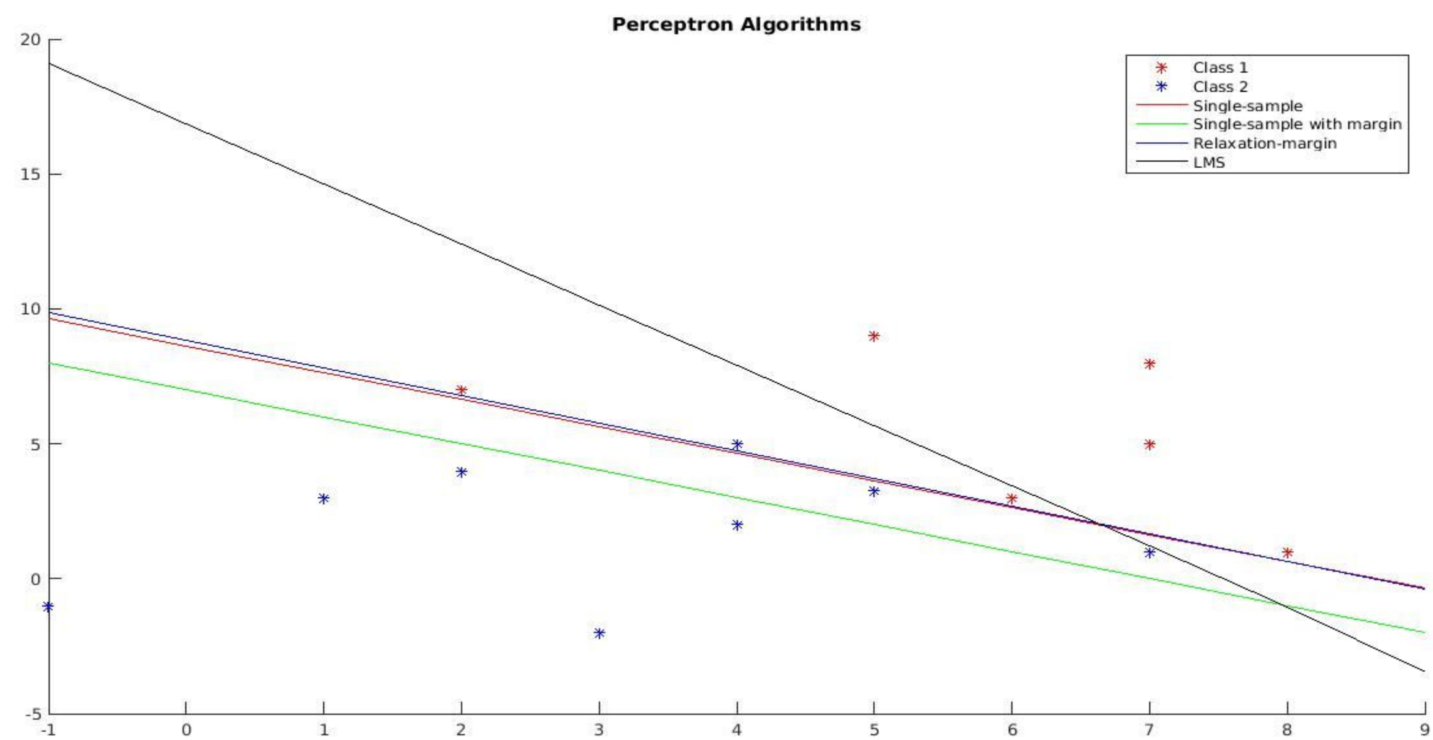
Assignment #1: LDFs and Neural Networks

CSE471: SMAI

Implementation : Matlab

Problem 1:



Part ii.

Algorithm	Weight Vector	Convergence time(s)
Single-sample perceptron	[1 , 1 , 1]	0.0679
	[0.5 , 0.5 , 0.5]	0.0674
	[4 , 4 , 4]	0.0692
	[-1 , -1 , -1]	0.0691
	[1 , 2 , 3]	0.0680

Single-sample perceptron with margin	[1 , 1 , 1]	0.1323
	[0.5 , 0.5 , 0.5]	0.1341
	[4 , 4 , 4]	0.1381
	[-1 , -1 , -1]	0.1444
	[1 , 2 , 3]	0.1328
Relaxation algorithm with margin	[1 , 1 , 1]	1.4322
	[0.5 , 0.5 , 0.5]	1.3065
	[4 , 4 , 4]	1.4536
	[-1 , -1 , -1]	0.0256
	[1 , 2 , 3]	0.0262
Widrow-Hoff/LMS Rule	[1 , 1 , 1]	3.3214e-05
	[0.5 , 0.5 , 0.5]	2.2763e-05
	[4 , 4 , 4]	2.1751e-05
	[-1 , -1 , -1]	2.1852e-05
	[1 , 2 , 3]	2.2267e-05

There is no direct trend between higher or lower values of initial weights and convergence times, hence we can say starting with a random value of initial weights and then adjusting the initial parameters would be a good strategy.

