NUS

National University of Singapore

Name :	YU SHIXIN
Matric. No. :	A0195017E
NUSNET:	E0383682
Subject:	NEURAL NETWORKS
Assignment :	HOMEWORK ONE
Date :	8, Feb, 2019

Solution 1

According to the question.1. that v_k is the induced local field, so it can be written as:

$$v_k = \sum_{i=1}^{m} w_{ki} x_i + b_k \quad (1)$$

And the condition of the classification decision is that :

The observation vector

$$x = [x_1, x_2, x_3, \dots, x_m]^T \quad (2)$$

The weight vector

$$w = [w_{k1}, w_{k2}, w_{k3}, \dots, w_{km}] \quad (3)$$

Thus, combine (1)(2)(3), we can get this:

$$v(n) = \sum_{i=1}^{m} w_i(n) x_i(n) + b(n) = w^T(n) x(n) + b_k \quad (4)$$

So the output:

$$y = \varphi(v) > \xi \rightarrow \text{x belongs to class1} \quad (5)$$

$$y = \varphi(v) < \xi \rightarrow \text{x belongs to class2} \quad (6)$$

The key to estimate whether the following three functions can be chosen as the activation function is that function satisfies following hyper-plane equations:

$$v(n) = w^T(n) x(n) = 0 \quad (7)$$

$$x_1 w_1 + x_2 w_2 + \cdots + x_m w_m + b = 0 \quad (8)$$

Next I will use MATLAB to help me to plot those functions in coordinate axis.

Func.1.logistic function:$\varphi(v) = \frac{1}{1+e^{-v}}$

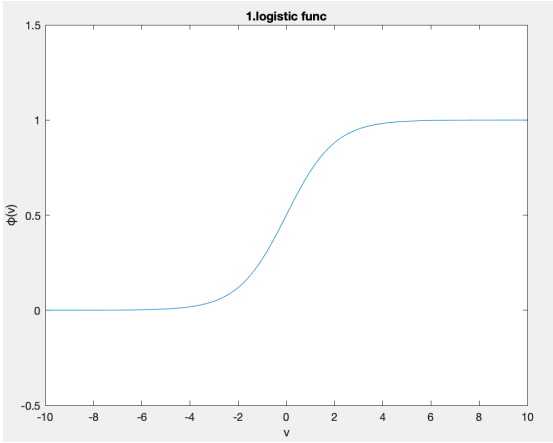


Fig.1.logistic function

Assuming that func.1. meets equation (7) and (8), therefore :

$$y = \varphi(v) = \frac{1}{1 + e^{-v}} = \xi$$

The decision boundary is：

$$v = ln\left(\frac{\xi}{1 - \xi}\right)$$

$$v = x_1 w_1 + x_2 w_2 + \cdots + x_m w_m + b$$

Assuming $q = v - b = x_1 w_1 + x_2 w_2 + \cdots + x_m w_m = ln\left(\frac{\xi}{1-\xi}\right) - b$, because q is a constant, it make $\varphi(v) = \frac{1}{1+e^{-v}} = \xi$.

Thus, according to equation (5), while $\sum_{i=1}^{m} w_i x_i > v - b$,the x belongs to C1(resp. $\sum_{i=1}^{m} w_i x_i < v - b$, the x belongs to C2)

Func.2:Bell-shaped Gaussian function:$\varphi(v) = e^{\frac{-(v-m)^2}{2}}$

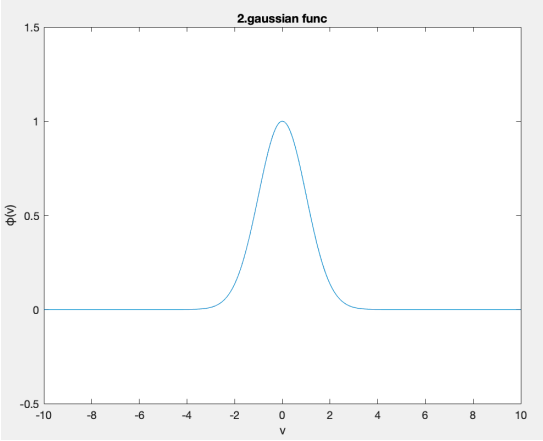


Fig.2.Bell-shaped Gaussian function

Similarly, the decision boundary in this function can be calculated as follow.

$$\varphi(v) = e^{\frac{-(v-m)^2}{2}} = \xi$$

So $v = m \pm \sqrt{2ln\left(\frac{1}{\xi}\right)}$ (m is a constant), when $0 < \xi < 1$, $q = v - b = m - b \pm \sqrt{2ln\left(\frac{1}{\xi}\right)}$ is a constant. However, in this case, it exists 2 hyper-plane.

Thus, Bell-shaped Gaussian function$\varphi(\mathrm{v}) = e^{\frac{-(v-m)^2}{2}}$ can not be chosen as activation function.

