Assignment 2

Abhinav Moudgil

201331039

Contents

Convergence proof for single sample perceptron

We need to show each iteration brings weight vector closer to solution region. Let the weight vector be a(k) in k_th iteration and solution vector be $\hat{a}$.

$$a(k+1) - \alpha\hat{a} = (a(k) - \alpha\hat{a}) + y^k$$

Taking norm and squaring,

$$\|a(k+1) - \alpha\hat{a}\|^2 = \|a(k) - \alpha\hat{a}\|^2 + 2(a(k) - \alpha\hat{a})^t y^k + \|y^k\|^2$$

Since y^k was misclassified, $a^t(k)y^k \leq 0$, thus

$$\|a(k+1) - \alpha\hat{a}\|^2 \leq \|a(k) - \alpha\hat{a}\|^2 + (\|y^k\|^2 - 2\alpha\hat{a}^t y^k)$$

Maximizing $(\|y^k\|^2 - 2\alpha\hat{a}^t y^k)$ will lead us closer to solution. Let

$$\beta^2 = max\|y\|^2$$

$$\gamma = min(\hat{a}y^k)$$

Above inequality reduces to,

$$\|a(k+1) - \alpha\hat{a}\|^2 \leq \|a(k) - \alpha\hat{a}\|^2 + \beta^2 - 2\alpha\gamma$$

Let us choose

$$\alpha = \frac{\beta^2}{\gamma}$$

we get

$$\|a(k+1) - \alpha\hat{a}\|^2 \leq \|a(k) - \alpha\hat{a}\|^2 - \beta^2$$

Each iteration reduces the difference by atleaset β^2. After k corrections,

$$\|a(k+1) - \alpha\hat{a}\|^2 \leq \|a(k) - \alpha\hat{a}\|^2 - k\beta^2$$

Hence, sequence of corrections must terminate after before or equal to k_0 corrections since squared distance can't be negative, where

$$k_0 = \frac{\|a(1) - c\hat{a}\|^2}{\beta^2}$$

Single sample perceptron

Method:

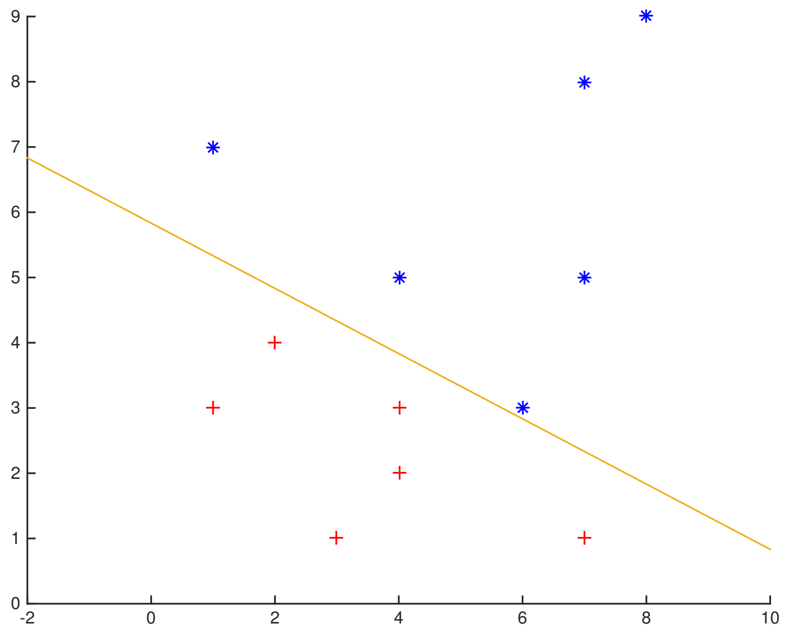
Weight vector for classification is updated each time we encounter a misclassified sample. This process is repeated over the training set until all samples are classified.

