EE 5904 Homework 1

February 9, 2020

Question1

According to the signal-flow graph of the perceptron, the induced local field v can be written as:

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

After feeding into the activation function, it can be classified as 0 or 1, which can be seen as below:

$$\phi = \begin{cases} C_1 & \text{if } \phi(c) \geq \xi \\ C_2 & \text{if } \phi(c) < \xi \end{cases}$$

Check the effects of these three activation functions on decision boundary.

1

Linear function $\phi(v) = av + d$
The decision boundary is when

$$\phi(v) = \xi$$
$$v = \frac{\xi - d}{a}$$
$$x_1 w_1 + x_2 w_2 + ... + x_m w_m = c$$

where $c = \frac{\xi - d}{a} - b$ is a constant. Thus, the resulting decision boundary is a hyper-plane.

2

logistic function $\phi(v) \frac{1}{1+e^{-2v}}$
The decision boundary is when

$$\phi(v) = \xi$$
$$v = \frac{1}{2} \ln \frac{\xi}{1 - \xi}$$
$$x_1 w_1 + x_2 w_2 + ... + x_m w_m = c$$

where $c = \frac{1}{2} \frac{\xi}{1-\xi} - b$ is a constant. Thus, the resulting decision boundary is a hyper-plane.

3

Bell-shaped Gaussian function $\phi(v)e^{-\frac{v^2}{2}}$
The decision boundary is when

$$\phi(v) = \xi$$

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

$$x_1w_1 + x_2w_2 + ... + x_mw_m = c$$

where $c = \pm\sqrt{-2\ln\xi} - b$ is not a constant. Thus, the resulting decision boundary is not a hyperplane. Two parallel planes exist.

Question 2

Contradiction proof. Assume it is linear separable and satisfy the requirements, we have:

$$\phi = \begin{cases} 0 & \text{if } (v) \geq \xi \\ 1 & \text{if } (v) < \xi \end{cases} \quad where\, v = x_1w_1 + x_2w_2 + b$$

Thus, we have the following four inequalities:

$$\begin{cases} 0w_1 + 0w_2 + b \geq \xi \\ 1w_1 + 0w_2 + b < \xi \\ 0w_1 + 1w_2 + b < \xi \\ 1w_1 + 1w_2 + b \geq \xi \end{cases}$$

We can have:

$$b \geq \xi \tag{1}$$

$$w_1 + b < \xi \tag{2}$$

$$w_2 + b < \xi \tag{3}$$

$$w_1 + w_2 + b \geq \xi \tag{4}$$

(1) + (4), we have:

$$w_1 + w_2 + 2b \geq 2\xi$$

(2) + (3), we have:

$$w_1 + w_2 + 2b < 2\xi$$

Thus, we can see the formula of (1) - (4) cannot be achieved simultaneously. The assumption we made initially is incorrect and the XOR is not linearly separable.

Question 3

In [1]:

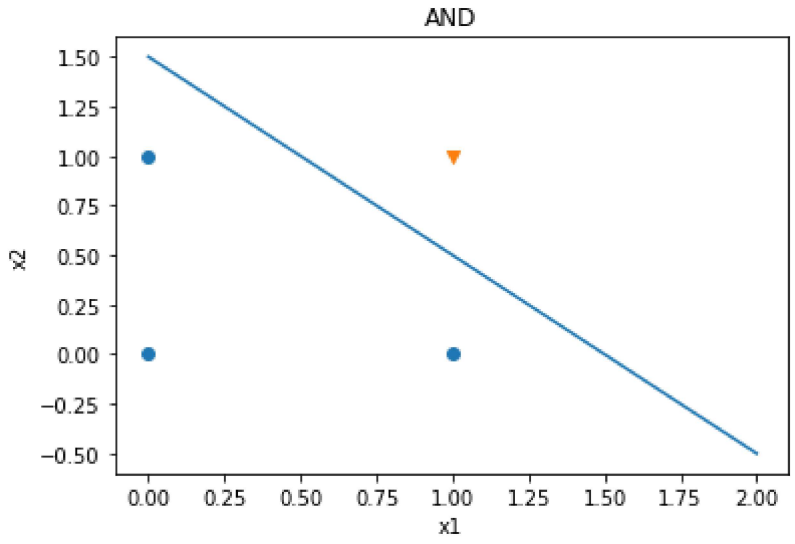
```
import numpy as np
import matplotlib.pyplot as plt
```

(1)Off-line calculations for AND, OR, COMPLENET, and NAND

AND: x1 + x2 -1.5; w= (-1.5, 1, 1)