Func.3:Softsign function: $\varphi(\mathrm{v}) = \frac{v}{1+|v|}$

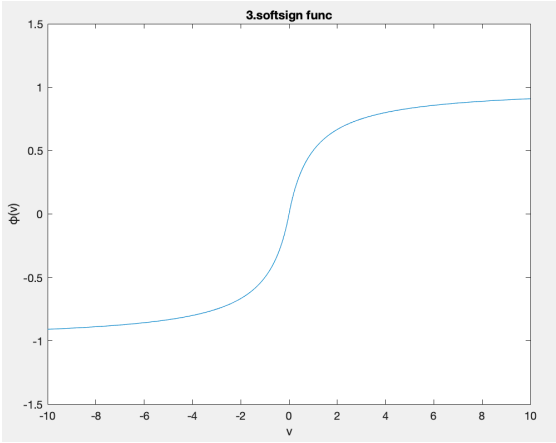


Fig.3.Softsign function

Also, the decision boundary in this function can be calculated as follow.

$$\varphi(\mathrm{v}) = \frac{v}{1+|v|} = \xi$$

so $v = \begin{cases} \frac{\xi}{1-\xi}, & v > 0 \\ \frac{\xi}{1+\xi}, & v < 0 \end{cases}$, $q = v - b = \frac{\xi}{1\pm\xi} - b$ is a constant which makes $\varphi(\mathrm{v}) = \xi$.

Thus, according to equation (5), while $\sum_{i=1}^{m} w_i x_i > \frac{\xi}{1\pm\xi} - b$,the x belongs to C1(resp. $\sum_{i=1}^{m} w_i x_i < \frac{\xi}{1\pm\xi} - b$, the x belongs to C2)

In conclusion, function 1&3 can be chosen as the activation functiowith, but function 2 can not due to its two hyper-plane.

Solution 2

It is obvious that XOR problem is linearly inseparable, but it is hard to proof directly. So this time I choose proofs by contradiction to proof this problem.

Suppose that XOR is linearly separable, so it exists a hyper-plane to meet those condition:

$$v(n) = w^T(n)x(n) + b \quad (1)$$

$$(x_1, x_2, y) \in [(1,1,0), (0,0,0), (1,0,1), (0,1,1)] \quad (2)$$

We can get four inequation from (1) and (2), that:

$$\begin{cases} w_1 + b > 0. & (3) \\ w_2 + b > 0 & (4) \\ w_1 + w_2 + b < 0 & (5) \\ b < 0 & (6) \end{cases}$$

from (3) (4) and (6), it means:

$$\begin{cases} b < 0 \\ w_1 > 0, |w_1| > \mathrm{b} \\ w_2 > 0, |w_2| > \mathrm{b} \end{cases}$$

But from (5), we know it is contradictory, so it do not exist a hyper-plane to meet this condition.

To sum up, XOR problem is not linearly separable.

Solution 3

a)I use MATLAB to help me to demonstrate the implementation of the logic functions AND, OR, COMPLEMENT, NAND with selection of weights by off-line calculations.