Code:

```
clear;
clc;
close all;
tic

x = [1 7; 6 3; 7 8; 8 9; 4 5; 7 5; 3 1; 4 3; 2 4; 7 1; 1 3; 4 2];
y(:, 2 : 3) = x;
y(:, 1) = 1;

% Normalization of vector spaces
y(7 : 12, :) = -y(7 : 12, :);

%Weight vector initialization
a = [1 1 1];

% Perceptron function
g = @(a, y) a * y';

figure
s = scatter(y(1 : 6, 2), y(1 : 6, 3), 25, 'b', '*');
hold on;
t = scatter(-y(7 : 12, 2),-y(7 : 12, 3), 25, 'r', '+');

k = 0;
p = -2:0.01:10;
n = size(y, 1);
while nnz((a * y') > 0) ~= n
```

```
    k = mod(k, n) + 1;
    yk = y(k, :);
    if (g(a, yk) <= 0)
        a = a + yk;
    end
end

% Exceptional Handling for a(3) = 0 (Vertical line)
if (a(3) ~= 0)
    q = (- a(2) * p - a(1))/a(3);
    plot(p, q);
else
    hx = -a(1)/a(2) * ones(1, 10);
    hy = 1 : 10;
    plot(hx, hy);
end

toc
```

```
Elapsed time is 0.061633 seconds.
```



Single sample perceptron with margin

Method:

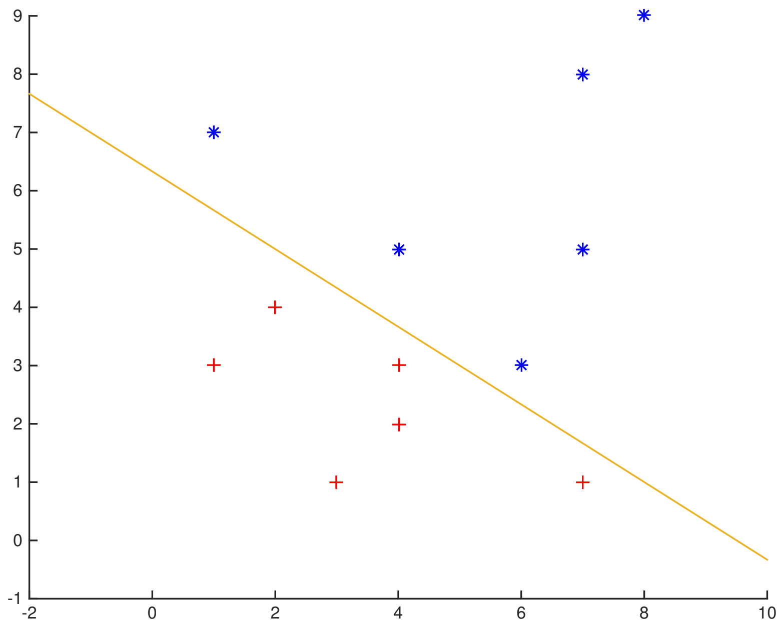
Single sample rule is followed along with margin 'b' make sure that points are not too close to decision boundry.

Code:

```
tic

x = [1 7; 6 3; 7 8; 8 9; 4 5; 7 5; 3 1; 4 3; 2 4; 7 1; 1 3; 4 2];
y(:, 2 : 3) = x;
y(:, 1) = 1;

% Normalization of vector spaces
y(7 : 12, :) = -y(7 : 12, :);

%Weight vector initialization
a = [1 1 1];

% Margin
b = -100;

% Perceptron function
g = @(a, y) a * y' + b;

%figure
%s = scatter(y(1 : 6, 2), y(1 : 6, 3), 25, 'b', '*');
%hold on;
%t = scatter(-y(7 : 12, 2),-y(7 : 12, 3), 25, 'r', '+');

k = 0;
p = -2:0.01:10;
n = size(y, 1);
while nnz(g(a, y) > 0) ~= n
    k = mod(k, n) + 1;
    yk = y(k, :);
    if (g(a, yk) <= 0)
        a = a + yk;
    end

end

% Exceptional Handling for a(3) = 0 (Vertical line)
if (a(3) ~= 0)
    q = (- a(2) * p - a(1))/a(3);
    plot(p, q);
else
```

```
    hx = -a(1)/a(2) * ones(1, 10);
    hy = 1 : 10;
    plot(hx, hy);
end

toc
```

```
Elapsed time is 0.627997 seconds.
```