In [2]:

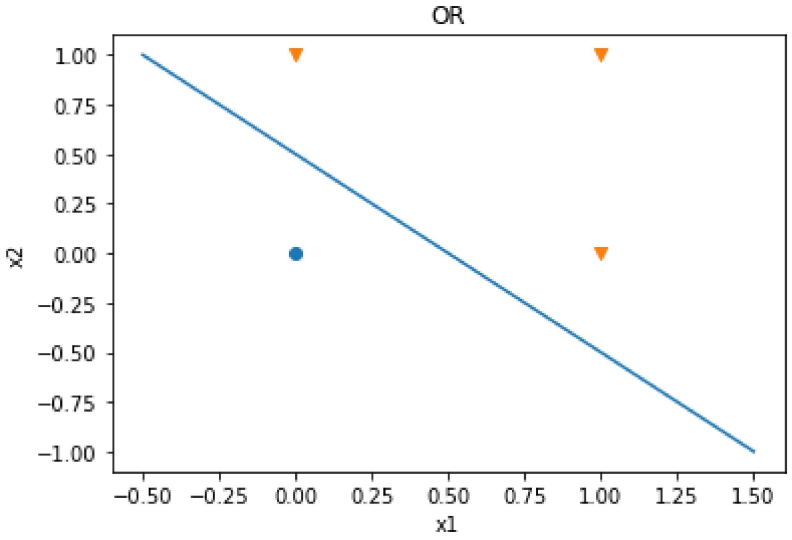
```
x = np.arange(0, 3)
y = -x +1.5

plt.scatter([0, 0, 1], [0, 1, 0])
plt.scatter([1], [1], marker = 'v')
plt.plot(x, y)
plt.title("AND")
plt.xlabel("x1")
plt.ylabel("x2")
plt.show()
```



OR: x1 + x2 -0.5; w= (-0.5, 1, 1)

In [3]:

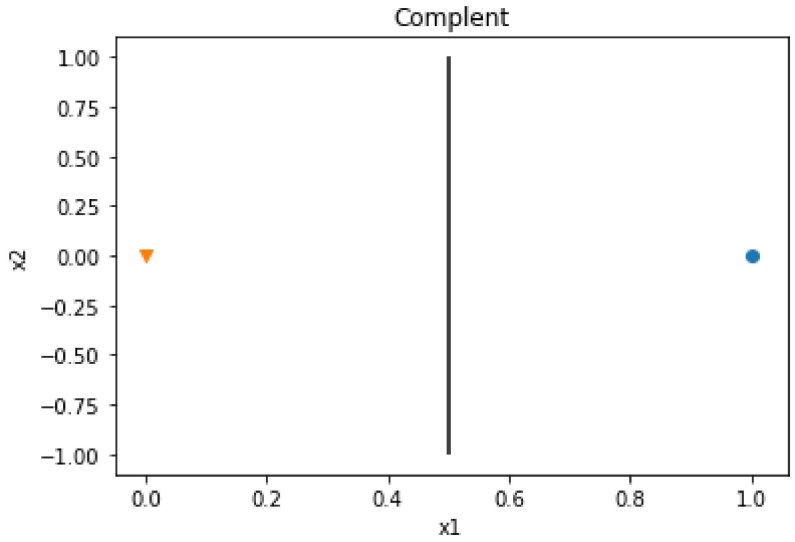
```
x = np.arange(-0.5, 2)
y = -x +0.5

plt.scatter([0], [0])
plt.scatter([1, 0, 1], [1, 1, 0], marker = 'v')
plt.plot(x, y)
plt.title("OR")
plt.xlabel("x1")
plt.ylabel("x2")
plt.show()
```



Complent: 0.5 - x1; w= (0.5, -1)

In [4]:

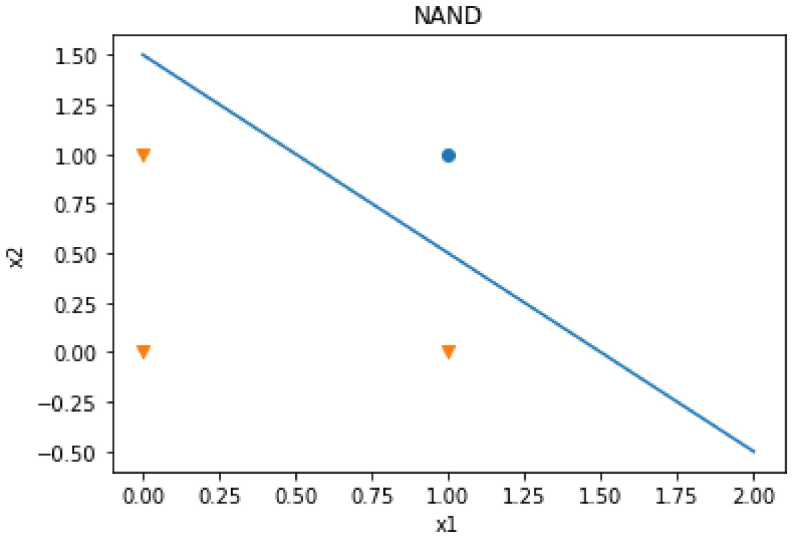
```
plt.scatter([1], [0])
plt.scatter([0], [0],marker = 'v')
plt.vlines(0.5,-1, 1)
plt.title("Complent")
plt.xlabel("x1")
plt.ylabel("x2")
plt.show()
```



NAND: 1.5 - x1 - x2; w= (1.5, -1, -1)

In [5]:

```
x = np.arange(0, 3)
y = -x +1.5

plt.scatter([1], [1])
plt.scatter([0, 0, 1], [0, 1, 0], marker = 'v')
plt.plot(x, y)
plt.title("NAND")
plt.xlabel("x1")
plt.ylabel("x2")
plt.show()
```

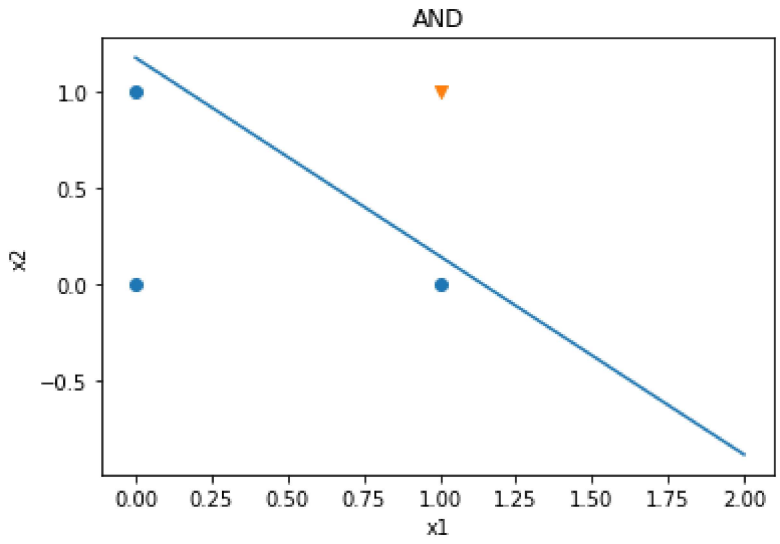


(2)Demonstrate the implementation of the logic functions AND, OR, COMPLEMENT and NAND with selection of weights by learning procedure. Suppose initial weights are chosen randomly and learning rate is 1.0. Plot out the trajectories of the weights for each case. Compare the results with those obtained in (a). Try other learning rates, and report your observations with different learning rates.

It can be achieved by updating the weights iteratively. W(n+1) = w(n) + e *learning rate* x(n). The stoping criteria is to achieve at certain number of iterations.

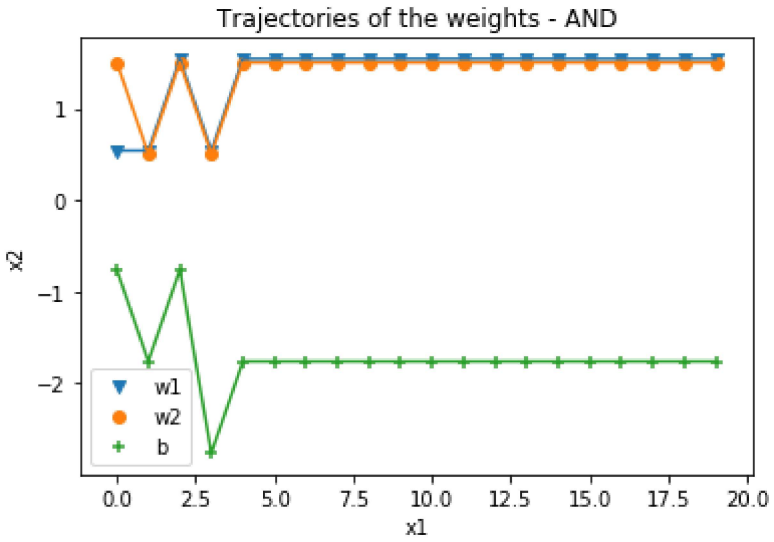
In [9]:

```
max_iteration = 20
error_threshold = 0.01
l_rate = 1.0
i = 0

x = np.array([[0,0,1,1],[0,1,0,1]])
bias = np.ones((1, x.shape[1]))
x = np.concatenate((x, bias))
y_and = np.array([0,0,0,1])
w = np.random.randn(1,x.shape[0])
w1_record = []
w2_record = []
b_record = []
while i < max_iteration:
    # record weights
    w1_record.append(w[0,0])
    w2_record.append(w[0,1])
    b_record.append(w[0,2])

    v = np.dot(w, x)
    y = np.array(v>0.0, dtype= float)
    error = y_and - y
    #update
    w = w + l_rate * np.dot(error, x.T)
    i += 1

x = np.arange(2.5)
line = -w[0,0]/w[0,1] * x - w[0,2]/w[0,1]
plt.scatter([0, 0, 1], [0, 1, 0])
plt.scatter([1], [1], marker = 'v')
plt.plot(x, line)
plt.title("AND")
plt.xlabel("x1")
plt.ylabel("x2")
plt.show()
```