Part iii.

Single sample run with parameters :

init_wts = [ones(1,dim)];

eta = 0.0005;

Relation : direct proportion

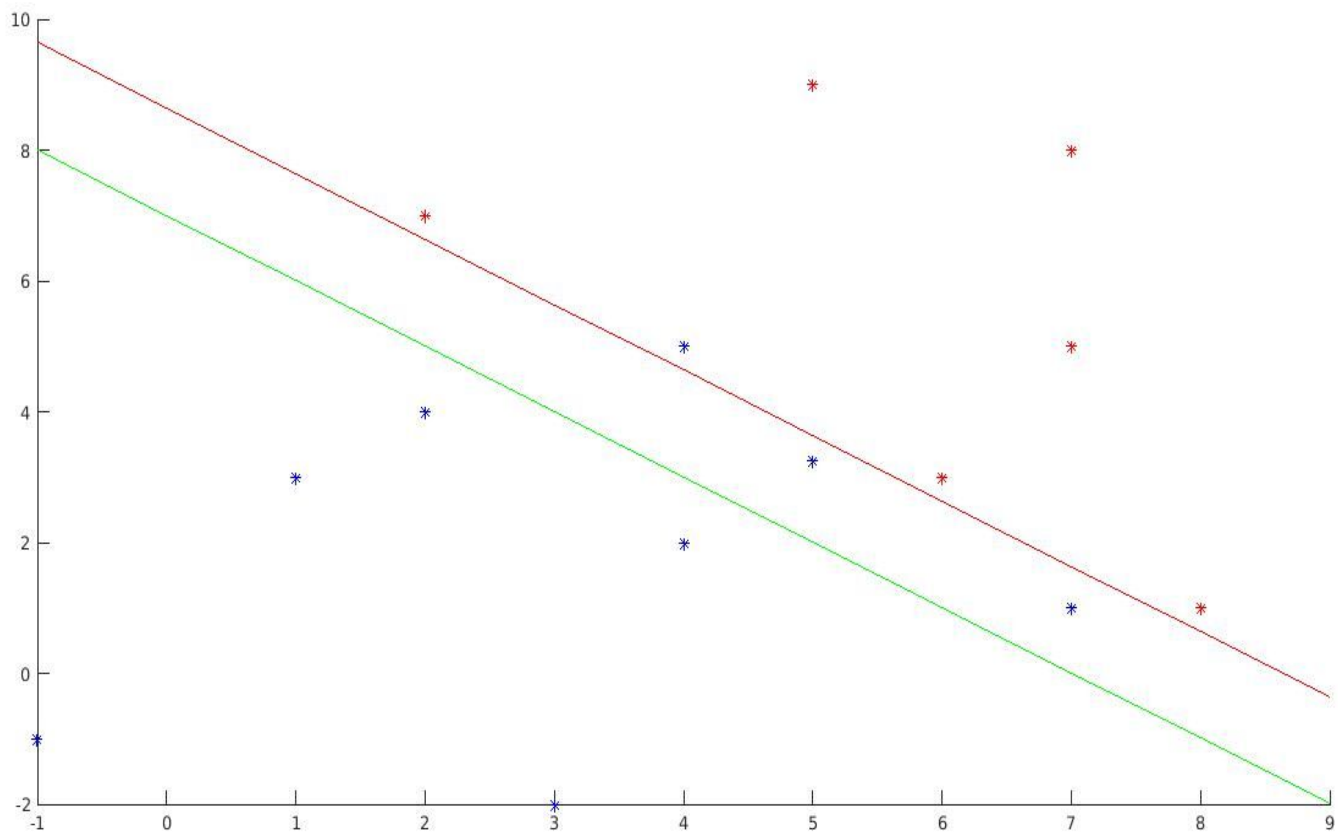
Perceptron Algorithm	Margin	Convergence Time (sec)
Single-sample perceptron with margin	0.001	2.3825e-04
	0.5	0.0038

	1	0.055
Relaxation algorithm with margin	0.1	0.340
	0.5	0.345.
	1	0.403

Hence we observe, If margin << η (k)*norm(Yk) , the amount by which a is updated increases by a(k)*y(k) , which is a small amount. If it is >> η (k)*norm(Yk), many corrections will be needed to satisfy the conditions $a^T * y < b$.

