EE 5904 Neural Network Homework 1

May 10, 2023

0.1 Question 1

The input vector:

$$x = [+1, x_1, x_2, x_3, ..., x_m]$$

The weight vector:

$$w = [b, w_1, w_2, w_3, ..., w_m]$$

The induced local field v:

$$v = \sum_{i=1}^{m} w_i x_i + b = w^T x$$

(1)when the decision boundary is :

$$\phi(v) = \zeta = av + b$$

it can be expressed as:

$$a(\sum_{i=1}^{m} w_i x_i + b) + b = \zeta$$

it is a hyperplane

(2)when the decision boundary is:

$$\phi(v) = \zeta = \frac{1}{1 + e^{-}2v}$$

$$v = -\frac{1}{2} ln \frac{1 - \zeta}{\zeta}$$

since ζ is a constant value, v is also a constant value, which can be expressed as $C = -\frac{1}{2} ln \frac{1-\zeta}{\zeta}$

$$\sum_{i=1}^{m} w_i x_i + b - C = 0$$

it is a hyperplane

(3)when the decision boundary is:

$$\phi(v) = \zeta = e^{-}\frac{v^2}{2}$$

$$v = \pm\sqrt{-2ln\zeta}$$

so

$$\begin{cases} \sum_{i=1}^{m} w_i x_i + b + \sqrt{-2ln\zeta} = 0 \\ \sum_{i=1}^{m} w_i x_i + b - \sqrt{-2ln\zeta} = 0 \end{cases}$$

it is not a hyperplane

0.2 Question 2

Proof:
Assume XOR is in early separable, which means there is a decision boundary can be expressed as:

$$\sum_{i=1}^{m} w_i x_i + b = w_1 x_1 + w_2 x_2 + b = 0$$

Then we set the threshold at zero, which means:

$$\begin{cases} w_1 x_1 + w_2 x_2 + b \leq 0 & y = 0 \\ w_1 x_1 + w_2 x_2 + b > 0 & y = 1 \end{cases} \tag{1}$$

Take all conditions into (1):

x_1	0	1	0	1
x_2	0	0	1	1

$$b \leq 0 \tag{2}$$
$$w_1 + b > 0 \tag{3}$$
$$w_2 + b > 0 \tag{4}$$
$$w_1 + w_2 + b \leq 0 \tag{5}$$

From (3) and (4) we can get:

$$w_1 + w_2 + 2b > 0 \tag{6}$$

From (2) and (5) we can get:

$$w_1 + w_2 + 2b \leq 0 \tag{7}$$

Since (6) and (7) can not exist at the same time, the assumption is invaid.
So XOR is not linearly separable.

0.3 Question 3

a)
(1) AND:

$$v = \begin{bmatrix} -1.5 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \qquad y = \begin{cases} 0 & if\ v < 0 \\ 1 & if\ v \geq 0 \end{cases}$$

(2) OR:

$$v = \begin{bmatrix} -0.5 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \qquad y = \begin{cases} 0 & if\ v < 0 \\ 1 & if\ v \geq 0 \end{cases}$$

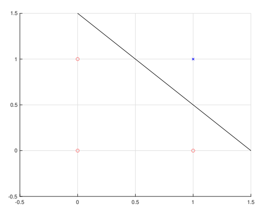
(3) COMPLEMENT:

$$v = \begin{bmatrix} 0.5 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ x \end{bmatrix} \qquad y = \begin{cases} 0 & if\ v < 0 \\ 1 & if\ v \geq 0 \end{cases}$$

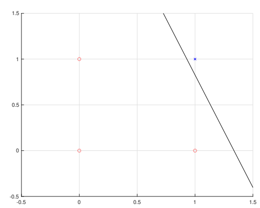
(4) NAND:

$$v = \begin{bmatrix} 1.5 & -1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \qquad y = \begin{cases} 0 & if\ v < 0 \\ 1 & if\ v \geq 0 \end{cases}$$

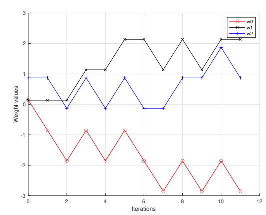
b)
(1)Comparision with a)