In [10]:

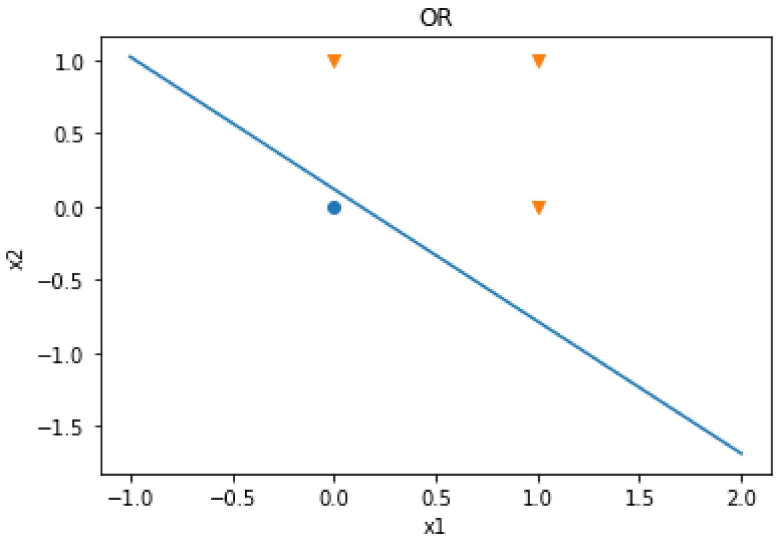
```
rge = np.array(range(20))
plt.scatter(rge,w1_record, label = 'w1', marker = 'v')
plt.plot(rge,w1_record)
plt.scatter(rge,w2_record,label = 'w2', marker = 'o')
plt.plot(rge,w2_record)
plt.scatter(rge,b_record, label = 'b', marker = '+')
plt.plot(rge,b_record)
plt.title('Trajectories of the weights - AND')
plt.xlabel("x1")
plt.ylabel("x2")
plt.legend()
plt.show()
```



OR

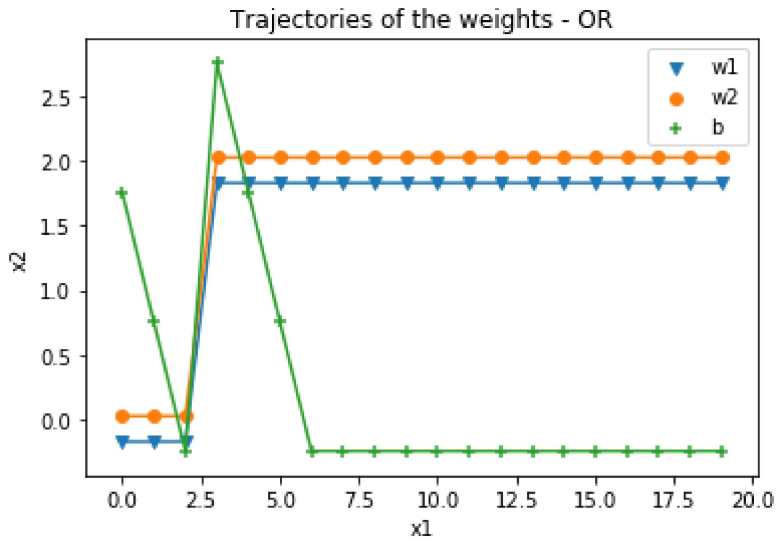
In [11]:

```
x = np.array([[0,0,1,1],[0,1,0,1]])
bias = np.ones((1, x.shape[1]))
x = np.concatenate((x, bias))
y_or = np.array([0,1,1,1])
w = np.random.randn(1,x.shape[0])
w1_record = []
w2_record = []
b_record = []
i =0

while i < max_iteration:
    # record weights
    w1_record.append(w[0,0])
    w2_record.append(w[0,1])
    b_record.append(w[0,2])

    v = np.dot(w, x)
    y = np.array(v>0.0, dtype= float)
    error = y_or - y

    #update
    w = w + l_rate * np.dot(error, x.T)
    i += 1

x = np.arange(-1,3)
line = -w[0,0]/w[0,1] * x - w[0,2]/w[0,1]
plt.scatter([0], [0])
plt.scatter([1, 0, 1], [1, 1, 0], marker = 'v')
plt.plot(x, line)
plt.title("OR")
plt.xlabel("x1")
plt.ylabel("x2")
plt.show()
```