Part iii.

Varying parameters twice for non seperable point :



Part iv.

Initialization and augmentation of matrices :

```
%% Implement : %%%%%%%%%%%%%%%%%%%%%%%%
%% Single-sample perceptron
%% Single-sample perceptron with margin
%% Relaxation algorithm with margin
%% Widrow-Hoff or Least Mean Squared  LMS  Rule
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


C1 = [2, 7,1; 8, 1,1; 7, 5,1;  6, 3,1 ;  7, 8,1 ;  5, 9,1 ;  4, 5,1 ];                    %% class +1
C2 = [ 4, 2,1 ;  -1, -1,1 ;  1, 3 ,1;  3, -2 ,1;  5, 3.25 ,1;  2, 4 ,1;  7, 1 ,1 ];       %% class -1

X =[ C1;C2.*-1 ];
org_X = [C1;C2];
[no_of_samples, dim] = size( X );
figure;
scatter (org_X(1:7,1), org_X(1:7,2), 'r*'); hold on;
scatter (org_X(7:14,1), org_X(7:14,2), 'b*');
```

Typical Perceptron Algorithm Run :

```
hold on
init_wts = [ones(1,dim)];
eta = 1;
res = single_sample(X,init_wts,eta);
[class1,class2] = classify(org_X,res,no_of_samples,dim);
%% plot solution a
Xpts = [-1:9];
Ypts =( res(1)* Xpts + res(3) )/ res(2) ;
plot(Xpts,-Ypts,'-r','MarkerSize',10); hold on;
```

The general approach for any perceptron algorithm that we apply is :

1. Preprocess the values by augmenting it and normalizing.
2. According to the appropriate criterion function that is minized if ‘a’ is solution vector, get value of ‘a’
3. Classify using the new value of a.

A) **Single-Sample and Single-Sample with margin :**

```
function[res] = single_sample(X,init_wts,eta)
        [no_of_samples,dim] = size(X);
        a = init_wts;
        prev_a = zeros(1,dim);
        theta = 0.005;
        misclassified = 1;
        k =1;
        counter = 0;

        while(1)
                misclassified = 0;
                while(k<=no_of_samples)
                        counter = counter + 1;
                        Y = X(k,:);
                        if a*Y' <0
                                prev_a = a;
                                a = a + (eta.*Y);
                                misclassified = 1;
                                counter = 0;
                                break;
                        end
                        k = mod(k+1,no_of_samples);
                        if k == 0
                                k=14;
                        end
                        if (counter==14&&misclassified==0)
                                break;
                        end
                end
                if ((pdist([a;prev_a])<theta)||(misclassified==0))
                        break;
                end
        end
        res = a;
end
```

The single sample algorithm, rather than testing a(k) on every sample, considers the samples in a sequence and modifies the weight vector when it misclassifies a single sample. The inner while loop runs a cycle for every updated a. The misclassified flag indicates if after the updation of a, a misclassified Yk still remains. The **single sample with margin** algorithm is exactly identical except for the condition : if $a^T Y<$ margin to decide whether a sample was misclassified or not.

Result : In the graph obtained,

A = 0.9968   0.9984 0.9992

C) Single Sample with Relaxation :

```
function[res] =  relaxation_margin(X,init_wts,eta,margin,no_of_samples,dim)
        [no_of_samples,dim] = size(X);
        a = init_wts;
        prev_a = zeros(1,dim);
        theta = 0.005;
        misclassified = 1;
        k =1;
        counter = 0;

        while(1)
                misclassified = 0;
                while(k<=no_of_samples)
                        counter = counter + 1;
                        Y = X(k,:);
                        if a*Y' <0
                                prev_a = a;
                                term = (margin - (a*Y')/norm(Y))*Y';
                                a = a + (eta.*term');
                                misclassified = 1;
                                counter = 0;
                                disp (Y)
                                break;
                        end
                        k = mod(k+1,no_of_samples);
                        if k == 0
                                k=14;
                        end
                        if (counter==14&&misclassified==0)
                                break;
                        end
                end
                if ((pdist([a;prev_a])<theta)||(misclassified==0))
                        break;
                end
        end
        res = a;
end
```

This code implements the single-sample relaxation rule. The value of ‘term’ in the code is the distance from a(k) to the hyperplane $a^t y_k = b$. The loop iterates over all samples once, as soon as it finds a misclassified sample, the value of a is updated using term. We start counting elements that are present in the cycle for the Y_ks that are present **AFTER** misclassified one and circle back till we find that all samples are correctly classified.

