EE5904 Neural Networks: HW #1

A0206597U Xin Zhengfang

Q1:

The input vector:

$$x = [+1, x_1, x_2, \dots, x_m]^T$$

The weight vector:

$$w = [b, w_1, w_2, \dots, w_m]^T$$

The induced local field v

$$v = w^T x = \sum_{i=0}^{m} w_i x_i$$

1) The decision boundary:

$$\varphi(v) = \xi = av + b$$

$$a(b + w_1 x_1 + w_2 x_2 + \cdots + w_m x_m) + (b - \xi) = 0$$

Hyper-plane

2) The decision boundary:

$$\varphi(v) = \xi = \frac{1}{1 + e^{-2v}}$$

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

Define constant $C = -\frac{1}{2} ln \frac{1-\xi}{\xi}$, then

$$b + w_1 x_1 + w_2 x_2 + \cdots + w_m x_m - C = 0$$

Hyper-plane

3) The decision boundary:

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

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

Not hyper-plane

Q2:

We assume XOR is linearly separable, which means we can find a decision boundary:

$$w_1 x_1 + w_2 x_2 + b = 0$$

That can successfully separate points.

We also assume a threshold, which satisfy classification:

$$w_1x_1 + w_2x_2 + b \geq 0 \rightarrow y = 1$$

$$w_1x_1 + w_2x_2 + b < 0 \rightarrow y = 0$$

From the four data in the table we can get four inequation:

$$\begin{cases} b < 0 \\ w_1 + b \geq 0 \\ w_2 + b \geq 0 \\ w_1 + w_2 + b < 0 \end{cases}$$

Combine inequation 2.and 3. We can get:

$$\begin{cases} b < 0 \\ w_1 + w_2 + 2b \geq 0 \\ w_1 + w_2 + b < 0 \end{cases}$$

Combine inequation 1.and 3. We can get:

$$\begin{cases} w_1 + w_2 + 2b \geq 0 \\ w_1 + w_2 + 2b < 0 \end{cases}$$

No solution which means XOR is not linearly separable.

Q3:

a)

a. AND:

$$w_1 = 1, w_2 = 1, b = -1.5$$

$$v = x_1 + x_2 - 1.5 = 0$$

If $v \geq 0$, $y = 1, else\ y = 0$.

b. OR:

$$w_1 = 1, w_2 = 1, b = -0.5$$

$$v = x_1 + x_2 - 0.5 = 0$$

If $v \geq 0$, $y = 1, else\ y = 0$.

c. COMPLEMENT:

$$w_1 = -1, b = 0.5$$

$$v = -x_1 + 0.5 = 0$$

If $v \geq 0$, $y = 1, else\ y = 0$.

d. NAND:

$$w_1 = -1, w_2 = -1, b = 1.5$$

$$v = -x_1 - x_2 + 1.5 = 0$$

b) a.