1. AND

Truth table of AND

x_1	0	0	1	1
x_2	0	1	0	1
y	0	0	0	1

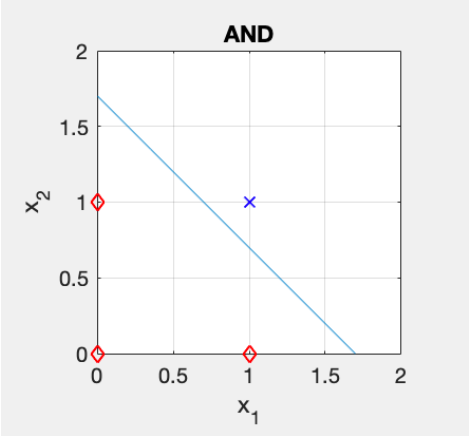


Fig.1.AND function

$$v = wx^T = [-1.7,1,1]\begin{pmatrix}1\\x_1\\x_2\end{pmatrix}$$

$$y = \begin{cases}0; if\ v < 0\\1; if\ v \geq 0\end{cases}$$

$$l(x_1 + x_2 - 1.7)$$

2. OR

Truth table of OR

x_1	0	0	1	1
x_2	0	1	0	1
y	0	1	1	1

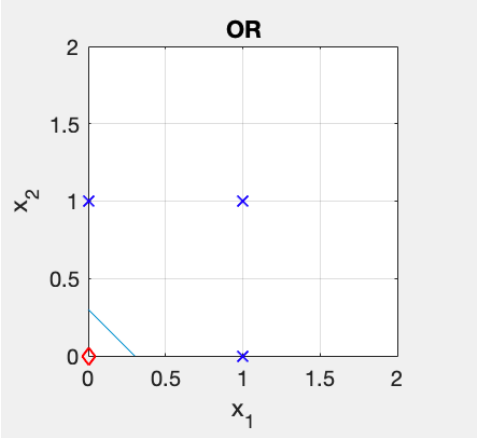


Fig.2.OR function

$$v = wx^T = [-0.3,1,1]\begin{pmatrix}1\\x_1\\x_2\end{pmatrix}$$

$$y = \begin{cases}0; if\ v < 0\\1; if\ v \geq 0\end{cases}$$

$$l(x_1 + x_2 - 0.3)$$

3. COMPLEMENT

Truth table of COMPLEMENT

x	0	1
y	1	0

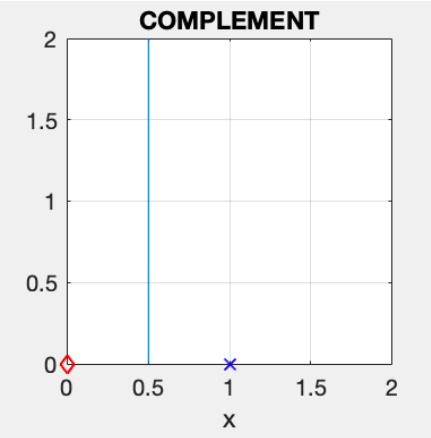


Fig.2.COMPLEMENT function

$$v = wx^T = [-0.5,1]\begin{pmatrix}1\\x\end{pmatrix}$$

$$l(x - 0.5)$$

4. NAND

Truth table of NAND

x_1	0	0	1	1
x_2	0	1	0	1
y	1	1	1	0

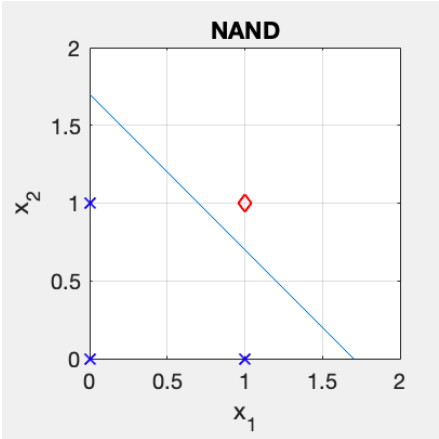


Fig.4.NAND function

$$v = wx^T = [1.7,-1,-1]\begin{pmatrix}1\\x_1\\x_2\end{pmatrix}$$

$$y = \begin{cases} 0; if\ v < 0 \\ 1; if\ v \geq 0 \end{cases}$$

$$l\ (-x_1 - x_2 + 1.7)$$

b)According to the Q3-b, the iterative formula of the perceptron learning algorithm can be written as below:

$$w(n+1) = w(n) + \eta e(n)x(n)(\eta > 0) \quad (1)$$

$$e(n) = d(n) - y(n) \quad (2)$$

I will use MATLAB to implement the learning procedure, and the core code shown as below.

Code.1 learning procedure code

```
%para
eta=1;
t=0;
i=1;
max_loop=50;
w0=zeros(1,max_loop);
w1=zeros(1,max_loop);
w2=zeros(1,max_loop);

%learning procedure
while (t<max_loop)
    w0(i)=w(1);
    w1(i)=w(2);
    w2(i)=w(3);
    i=i+1;

    v=w*x;
    y=hardlim(v);
    e=d-y;
    w=w+eta*e*x';
    t=t+1;
end
```

1. AND

Input for AND

$$X = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} d = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad w_{r0} = [0.04265 \quad 0.6352 \quad 0.2819]$$

The trajectory of weights in this case is plotted in Fig.5， and the obtained decision boundary is plotted in Fig.6.

$$l(-1.957+1.635x_1 + 1.282x_2)$$

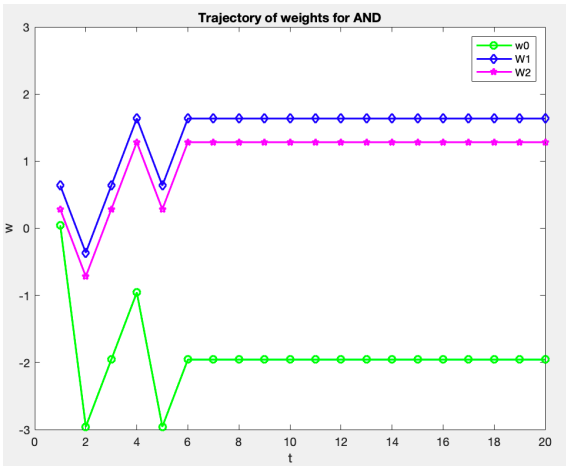


Fig.5.Trajectory of weights of AND

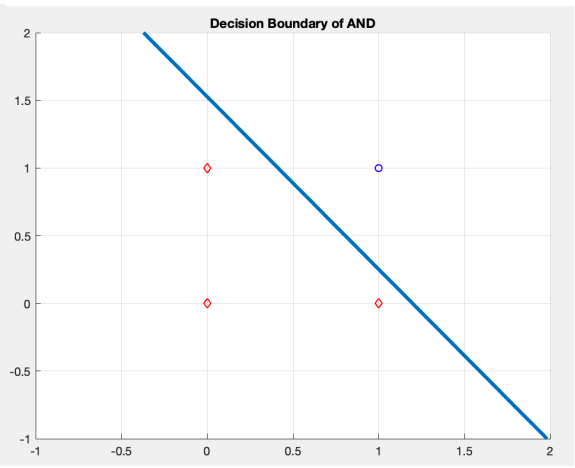


Fig.6.Decision boundary of AND

2. OR

Input for OR

$$X = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} d = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad w_{r0} = [0.5358 \quad 0.4452 \quad 0.1239]$$

The trajectory of weights in this case is plotted in Fig.7，and the obtained decision boundary is plotted in Fig.8.