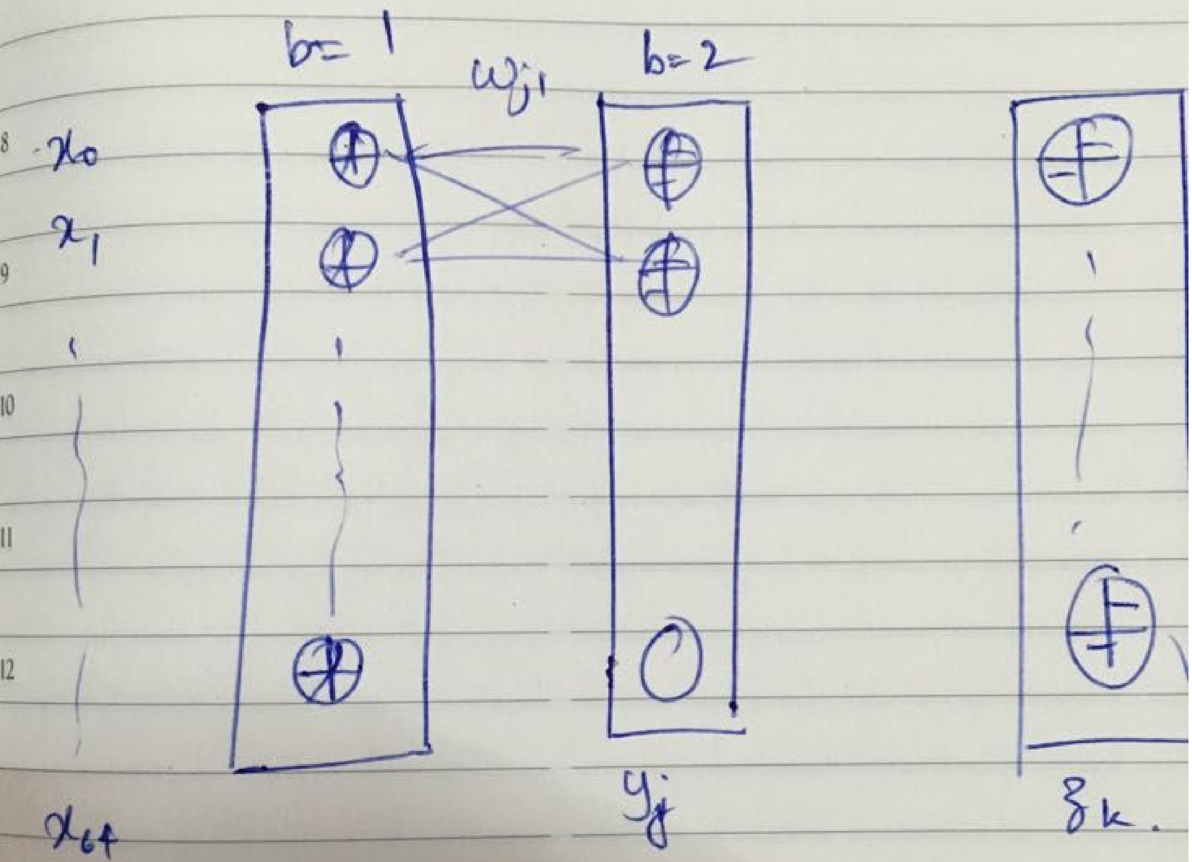
D) LMS :

```
function[a] = lms(X,a,b,theta,eta,no_of_samples,dim)
        k = 1;
        while(1)
                k = mod(k+1,no_of_samples);
                if k==0
                        k = 14;
                end
                Yk = X(k,:);
                Bk = b(k);
                term = (eta*(Bk - a*Yk')*Yk);
                a = a + term;
                if term'<theta
                        break
                end
        end
end
```

In this algorithm, we consider all sampled and not just the misclassified ones. Hence the code snippet doesn’t have the $a^t y$<margin as we observed before.
The error correction $\eta(k)(b_k - a^t y_k) * y_k < \theta$ is stored in the variable term. The loop is run until term < theta.

Problem 2 :

```
while(1)
        r = r+1;
        m = 0;
        while(m~=n)
                m =  m + 1;
                xm = select_pattern(input);
                dwij = dwij + eta*delta_j(j)*i ;
                dwjk = dwjk + eta*delta_k(k)*yj;
                if m == n
                        break;
                end
                wij = wij + dwij ; wjk = wjk + dwjk;
        end
        if(norm(w)<theta)
                break;
        end
end
```



64 - i/p

8 - No of hidden units

2 - O/p