Relaxation algorithm with margin

Method:

Following perceptron criterion function Jp is chosen:

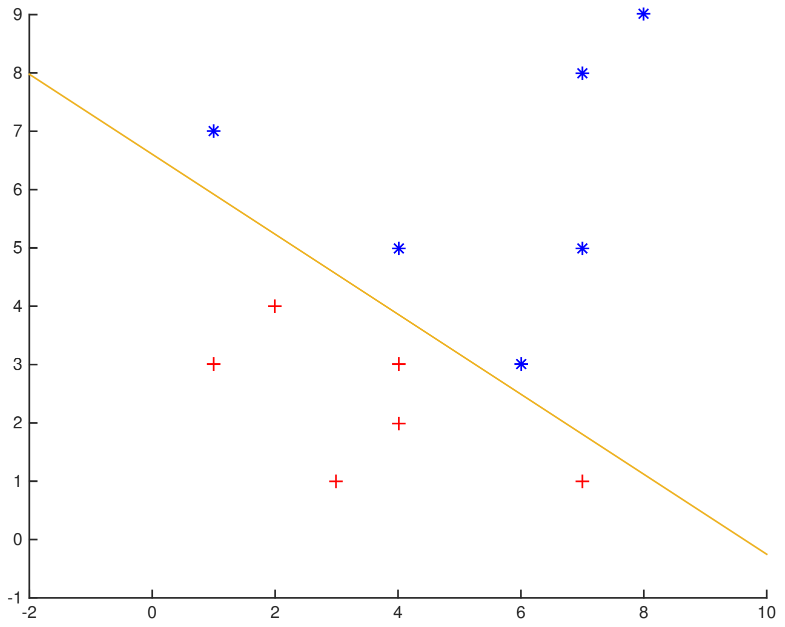
$$J_r(a) = \frac{1}{2} \sum_{y \in \gamma} \frac{(a^t y - b)^2}{\|y\|^2}$$

Its gradient is more continuous and smooth. Since longest sample vector can dominate the perceptron criterion function, hence normalization is done.

```
tic
x = [1 7; 6 3; 7 8; 8 9; 4 5; 7 5; 3 1; 4 3; 2 4; 7 1; 1 3; 4 2];
```

```
y(:, 2 : 3) = x;
y(:, 1) = 1;

% Normalization of vector spaces
y(7 : 12, :) = -y(7 : 12, :);

%Weight vector initialization
a = [1 1 1];

%Margin
b = 100;

% Perceptron function
g = @(a, y) a * y' - b;

%figure
%s = scatter(y(1 : 6, 2), y(1 : 6, 3), 25, 'b', '*');
%hold on;
%t = scatter(-y(7 : 12, 2),-y(7 : 12, 3), 25, 'r', '+');

k = 0;
p = -2:0.01:10;
n = size(y, 1);

eta = 2.1;
while nnz(g(a, y) > 0) ~= n
    k = mod(k, n) + 1;
    yk = y(k, :);
    if (g(a, yk) <= 0)
       a = a - ((eta * g(a, yk))/(norm(yk)^2)) * yk;
    end
end

% Exceptional Handling for a(3) = 0 (Vertical line)
if (a(3) ~= 0)
    q = (- a(2) * p - a(1))/a(3);
    plot(p, q);
else
    hx = -a(1)/a(2) * ones(1, 10);
    hy = 1 : 10;
    plot(hx, hy);
end

toc
```

```
Elapsed time is 0.024712 seconds.
```



Widrow-Hoff or Least Mean Squared (LMS) Rule

Method:

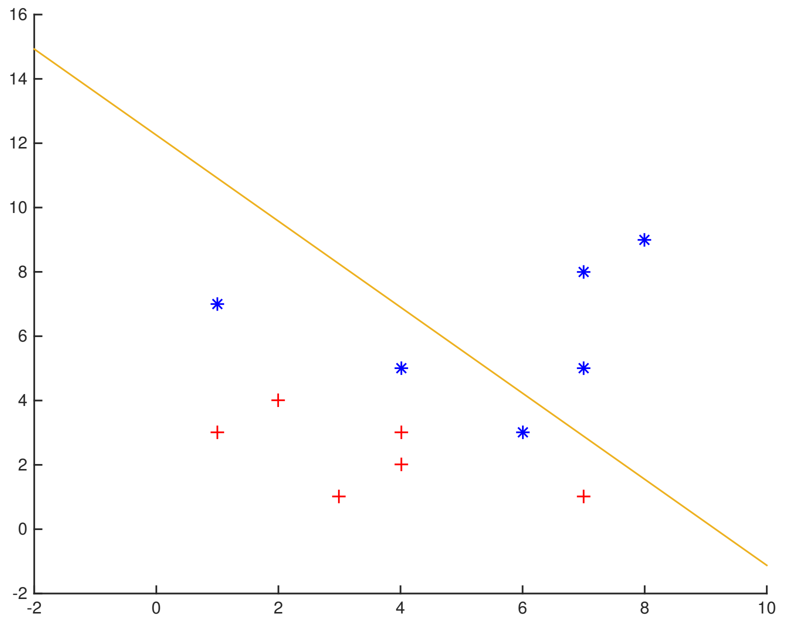
In this procedure, we consider all data samples rather than misclassified ones. Margin vector 'b' is taken. This procedure might not yield a seperating hyperplance but a reasonable one.

Code:

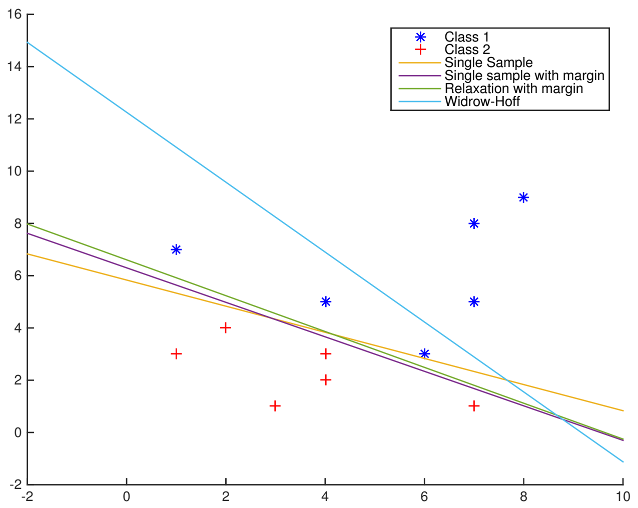
```
tic

x = [1 7; 6 3; 7 8; 8 9; 4 5; 7 5; 3 1; 4 3; 2 4; 7 1; 1 3; 4 2];
y(:, 2 : 3) = x;
y(:, 1) = 1;

% Normalization of vector spaces
y(7 : 12, :) = -y(7 : 12, :);

%Weight vector initialization
a = [1 1 1];

%Margin
b = 10;
```

```
% Perceptron function
g = @(a, y) a * y' - b;
rownorm = @(x,p) sum(abs(x).^p,2).^(1/p);

%figure
%s = scatter(y(1 : 6, 2), y(1 : 6, 3), 25, 'b', '*');
%hold on;
%t = scatter(-y(7 : 12, 2),-y(7 : 12, 3), 25, 'r', '+');

k = 0;
p = -2:0.01:10;
n = size(y, 1);

theta = 1 * ones(12, 1);
eta = 0.5;
count = 1;
while nnz(rownorm(((eta/count) * repmat(g(a, y)', 1, 3) .* y), 2) < theta) ~= n
    k = mod(k, n) + 1;
    yk = y(k, :);
    a = a - ((eta/count) * g(a, yk)) * yk;
    count = count + 1;
end

% Exceptional Handling for a(3) = 0 (Vertical line)
if (a(3) ~= 0)
    q = (- a(2) * p - a(1))/a(3);
    plot(p, q);
else
    hx = -a(1)/a(2) * ones(1, 10);
    hy = 1 : 10;
    plot(hx, hy);
end

toc


hold off
```

```
Elapsed time is 1.578698 seconds.
```



