with the desired value d, if y is not equal to d then we perform the error correction .for the perceptron during the error correction we change the weights as follows

W_i (new)= w_i (old)+ $n*(d_i-y_i)*x_i$

d is desired value

Source code: perceptron.cpp (attached)

Part 2



The following can be seen as the plot or the given training data set. This is clear from the graph that the classes are linearly inseparable. So the delta tries to find an approximation of the boundary as it can't find the exact boundary. We have seen the plot of the test data earlier, which also linearly in separable. Now we try to find the approx. boundary using training data, we try to classify the test data.

d. In the PDR testing. Again the data is plotted on the graph and it was observed that the classes lie over each other and there can be no proper boundary see. They are very much mixed up, therefore it is difficult to plot a particular boundary that can divide them properly. Even if it is tried, there would be a huge error in division thus rendering the trial futile. From this testing it can be learnt that the data is better left alone and any fidgeting with it would lead to a huge error and thus failure of the neuron.

Epoch	Training class 1 errors	Training Class2 errors	Testing class 1 errors	Testing class2 errors
1	0	236	233	754
2	221	209	255	772
3	239	213	222	729
4	221	209	255	772
5	239	213	222	729
6	221	209	255	772

e. This experiment is done by running epochs using different types of weights and learning rates.

Mathematically,

Using weight as w_1 and w_2
The output of the activation function can be given as vFor the delta the activation function is
$y = \sum(wi.xi)$

For class 1 if y is less than 0, then we perform error correction. Similarly if y is >= d, for class 2 then we perform error correction.
Now y is used to compare with the desired value d, if y is not equal to d then we perform the error correction during the error correction we change the weights as follows

$W\ (new) = w\ (old) + n\sum_{i=0}^{j}(d_i - y_i)\ x_i$

Source code: delta.cpp (attached)