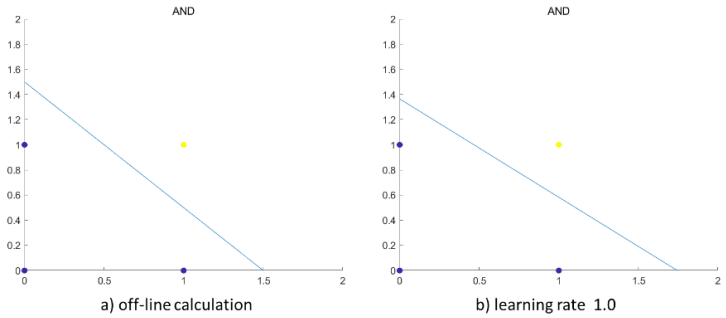


Figure 1. logic AND comparison

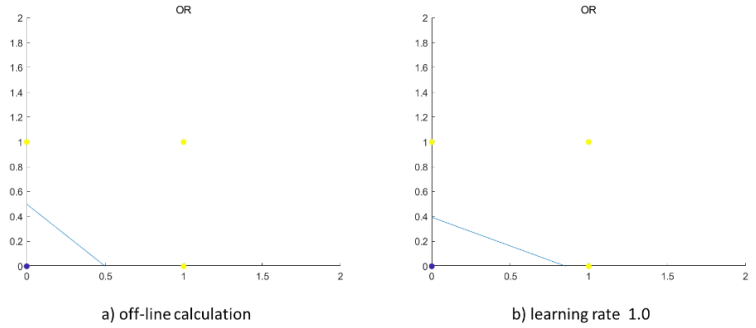


Figure 2. logic OR comparison

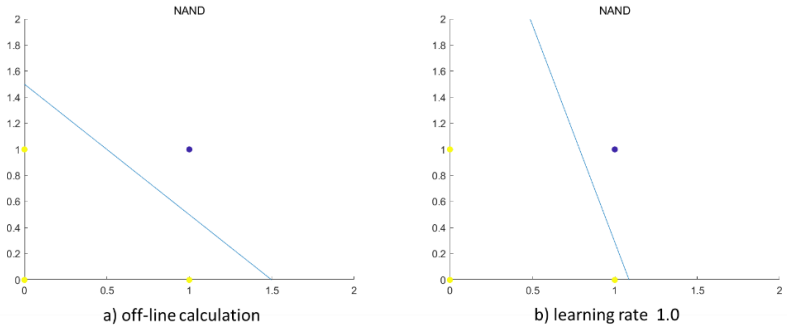


Figure 3. logic NAND comparison

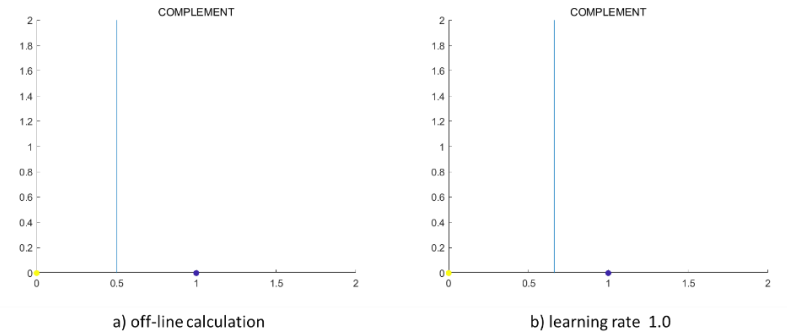
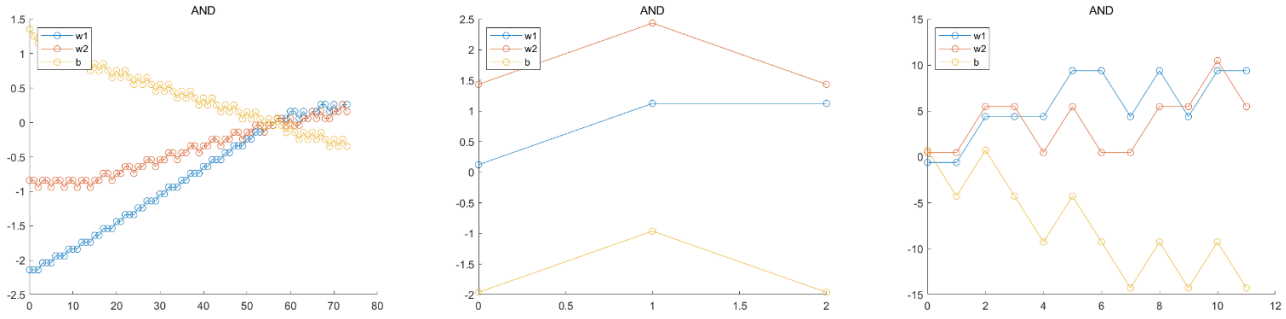