(a) off-line caculation

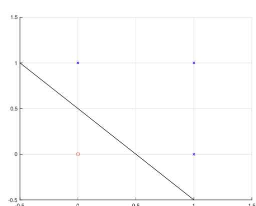


(b) learning result

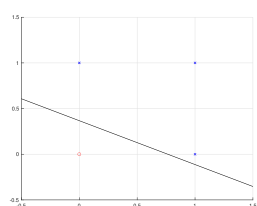


(c) weights trajectories

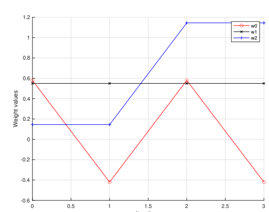
Figure 1: AND gate



(a) off-line caculation

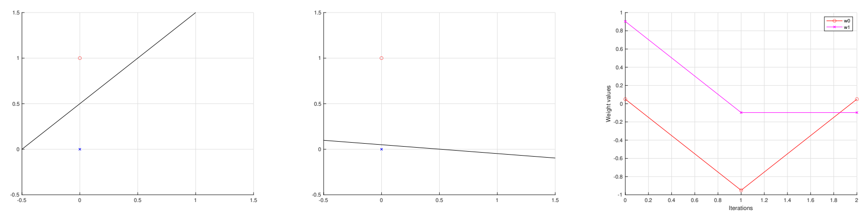


(b) learning result



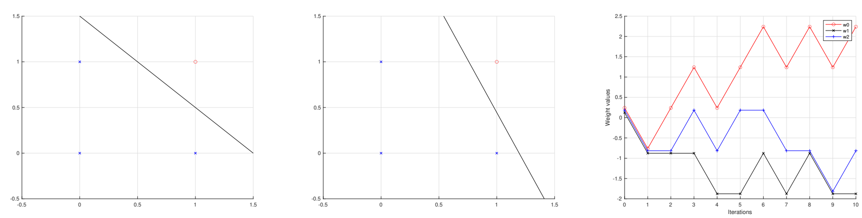
(c) weights trajectories

Figure 2: OR gate



(a) off-line caculation (b) learning result (c) weights trajectories

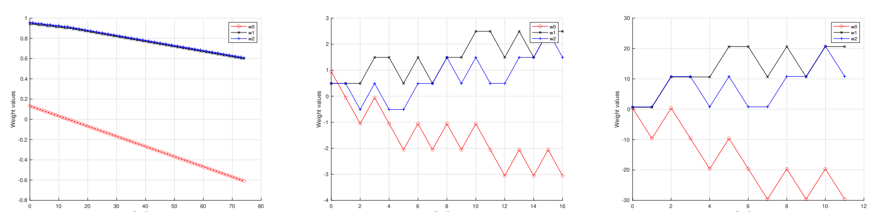
Figure 3: COMPLEMENT gate



(a) off-line caculation (b) learning result (c) weights trajectories

Figure 4: NAND gate

(2)Comparision of different learning rate



(a) $\eta = 0.01$ (b) $\eta = 1$ (c) $\eta = 100$

Figure 5: AND gate

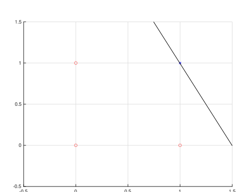
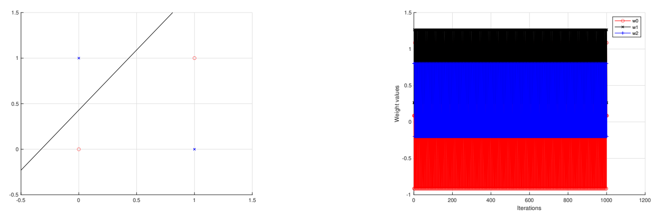


Figure 6: the decision line of $\eta = 100$

A learning rate that is too large can cause the model to converge too quickly to a suboptimal solution, as is shown in Fig.6. However, a learning rate that is too small can cause the process to get stuck and converge too slow as is shown in Fig.5 (a). Learning rate should be chosen according to the situation.

c)



(a) learning result (b) weights trajectories

Figure 7: XOR gate

The program is stuck into endless loop, which means the perceptron cannot find the solution for EXCLUSIVE OR. Meanwhile, the value of weights keeps jumping from one side to another.

0.4 Question 4

a)

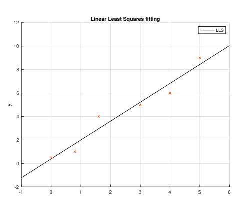
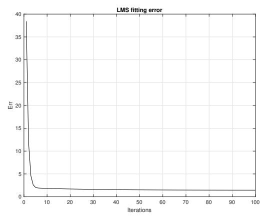
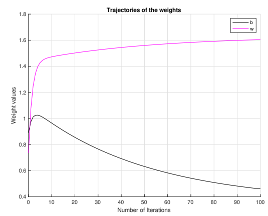