Combined Plot



Effect of initialization

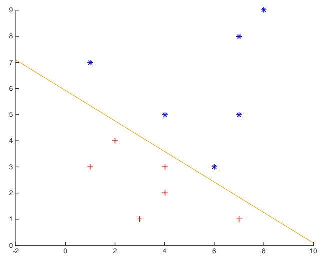
Weights are initialized randomly within a certain range. Average convergence time for certain ranges of weights are tabulated below:

Initial Weights Range	Single sample	Single sample with margin	Relaxation with margin	Widrow Hoff
$[-1, 1]$	0.068177	0.570688	0.070865	9.170230
$[1000, 1000000]$	50.107077	17.411268	0.073862	30.43232
$[0, 2]$	0.082015	0.568841	0.075528	6.448462
$[-100000, 100000]$	3.703376	6.654690	0.068799	20.32312

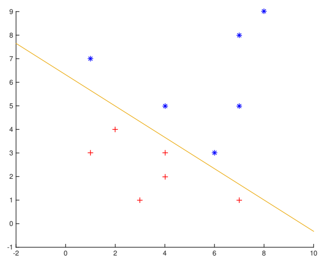
Convergence time also depends on margin. Above convergence time are calculated with margin = 100. LMS has highest time of convergence in above table. It is not because of weight initialization. Initial value of learning and threshold for error also plays a very important role in convergence. In LMS, low threshold for error is set to yield maximum classification possible.

Effect of margin

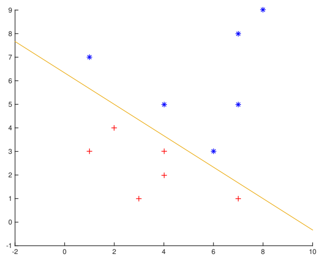
Singe sample with margin



Margin = 5

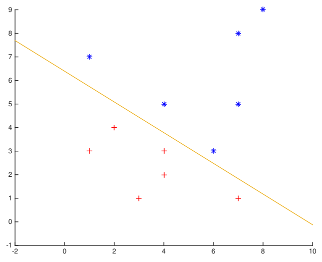


Margin = 100

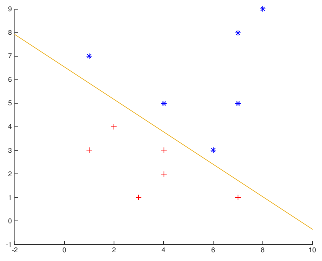


Margin = 1000

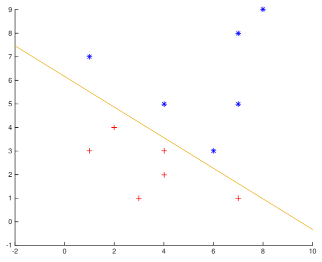
Relaxation with margin



Margin = 5



Margin = 100



Margin = 1000

Following are the convergence time(in seconds) for both the algorithms:

Margin	Single sample with margin	Relaxation with margin
5	0.126212	0.079206
100	0.570688	0.086744
1000	5.050803	0.095713

Neural network for handwritten digit classification

Dataset: Optical Recognition of Handwritten Digits Data Set

https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits
Dataset was trimmed to obtain only training and test samples of class 0 and 7.
This was done using single bash command.

Preprocessing: 32x32 bitmap images are processed to 8x8 data using by counting number of pixels in 4x4 blocks. Thus each sample contains 64 features and one class (0 or 7).