$$l(-0.4642 + 1.445x_1 + 1.124x_2)$$

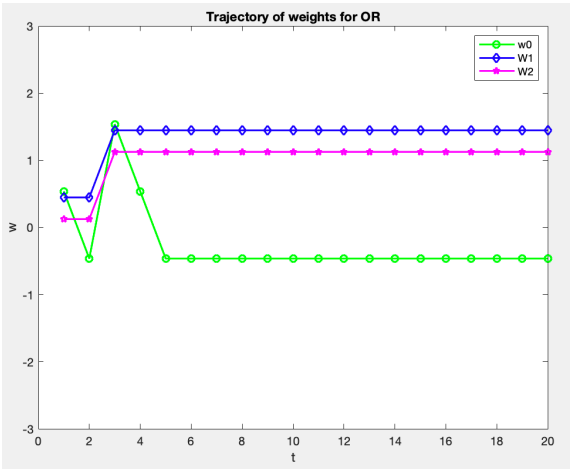


Fig.7.Trajectory of weights of OR

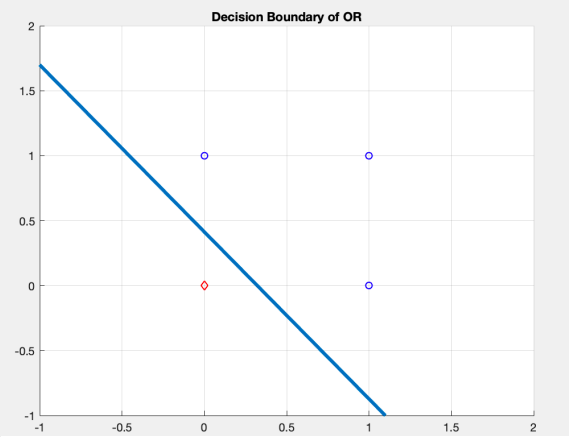


Fig.8.Decision boundary of OR

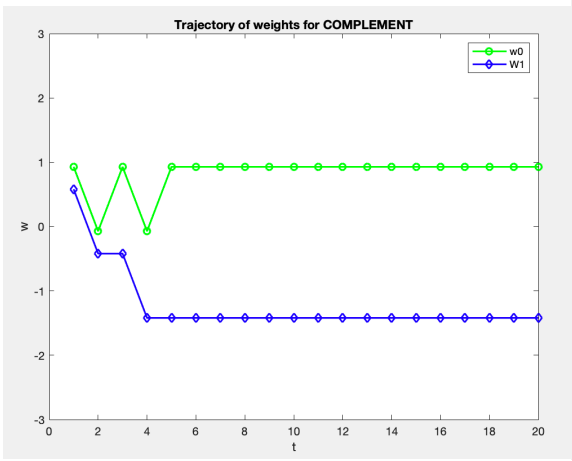
3. COMPLEMENT

Input for COMPLEMENT

$$X = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} d = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad w_{r0} = [0.6403 \quad 0.417]$$

The trajectory of weights in this case is plotted in Fig.9，and the obtained decision boundary is plotted in Fig.10.

$$l(0.6403 - 1.583x_1)$$



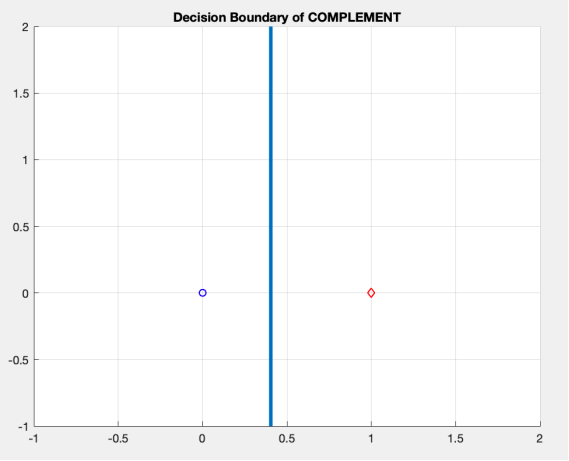


Fig.9.Trajectory of weights of COMPLEMENT Fig.10.Decision boundary of COMPLEMENT

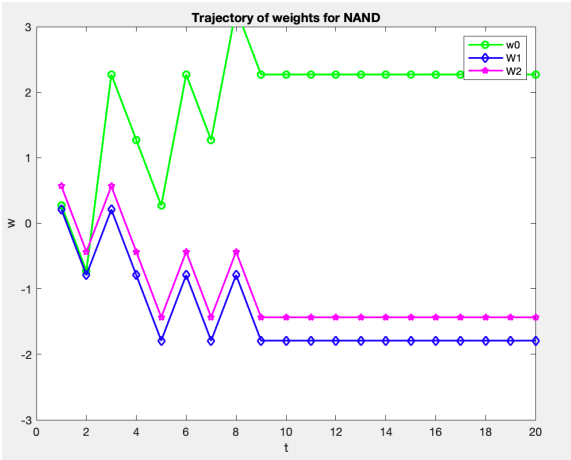
4. NAND

Input for NAND

$$X = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} d = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad w_{r0} = [0.2703 \quad 0.2085 \quad 0.565]$$

The trajectory of weights in this case is plotted in Fig.11，and the obtained decision boundary is plotted in Fig.12.