In [12]:

```
rge = np.array(range(20))
plt.scatter(rge,w1_record, label = 'w1', marker = 'v')
plt.plot(rge,w1_record)
plt.scatter(rge,w2_record,label = 'w2', marker = 'o')
plt.plot(rge,w2_record)
plt.scatter(rge,b_record, label = 'b', marker = '+')
plt.plot(rge,b_record)
plt.title('Trajectories of the weights - OR')
plt.xlabel("x1")
plt.ylabel("x2")
plt.legend()
plt.show()
```

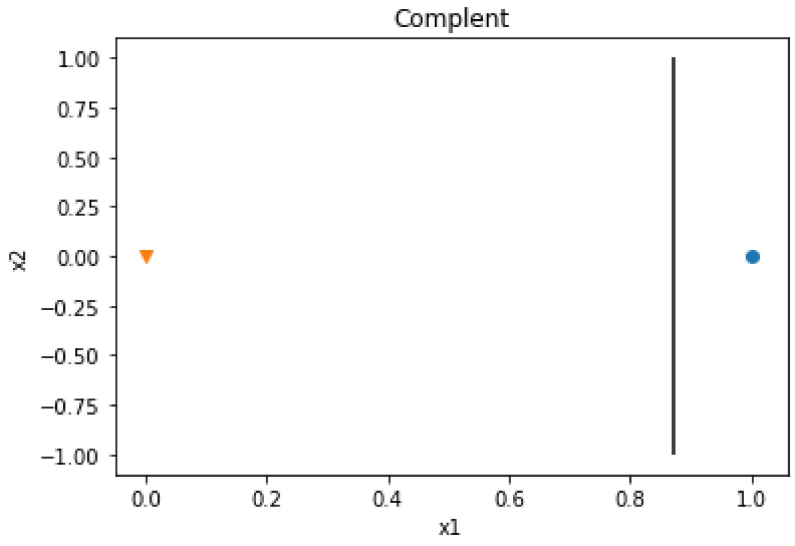


Complent:

In [16]:

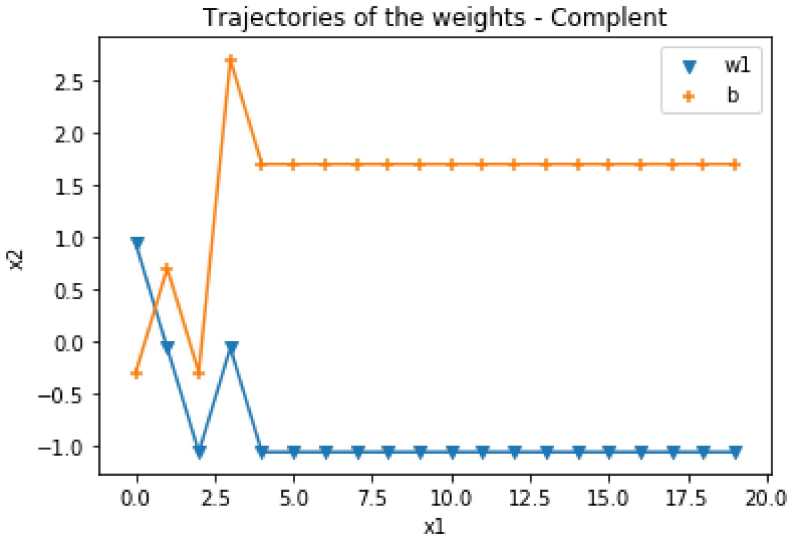
```
x = np.array([[0,1]])
bias = np.ones((1, x.shape[1]))
x = np.concatenate((x, bias))
y_and = np.array([1,0])
w = np.random.randn(1,x.shape[0])
w1_record = []
b_record = []

print(w)
while i < max_iteration:
    # record weights
    w1_record.append(w[0,0])
    b_record.append(w[0,1])

    v = np.dot(w, x)
    y = np.array(v>0.0, dtype= float)
    error = y_and - y
    #update
    w = w + l_rate * np.dot(error, x.T)
    i += 1

plt.scatter([1], [0])
plt.scatter([0], [0],marker = 'v')
plt.vlines(w[0,1]/w[0,0], -1,1)
plt.title("Complent")
plt.xlabel("x1")
plt.ylabel("x2")
plt.show()
```

```
[[-0.78155667 -0.68066714]]
```