Figure 4. logic COMPLEMENT comparison

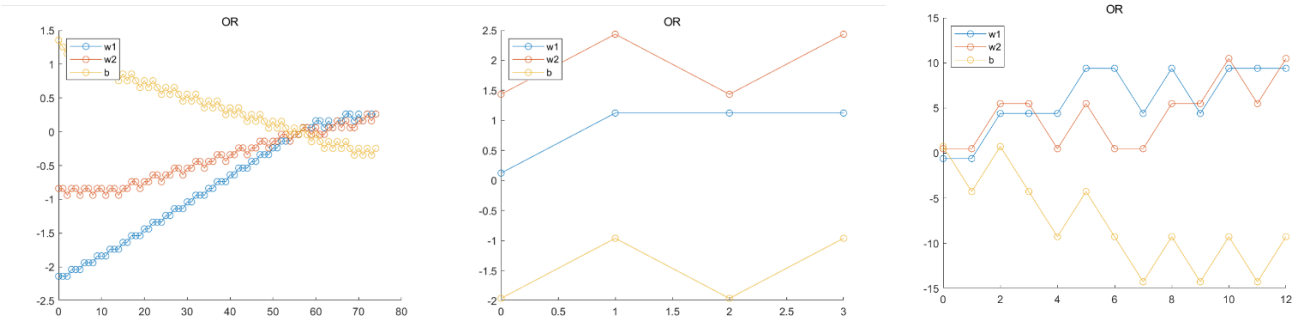
Q3.b.b



a) learning rate 0.1 b) learning rate 1.0 c) learning rate 5.0

Figure 5. logic AND comparison

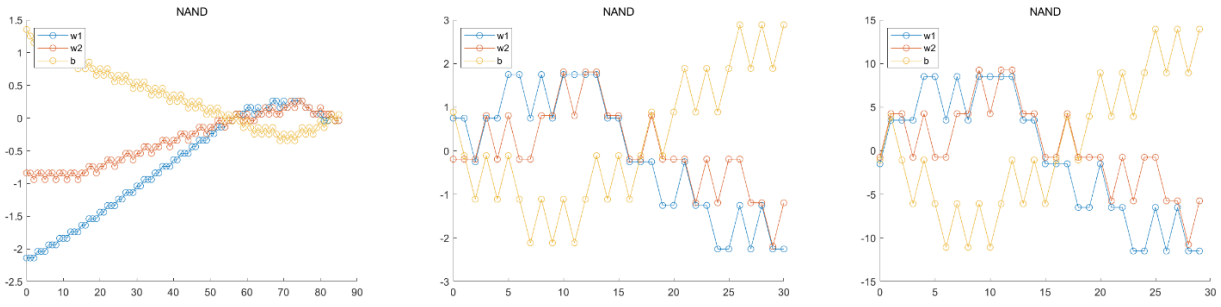
We can see the learning rate 1.0 is the best one in 0.1,1.0 and 5.0. Too small or too big, will increase the steps to converge.



a) learning rate 0.1 b) learning rate 1.0 c) learning rate 5.0

Figure 6. logic OR comparison

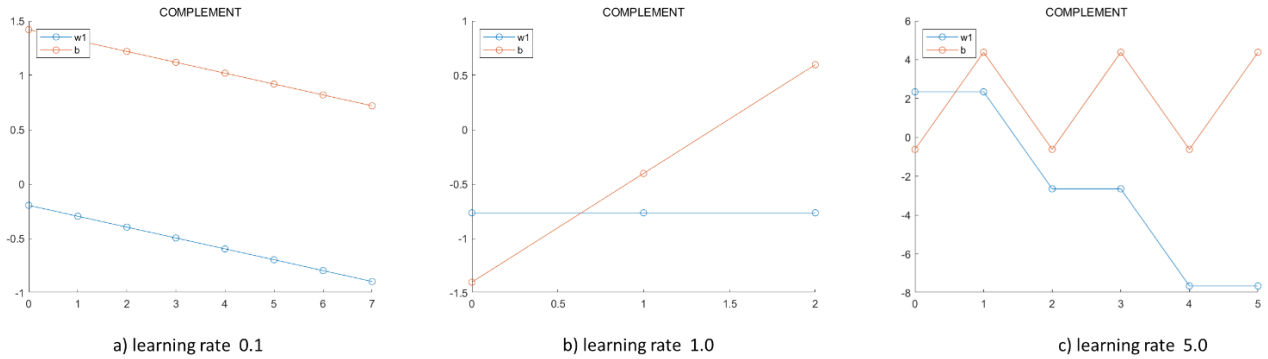
We can see the learning rate 1.0 is also the best one in 0.1,1.0 and 5.0. Too small or too big will increase the steps to converge.