$$l(2.27 - 1.792x_1 - 1.435x_2)$$



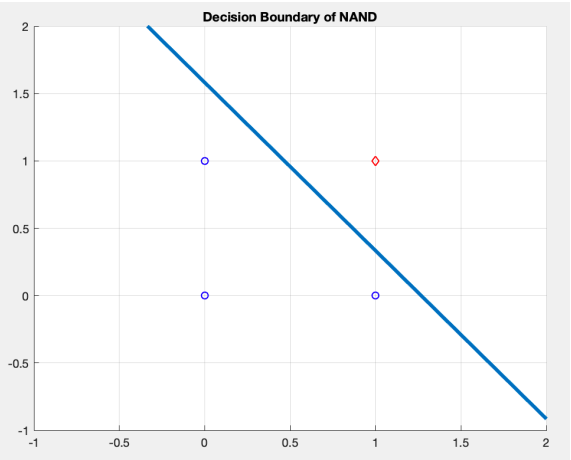


Fig.11.Trajectory of weights of NAND Fig.12.Decision boundary of NAND

Comparing Fig. 6, Fig. 8, Fig. 10, Fig. 12 with Fig.1-4, it can be seen clearly that although the coefficients between off-line calculations and learning procedure are different, they follow the same formula, which they can separate the class correctly.

Using different learning rate η to show this parameter' effect in learning procedure.(i.e. NAND). We can find that with the increase of the learning rate η, the weight vector converges become faster. However, it seems useless if we increase learning rate η infinitely. So for each model, it exists one best learning rate η, which is essential for training.

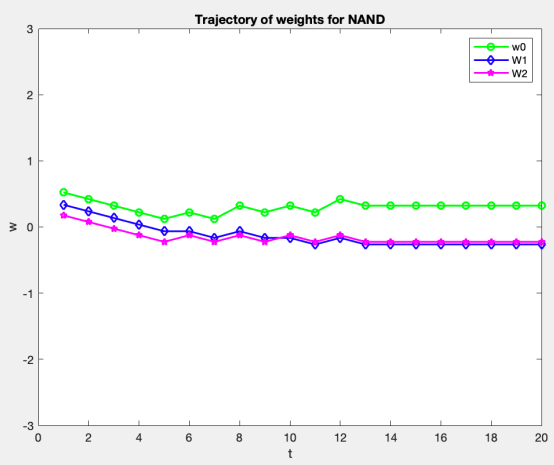


Fig.13. η=0.1

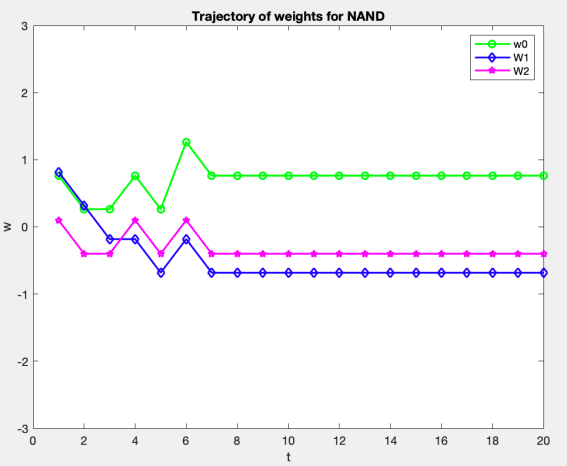


Fig.14. η=0.5

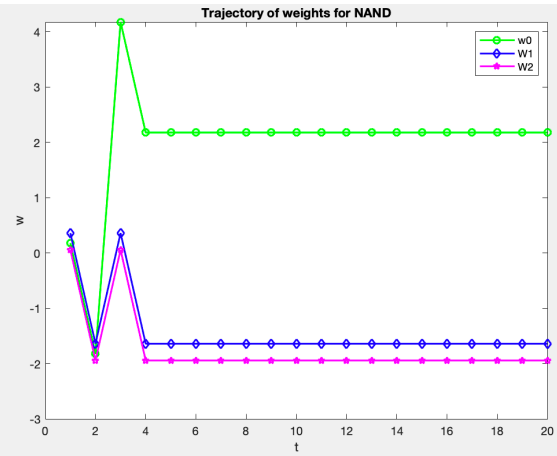


Fig.15. η=2

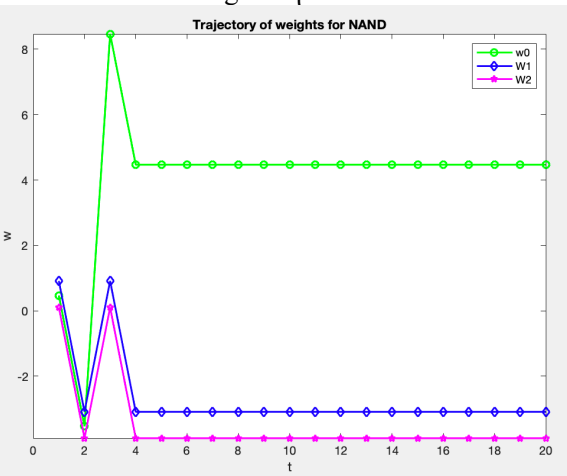


Fig.16. η=4

c)When I apply the perceptron to implement the XOR function with selection weights by learning procedure, I find that the weight can not converge (Fig.17) and no correct decision boundary can be obtained(Fig.18). This is because XOR function is not linearly separable, which be proved in Q2.