In [19]:

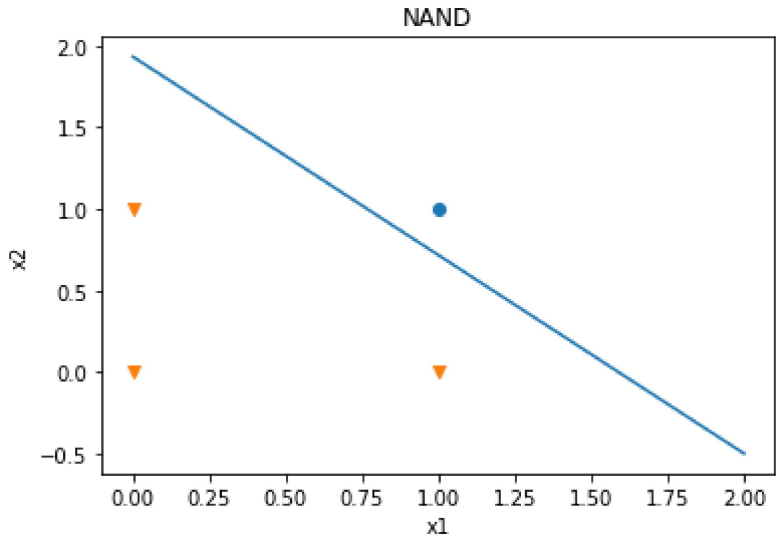
```
rge = np.array(range(20))
plt.scatter(rge,w1_record, label = 'w1', marker = 'v')
plt.plot(rge,w1_record)
plt.scatter(rge,b_record, label = 'b', marker = '+')
plt.plot(rge,b_record)
plt.title('Trajectories of the weights - Complent')
plt.xlabel("x1")
plt.ylabel("x2")
plt.legend()
plt.show()
```



NAND:

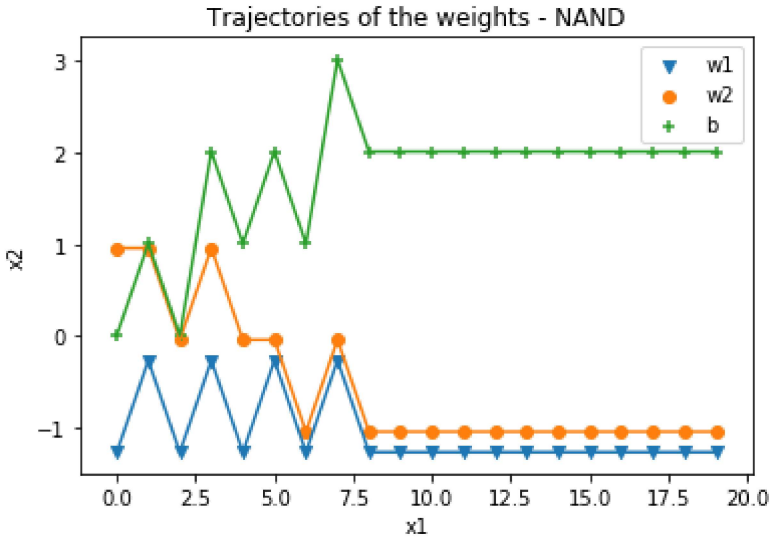
In [20]:

```
x = np.array([[0,0,1,1],[0,1,0,1]])
bias = np.ones((1, x.shape[1]))
x = np.concatenate((x, bias))
y_and = np.array([1,1,1,0])
w = np.random.randn(1,x.shape[0])
w1_record = []
w2_record = []
b_record = []
i = 0

while i < max_iteration:
    # record weights
    w1_record.append(w[0,0])
    w2_record.append(w[0,1])
    b_record.append(w[0,2])

    v = np.dot(w, x)
    y = np.array(v>0.0, dtype= float)
    error = y_and - y
    #update
    w = w + l_rate * np.dot(error, x.T)
    i += 1

x = np.arange(2.5)
line = -w[0,0]/w[0,1] * x - w[0,2]/w[0,1]
plt.scatter([1], [1])
plt.scatter([0, 0, 1], [0, 1, 0], marker = 'v')
plt.plot(x, line)
plt.title("NAND")
plt.xlabel("x1")
plt.ylabel("x2")
plt.show()
```