Figure 8: the fitting line of LLS

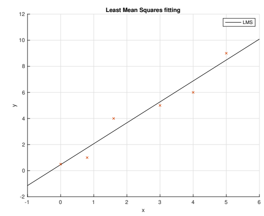
b)



(a) total error



(b) weights trajectories



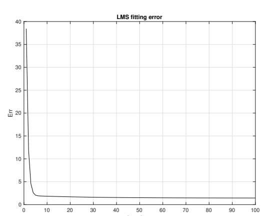
(c) fitting line

Figure 9: LMS

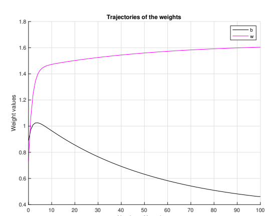
Obviously, the weights of the perceptron converge finally. And the last value of weights is [0.390478854532736,1.62228385854321].

c)
LLS can always achieve the global minimum. However, LMS use gradient descend to approach the minimum solution, it cannot guarantee it is the global minimum. So sometimes, LMS is stuck by local minimum. However, LLS cannot solve the data with large value. Because the inverse operation is computation expensive, where LMS shows its advantage.

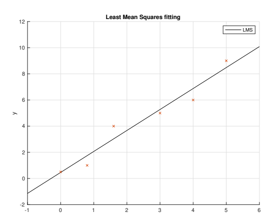
d)



(a) total error

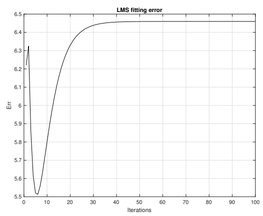


(b) weights trajectories

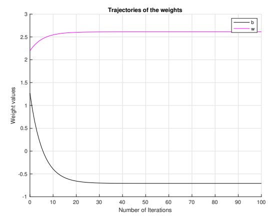


(c) fitting line

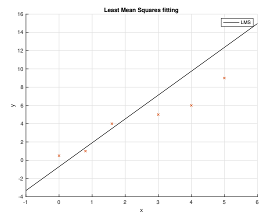
Figure 10: LMS with $\eta = 0.01$



(a) total error

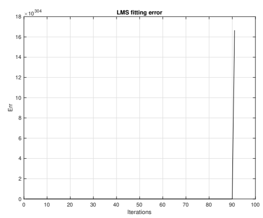


(b) weights trajectories

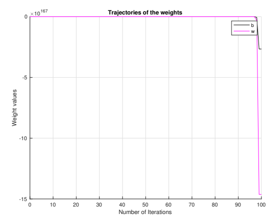


(c) fitting line

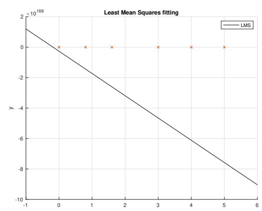
Figure 11: LMS with $\eta = 0.1$



(a) total error



(b) weights trajectories



(c) fitting line

Figure 12: LMS with $\eta = 0.5$

As the value of learning rate grows, the final error increases. If the learning rate is too large, the weights will not converge. Because, the learning process only concerns current point while forgets the learned points. Therefore, learning rate should be chosen carefully to keep a good balance of speed and performance.

0.5 Question 5

Define diagnal matrix R as following:

$$R = diag(r(1), r(2), r(3), ..., (n))$$

Then we can get:

$$y = w^T X$$
$$e = d - y = d - w^T X$$
$$J = eRe^T$$

Since $\frac{\partial y}{\partial w} = X$, $\frac{\partial e}{\partial w} = -X$, $\frac{\partial J}{\partial e} = 2eR$, we can obtain:

$$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial e}\frac{\partial e}{\partial w} = -2XRe^T$$