Classifier:
Optimal Model: 64-70-2
Different number of hidden units were tested to obtain the most optimal model that has average accuracy of 99.72% for this dataset. It also agrees with general rule of m/10 hidden units where m is the number of training samples.
Bias units are taken both at input and hidden layer.
Feed forward operation function:

Code:

```
%% Neural Network with one hidden layer
% Neural network for handwritten digit classification
% Model of neural network - 64-70-2

clear;
clc;
close all;

%% Training Phase
tic

filename = 'train.txt';
X = dlmread(filename, ',');
n = size(X, 1); % Number of training samples
t = X(:, size(X, 2));
%X = X(:, 1 : (size(X, 2) - 1));
X(:, size(X, 2)) = 1;
X = X';
```

```
% Initialization by feedforward operation
%bias1 = 1;
%bias2 = 1;
nHiddenUnits = 70;
f = @(x) logsig(x);
df = @(x) logsig(x) .* (1 - logsig(x));

a = -1/sqrt(size(X, 1));
b = 1/sqrt(size(X, 1));
wji = (b - a) .* rand(nHiddenUnits, size(X, 1)) + a;
netj = wji * X;

% Apply fnet from 1 : 32
yj(1 : nHiddenUnits - 1, :) = f(netj(1 : nHiddenUnits - 1, :));
yj(nHiddenUnits, :) = 1;
a = -1/sqrt(size(netj, 1));
b = 1/sqrt(size(netj, 1));
wkj = (b - a) .* rand(2, size(yj, 1)) + a;
netk = wkj * yj;
zk = f(netk);

% Backpropagation operation
tk(:, 1) = t < 6;
tk(:, 2) = t > 6;
tk = tk';
Jw = 0.5 * sum((tk - zk) .^ 2); % Gives error for each training sample in each column
delk = (tk - zk) .* df(netk);
delj = df(netj) .* (wkj' * delk);

eta = 1;
k = 0;
theta = 0.7;
while k < n
    k = mod(k, n) + 1; %Kth training sample
    xk = X(:, k);
    netj(:, k) = wji * X(:, k);
    yj(1 : nHiddenUnits - 1, k) = f(netj(1 : nHiddenUnits - 1, k));
    yj(nHiddenUnits, k) = 1;
    netk(:, k) = wkj * yj(:, k);
    zk(:, k) = f(netk(:, k));
    delk(:, k) = (tk(:, k) - zk(:, k)) .* df(netk(:, k));
    % delj(:, k) = df(netj(:, k)) * delk(:, k)' * wkj(:, k);
    delj(:, k) = df(netj(:, k)) .* (wkj' * delk(:, k));
    wkj = wkj + eta * delk(:, k) * yj(:, k)';
    wji = wji + eta * delj(:, k) * xk';
    % Updating all values according to new weights

    netj = wji * X;
    yj(1 : nHiddenUnits - 1, :) = f(netj(1 : nHiddenUnits - 1, :));
    yj(nHiddenUnits, :) = 1;
    netk = wkj * yj;
    zk = f(netk);

    % Error
    Jw(k) = 0.5 * sum((tk(:, k) - zk(:, k)) .^ 2);
end


%% Testing Phase


clear tk zk
filename = 'test.txt';
X = dlmread(filename, ',');
n = size(X, 1); % Number of test samples
t = X(:, size(X, 2));
% X = X(:, 1 : 64);
X(:, size(X, 2)) = 1;
X = X';
zk = zeros(2, size(X, 2));
tk(:, 1) = t < 6;
tk(:, 2) = t > 6;
tk = tk';
k = 0;

while k < n
    k = mod(k, n) + 1;
    xk = X(:, k);
    netj(:, k) = wji * X(:, k);
    yj(1 : nHiddenUnits - 1, k) = f(netj(1 : nHiddenUnits - 1, k));
    yj(nHiddenUnits, k) = 1;
    netk(:, k) = wkj * yj(:, k);
    zk(:, k) = f(netk(:, k));
end
zk = zk > 0.5;
err = tk - zk;
numberofErrors = sum(max(err ~= 0))
accuracy = (size(X, 2) - (numberofErrors))/size(X, 2)

toc
```

Observations:

1. Accuracy increases with increase in number of hidden units. But this might lead to over-fitting of data.
2. Without bias units, netj will have very high value, which implies yj (=f(netj)) will saturate to 1 if weights are positively initialized. Thus, f'(netj) will be 0 and weghts will not get updated.
3. Classification can also be done without bias units. Do normalization in each feature. This way, value of netj will never overshoot and we'll get average accuracy of around 99%.
4. Convergence time increases with increase in number of hidden units.