In [21]:

```
rge = np.array(range(20))
plt.scatter(rge,w1_record, label = 'w1', marker = 'v')
plt.plot(rge,w1_record)
plt.scatter(rge,w2_record,label = 'w2', marker = 'o')
plt.plot(rge,w2_record)
plt.scatter(rge,b_record, label = 'b', marker = '+')
plt.plot(rge,b_record)
plt.title('Trajectories of the weights - NAND')
plt.xlabel("x1")
plt.ylabel("x2")
plt.legend()
plt.show()
```

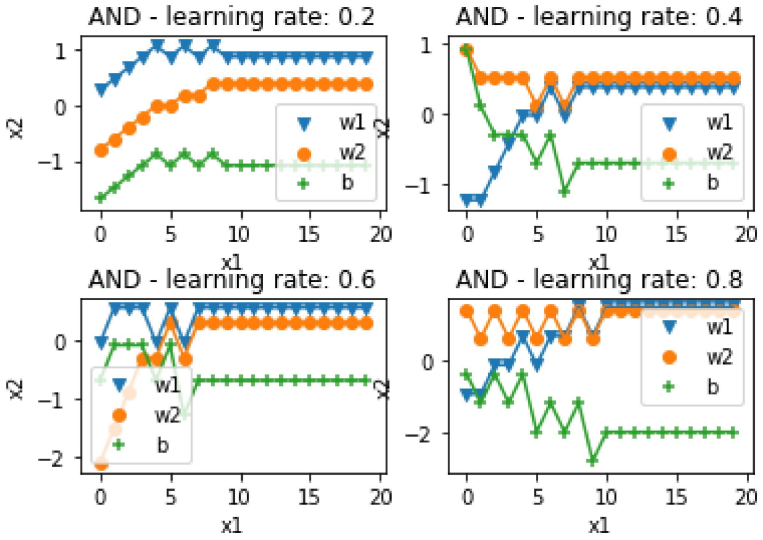


Take the AND problem as an example to explore the effects of learning rate.

In [24]:

```
max_iteration = 20
error_threshold = 0.01
l_rates = [0.2, 0.4, 0.6, 0.8]


x = np.array([[0,0,1,1],[0,1,0,1]])
bias = np.ones((1, x.shape[1]))
x = np.concatenate((x, bias))
y_and = np.array([0,0,0,1])
w1_record = np.zeros([4, 20])
w2_record = np.zeros([4, 20])
b_record = np.zeros([4, 20])

for num, l_rate in enumerate(l_rates):
    i = 0
    w = np.random.randn(1,x.shape[0])
    while i < max_iteration:
        # record weights
        w1_record[num, i] = (w[0,0])
        w2_record[num, i] = (w[0,1])
        b_record[num, i] = (w[0,2])

        v = np.dot(w, x)
```

```
        y = np.array(v>0.0, dtype= float)
        error = y_and - y
        #update
        w = w + l_rate * np.dot(error, x.T)
        i += 1

rge = np.array(range(20))
plt.subplots_adjust(wspace =0.2, hspace =0.5)
plt.subplot(2,2,1)
plt.scatter(rge,w1_record[0, :], label = 'w1', marker = 'v')
plt.plot(rge,w1_record[0, :])
plt.scatter(rge,w2_record[0, :],label = 'w2', marker = 'o')
plt.plot(rge,w2_record[0, :])
plt.scatter(rge,b_record[0, :], label = 'b', marker = '+')
plt.plot(rge,b_record[0, :])
plt.title('AND - learning rate: 0.2')
plt.xlabel("x1")
plt.ylabel("x2")
plt.legend(loc = 0)
plt.subplot(2,2,2)
plt.scatter(rge,w1_record[1, :], label = 'w1', marker = 'v')
plt.plot(rge,w1_record[1, :])
plt.scatter(rge,w2_record[1, :],label = 'w2', marker = 'o')
plt.plot(rge,w2_record[1, :])
plt.scatter(rge,b_record[1, :], label = 'b', marker = '+')
plt.plot(rge,b_record[1, :])
plt.title('AND - learning rate: 0.4')
plt.xlabel("x1")
plt.ylabel("x2")
plt.legend()
plt.subplot(2,2,3)
plt.scatter(rge,w1_record[2, :], label = 'w1', marker = 'v')
plt.plot(rge,w1_record[2, :])
plt.scatter(rge,w2_record[2, :],label = 'w2', marker = 'o')
plt.plot(rge,w2_record[2, :])
plt.scatter(rge,b_record[2, :], label = 'b', marker = '+')
plt.plot(rge,b_record[2, :])
plt.title('AND - learning rate: 0.6')
plt.xlabel("x1")
plt.ylabel("x2")
plt.legend()
plt.subplot(2,2,4)
plt.scatter(rge,w1_record[3, :], label = 'w1', marker = 'v')
plt.plot(rge,w1_record[3, :])
plt.scatter(rge,w2_record[3, :],label = 'w2', marker = 'o')
plt.plot(rge,w2_record[3, :])
plt.scatter(rge,b_record[3, :], label = 'b', marker = '+')
plt.plot(rge,b_record[3, :])
plt.title('AND - learning rate: 0.8')
plt.xlabel("x1")
plt.ylabel("x2")
plt.legend()
plt.show()
```