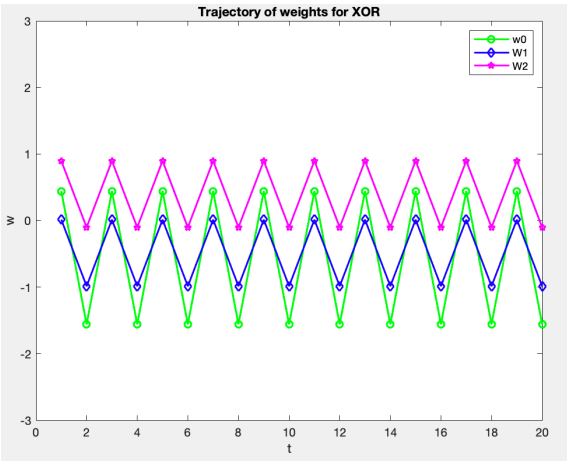


Fig.17.Trajectory of weights of XOR

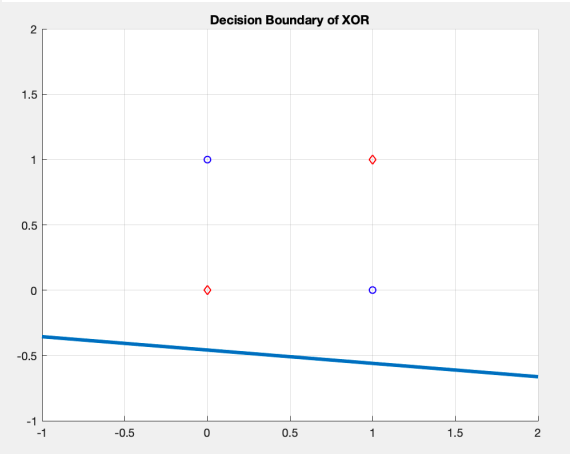


Fig.18.Decision boundary of XOR

Solution 4

a)First , we can get the cost function is:

$$E = \sum_{i=1}^{n} e(i)^2 = e^T e \tag{1}$$

And the

$$e(i) = d(i) - y(i) \rightarrow e = d - y \tag{2}$$

$$y = [y(1), y(2), \dots, y(n)]^T \tag{3}$$

$$d = [d(1), d(2), \dots, d(n)]^T \tag{4}$$

Because $y(i) = w^T x(i) = x(i)^T w \rightarrow y = Xw$ (5)

So $e = d - Xw$ (6)

By chain rule, we get that

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial e}\frac{\partial e}{\partial w} \tag{7}$$

According to (1) and (6)

$$\frac{\partial E}{\partial e} = 2e^T$$

$$\frac{\partial e}{\partial w} = -X$$

Thus,

$$\frac{\partial E}{\partial w} = -2e^T X$$

Solve $\nabla_w E = 0$, it gives $w = (X^T X)^{-1} X^T d$

From the given pairs, we can get:

$$X = \begin{bmatrix} 1 & 0 \\ 1 & 0.8 \\ 1 & 1.6 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{bmatrix} \qquad d = \begin{bmatrix} 0.5 \\ 1 \\ 4 \\ 5 \\ 6 \\ 8 \end{bmatrix}$$

From equation (7), we can calculate the weight vector.

$$\begin{bmatrix} b \\ w \end{bmatrix} = (X^T X)^{-1} X^T d = \begin{bmatrix} 0.5554 \\ 1.4700 \end{bmatrix}$$

In conclusion, the result of b is 0.5554 and w is 1.4700.

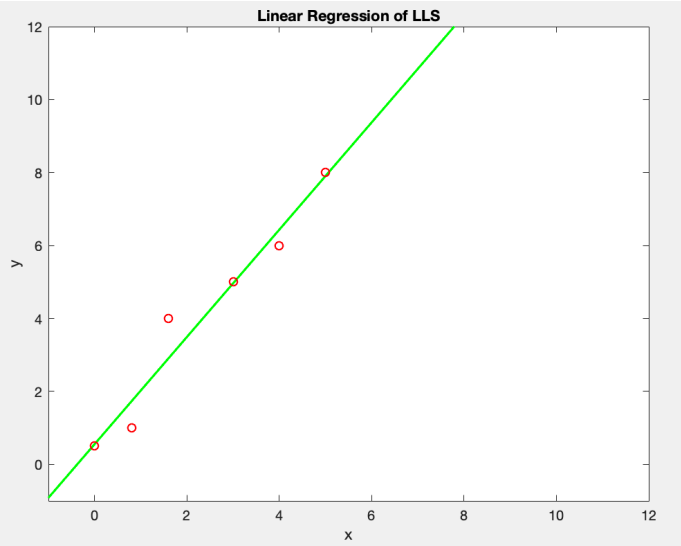


Fig.1. Linear Regression of LLS

b)According to the description of b, we need set learning rate $\eta = 0.01$, $epochs = 100$. In this case, the random weight vector $w = \begin{bmatrix} 0.7593 \\ 0.7406 \end{bmatrix}$, and after 100 epochs, the $w_{100} = \begin{bmatrix} 0.6092 \\ 1.455 \end{bmatrix}$. Thus, the result of w is 1.455 and b is 0.6092. The fitting result is shown in Fig.4, the trajectories of weights versus learning step is shown in Fig.2, and the average error versus epochs is shown in Fig.3.