a) learning rate 0.1 b) learning rate 1.0 c) learning rate 5.0

Figure 7. logic NAND comparison

However, the best learning rate may be 1.0 or 5.0 in 0.1,1.0 and 5.0. It is different from the logic AND and logic OR. Therefore, it really depends on the task when we choose the learning rate.



a) learning rate 0.1 b) learning rate 1.0 c) learning rate 5.0

Figure 8. logic COMPLEMENT comparison

We can see the learning rate 1.0 is the best one in 0.1,1.0 and 5.0. Too small or too big, will increase the steps to converge.

Above all, if it is chosen very large and applied to the example x(n), then learning is excellent as far as the present example is concerned, but at the cost of spoiling the learning that has taken place earlier with respect to other examples. Thus, a large value of learning rate is not necessarily good. If an extremely small value is chosen for, that also leads to slow learning. Some intermediate value is the best. Usually the choice is problem dependent.

Q3.c.

Matlab code:

```
EXCLUSIVE_OR = [0 0 1 0;0 1 1 1;1 0 1 1;1 1 1 0];

eta = 0.1;
w = randn(1,3);
w_h = w;

% EXCLUSIVE_OR
classified = false;

while not (classified)
    classified = true;
    for i = 1:length(EXCLUSIVE_OR(:,1))
        if EXCLUSIVE_OR(i,1:3)*w' >= 0 y = 1;else y = 0; end
        if EXCLUSIVE_OR(i,4) ~= y
            classified = false;
            w = w + eta*(EXCLUSIVE_OR(i,4)-y)*EXCLUSIVE_OR(i,1:3);
            w_h = [w_h;w];
        end
    end
end
```

The program went into endless loop, which means the perceptron cannot find the solution for EXCLUSIVE OR.

Q4:

Q4.a

Matlab code:

```
x = [0 1;0.8 1;1.6 1;3 1;4.0 1;5.0 1];
d = [0.5; 1; 4; 5; 6; 9];

w = (x'*x)^(-1)*x'*d;
y = x*w;

figure
hold on
scatter(x(:,1),d,'filled')
p = plot(x(:,1),y);
p.LineWidth  = 2;
```