Therefore,

$$w* = w + \eta XRe^T$$

where η is learning rate.

```
%% Q3
% ground truth
AND = [ 0 0 1 1 ; 0 1 0 1 ; 0 0 0 1];
OR = [ 0 0 1 1 ; 0 1 0 1 ; 0 1 1 1];
COMPLEMENT = [ 0 1 ; 1 0];
NAND = [ 0 0 1 1 ; 0 1 0 1 ; 1 1 1 0];
XOR = [ 0 1 0 1 ; 0 0 1 1 ; 0 1 1 0];

% learning parameter setting
gate = XOR;           %the logic gate
rate = 1;           %learning rate
[dim, num_input] = size(gate);
loop = 1;
error = zeros(1,num_input);

%% off-line calculation
% w_and = [-1.5,1,1];
% w_or = [-0.5,1,1];
% w_complement = [0.5,1];
% w_nand = [1.5,-1,-1];
% w = w_and;
%
% figure;
% hold on;
% axis([-0.5,1.5,-0.5,1.5])
% for i = 1:num_input
%     if gate(end,i) == 1
%         plot(gate(1,i),gate(2,i),'bx');
%     else
%         plot(gate(1,i),gate(2,i),'ro');
%     end
% end
% x = linspace(-1,2,100);
% k = -w(end,2)/w(end,3);
% b = -w(end,1)/w(end,3);
% y = k * x + b;
% plot(x, y,'k')
% grid on
% hold off

% learning operation
w = rand(1,dim);
while true
for i = 1 : num_input
y = (w(loop,:) * [1;gate(1:dim-1 , i)]) > 0;
error(1,i) = gate(dim,i) - y;
if error(1,i) ~= 0
w(loop+1,:) = w(loop,:) + (rate*error(1,i)*[1;gate(1:dim-1 , ...
    i)])';
loop = loop + 1;
end
end
if all(error == 0)
break
elseif loop > 1000
break
end
```

```
    end


    % plot
    if dim == 2
    figure;
    hold on;
    xlabel("Iterations");
    ylabel("Weight values");
    x = 0:size(w,1)-1;
    plot(x,w(:,1),'-ro');
    plot(x,w(:,2),'-mx');
    legend({'w0','w1'});
    grid on
    hold off

    figure;
    hold on;
    axis([-0.5,1.5,-0.5,1.5])
    for i = 1:num_input
    if gate(end,i) == 1
    plot(0,gate(1,i),'bx');
    else
    plot(0,gate(1,i),'ro');
    end
    end
    x = linspace(-1,2,100);
    k = w(end,2);
    b = w(end,1);
    y = k * x + b;
    plot(x, y,'k')
    grid on
    hold off
    end

    if dim == 3
    figure;
    hold on;
    xlabel("Iterations");
    ylabel("Weight values");
    x = 0:size(w,1)-1;
    plot(x,w(:,1),'-ro');
    plot(x,w(:,2),'-kx');
    plot(x,w(:,3),'-b+');
    legend({'w0','w1','w2'});
    grid on
    hold off

    figure;
    hold on;
    axis([-0.5,1.5,-0.5,1.5])
    for i = 1:num_input
    if gate(end,i) == 1
    plot(gate(1,i),gate(2,i),'bx');
    else
    plot(gate(1,i),gate(2,i),'ro');
    end
```

```
end
x = linspace(-1,2,100);
k = -w(end,2)/w(end,3);
b = -w(end,1)/w(end,3);
y = k * x + b;
plot(x, y,'k')
grid on
hold off
end
```

```
%% Q4a
clc;
clear all;
close all;

points = [0,0.5;0.8,1;1.6,4;3,5;4,6;5,9];
x = points(:,1);
y = points(:,2);
X = [ones(6,1),x];
w= (inv(X'*X)*X'*y)';

k = w(1,2);
b = w(1,1);

a = linspace(-1,6,100);
y = k * a + b;

hold on
plot(a, y,'k')
scatter(points(:,1), points(:,2),'x');
legend("LLS");
xlabel('x')
ylabel('y')
title("Linear Least Squares fitting")
grid;
hold off

%% Q4b
clear all;
close all;

points = [0,0.5;0.8,1;1.6,4;3,5;4,6;5,9];
x = points(:,1);
y = points(:,2);
X = [ones(6,1),x];
num_input = length(points);
weights = rand(1,2);            % initial weight is chosen ...
    randomly
rate = 0.1;
error_sum = zeros(100,1);

for i = 1:100
for j = 1:6
error = y(j) - weights(i,:)*X(j,:)';
error_sum(i,1) = error^2/2 + error_sum(i,1);
weights(i,:) = weights(i,:) + rate*error*X(j,:);
```

```
    end
    weights(i+1,:) = weights(i,:);
    end

    figure
    plot(1:100,error_sum,'k');
    xlabel('Iterations')
    ylabel('Err')
    title("LMS fitting error")
    grid on;

    figure
    hold on
    plot(0:100, weights(:,1), 'k');
    plot(0:100, weights(:,2), 'm');
    legend("b", "w");
    xlabel('Number of Iterations')
    ylabel('Weight values')
    title("Trajectories of the weights")
    hold off
    grid on;

    k = weights(end,2);
    b = weights(end,1);
    a = linspace(-1,6,100);
    y = k * a + b;
    figure
    hold on
    plot(a, y,'k')
    scatter(points(:,1), points(:,2),'x');
    legend("LMS");
    xlabel('x')
    ylabel('y')
    title("Least Mean Squares fitting")
    grid;
    hold off
```