Code.2 LMS algorithm

```
%para
eta=0.1;
i=1;
t=0;
epoch=100;
```

```
w0=zeros(1,epoch);
w1=zeros(1,epoch);
e1=zeros(1,epoch);

%input
x=[1,0;1,0.8;1,1.6;1,3;1,4;1,5];
d=[0.5;1;4;5;6;8];
w=rand(1,2);
a=x'*x;
b=a^-1;
c=b*x'*d;

%learning procedure
while (t<epoch)
    w0(i)=w(1);
    w1(i)=w(2);

    y=w*x';
    e=d-y';
    e1(i)=(e'*e)/2;
    w=w+eta*e'*x;

    t=t+1;
    i=i+1;
end

%fitting result LLS and LMS
k1=c(2);
j1=c(1);
figure(3);
x=-1:10;
y1=k1.*x+j1;
k2=w(2);
j2=w(1);
y2=k2.*x+j2;
plot(x,y1,'color','r','LineWidth',1);
hold on;
plot(x,y2,'color','k','LineWidth',1);
hold on;
scatter([0,0.8,1.6,3,4,5],[0.5,1,4,5,6,8],'r','o');
axis([-1,12,-1,12]);
title('Linear Regression of LLS and LMS');
xlabel('x');
ylabel('y');
```

From this figures, we can find that the weights and average error show a converging tendency after several learning steps and epochs. However, the weights can not converge to a certain value in 100 epochs. We can do a reasonable guess that the weight will not converge, because those given points do not lie on a straight line, which exists infinite linear regressions.

Thus, the error will exist and the weights can not converge to a certain value.

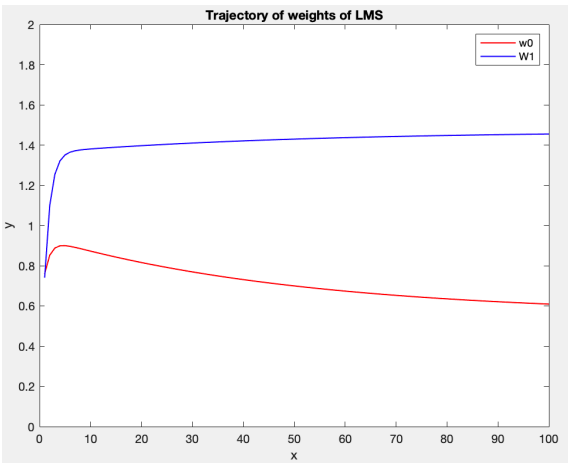


Fig.2.Trajectory of weights of LMS

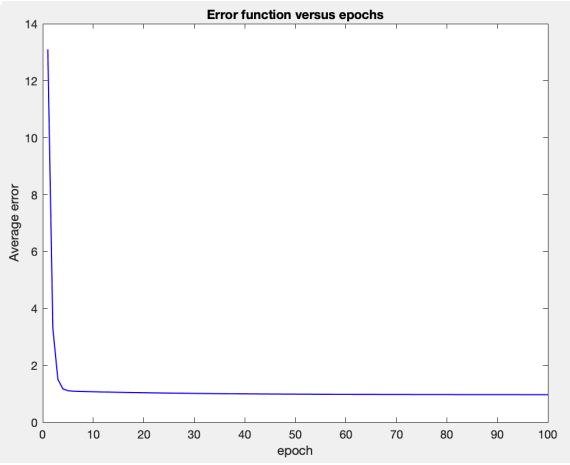


Fig.3.Error function versus epochs

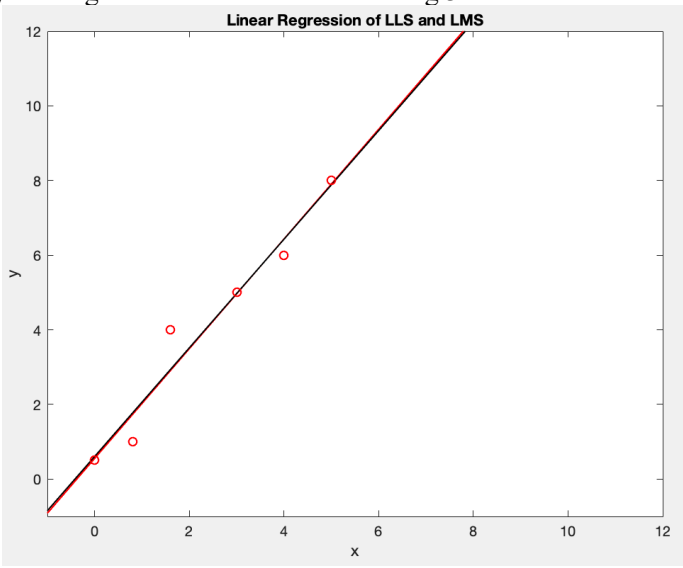


Fig.4.Linear Regression of LMS

c)We can see clearly from Fig.4 that these two fitting results(both LLS and LMS) are very similar. Thus, for small data, we can use both algorithms, but for big data, it is better to apply LMS. Although LMS requires a long training procedure to converge, the LLS will need high computation.

d)Using different learning rate η to show this parameter' effect in training procedure. We can find that with the increase of the learning rate η, the weight vector diverges and shows a different tendency with the figure(η=0.01). We can infer that LMS algorithm only valid when the learning rate η is reasonably small.