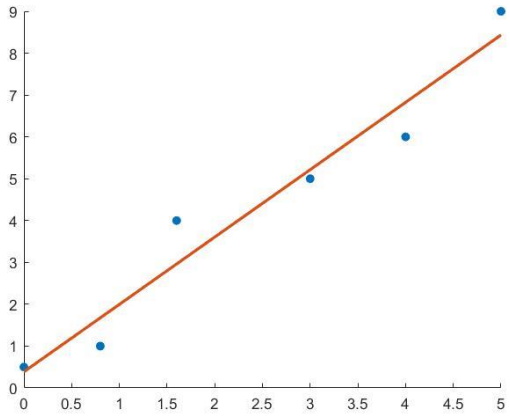


Figure 8. LLS fitting

Q4.b

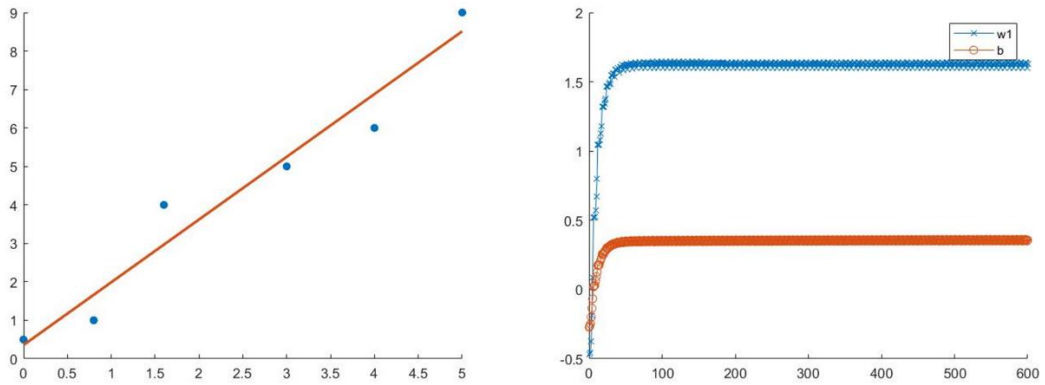


Figure 9. LMS fitting

Obviously, the weights of the perceptron converge finally. And the last value of weights is [1.71294827669770, 0.000147664733441000].

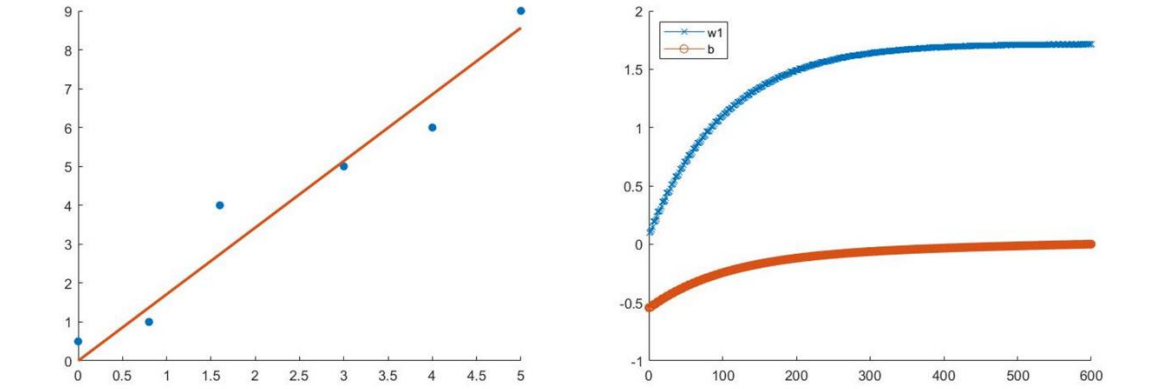
Matlab code:

```
x = [0 1;0.8 1;1.6 1;3 1;4.0 1;5.0 1];
d = [0.5; 1; 4; 5; 6; 9];

w = randn(1,2);
eta = 0.01; % learning rate
classified = false;


for epoch = 1:100
    classified = true;
    for i = 1:length(x(:,1))
         if x(i,1:2)*w' == d(i)
         else
             classified = false;
             e = d(i) - x(i,1:2)*w';
             w = w + eta*e*x(i,1:2);
         end
    end
end

y = x*w';

figure
hold on
scatter(x(:,1),d,'filled')
p = plot(x(:,1),y);
p.LineWidth  = 2;
```