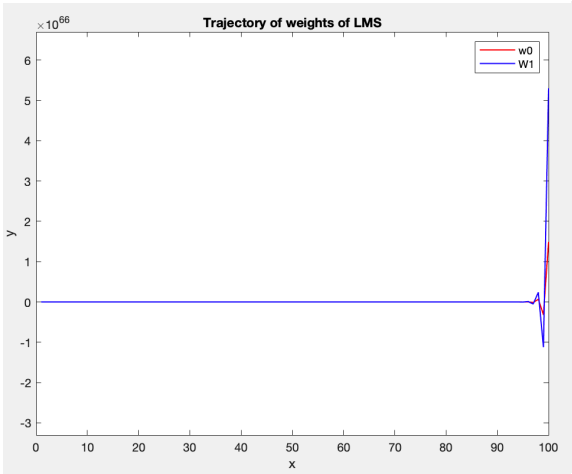


Fig.5.Trajectory of weights of LMS η=0.1

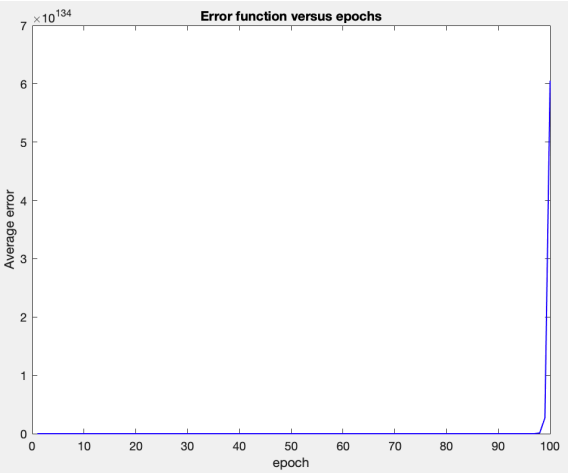


Fig.6.Error function versus epochs. η=0.1

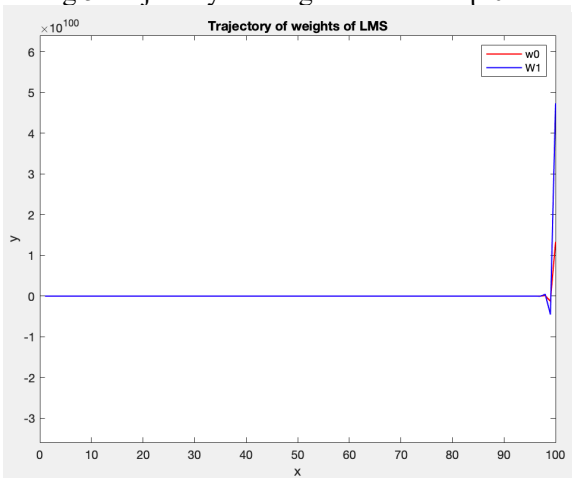


Fig.7.Trajectory of weights of LMS η=0.2

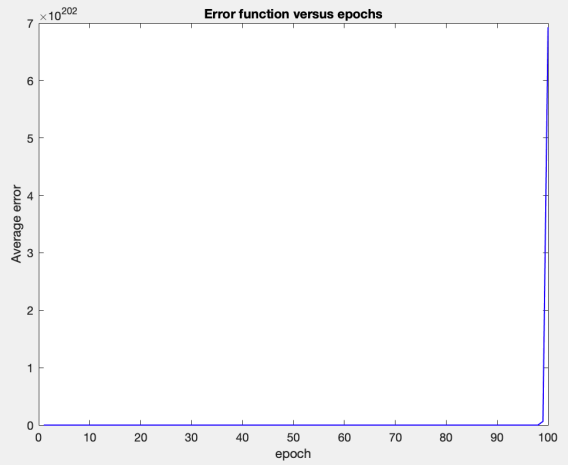


Fig.8.Error function versus epochs. η=0.2

Solution 5

According to the description of Q5, we can know that $r(i) > 0$ are the weighting factors for each output error $e(i)$, so we can define that

$$R_{n\times n} = diag\big(r(1), r(2), \dots, r(n)\big)$$

$$E_{n\times 1} = \begin{bmatrix} e(1) \\ e(2) \\ . \\ . \\ . \\ e(n) \end{bmatrix}$$

Thus, it can be written as $\sum_{i=1}^{n} r^2(i)e(i)^2 = E_{1\times n}^T \times (R^T R)_{n\times n} \times E_{n\times 1}$

And the cost function $J(w)$ can be simplified as below:

$$J(w) = \frac{1}{2} e^T R^T R e + \frac{1}{2} \lambda w^T w$$

Besides, error $e(i) = d(i) - y(i)$, $e = d - y = d - Xw$. So

$$\frac{\partial e}{\partial w} = -X$$

When $\frac{\partial J(w)}{\partial w} = 0$, we can calculate the optimal parameter w^*.

$$\frac{\partial J(w)}{\partial w} = e^T R^T R \frac{\partial e}{\partial w} + \lambda w^T = -e^T R^T R X + \lambda w^T = 0$$

$$(d - Xw)^T R^T R X = \lambda w^T$$

$$d^T R^T R X - w^T X^T R^T R X = \lambda w^T$$

Transposition on both sides of equation, simultaneously:

$$X^T R^T R d - X^T R^T R X w = \lambda w$$

$$(\lambda + X^T R^T R X)^{-1} X^T R^T R d = w$$

Thus, the $w^* = (\lambda + X^T R^T R X)^{-1} X^T R^T R d$