Q4.c

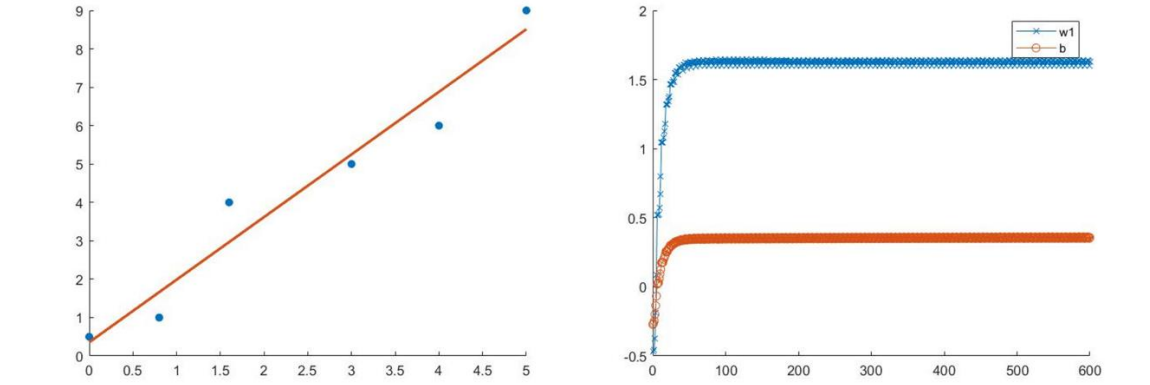
LLS can always get the global minimum loss. However, LMS use gradient descend to come near the minimum solution, it cannot guarantee the best solution. However, LLS cannot solve the big data, because the inverse operation will be computation expensive, where LMS shows the advantage.

Q4.d



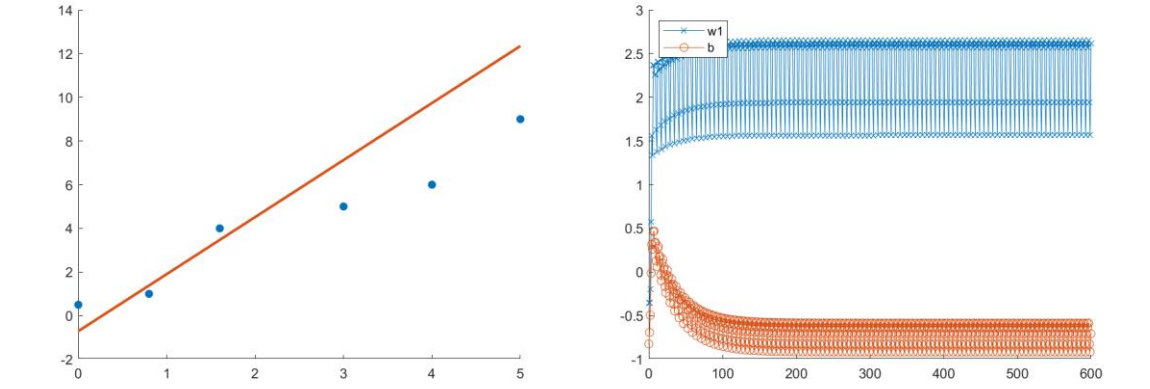
a) LMS fitting result 0.001 b) Weights of LMS fitting 0.001

Figure 9. LMS learning rate 0.001



a) LMS fitting result 0.01 b) Weights of LMS fitting 0.01

Figure 10. LMS learning rate 0.01



a) LMS fitting result 0.1 b) Weights of LMS fitting 0.1

Figure 11. LMS learning rate 0.1

From the figure 9. 10 and 11, the update process become more fluctuate when the learning rate increases. Large learning rate contribute the speed of convergence, however, if the learning rate is too large, the learning process as far as concern about current point while forgets the learned points before. Therefore, choosing learning rate carefully not only contribute to speed but also the performance.

Q5:

We define matrix R as follow:

$$R = diag(r(1), r(2), \dots, r(n))$$

Then we can rewrite the equations as follow:

$$y = w^T X$$

$$e = d - y$$

$$J = eRe^T$$

Derivate y by w, we can get $\frac{\partial y}{\partial w}$ as follow:

$$\frac{\partial y}{\partial w} = X$$

Derivate e by w, we can get $\frac{\partial e}{\partial w}$ as follow:

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

Derivate J by e, we can get $\frac{\partial J}{\partial e}$ as follow:

$$\frac{\partial J}{\partial e} = 2eR$$

By using chain rule, we can get $\frac{\partial J}{\partial w}$ as follow:

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

Therefore,

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

The constant η is the learning rate and a. 2 is absorbed by η .