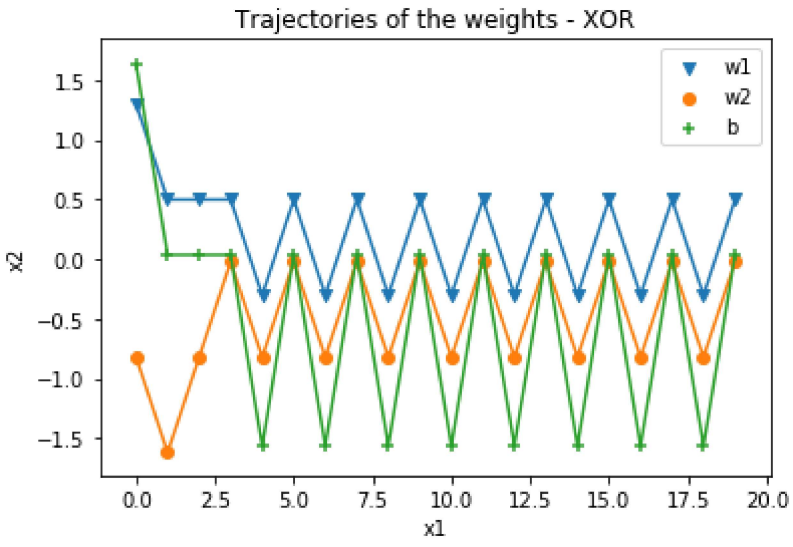


Through observation of above four figures, we can state all weights would converge even with different learning rate. For this problem, we can set learning rate as 0.6 since it converge fastest.

(3)What would happen if the perceptron is applied to implement the EXCLUSIVE OR function with selection of weights by learning procedure? Suppose initial weight is chosen randomly and learning rate is 1.0. Do the computer experiment and explain your finding.

In [25]:

```
x = np.array([[0,0,1,1],[0,1,0,1]])
bias = np.ones((1, x.shape[1]))
x = np.concatenate((x, bias))
y_or = np.array([0,1,1, 0])
w = np.random.randn(1,x.shape[0])
w1_record = []
w2_record = []
b_record = []
i =0

while i < max_iteration:
    # record weights
    w1_record.append(w[0,0])
    w2_record.append(w[0,1])
    b_record.append(w[0,2])

    v = np.dot(w, x)
    y = np.array(v>0.0, dtype= float)
    error = y_or - y

    #update
    w = w + l_rate * np.dot(error, x.T)
    i += 1

rge = np.array(range(20))
plt.scatter(rge,w1_record, label = 'w1', marker = 'v')
plt.plot(rge,w1_record)
plt.scatter(rge,w2_record,label = 'w2', marker = 'o')
plt.plot(rge,w2_record)
plt.scatter(rge,b_record, label = 'b', marker = '+')
plt.plot(rge,b_record)
plt.title('Trajectories of the weights - XOR')
plt.xlabel("x1")
plt.ylabel("x2")
plt.legend()
plt.show()
```



Since XOR is not linear seperable, the w1, w2, and b will not converge at finite iterations. They fluctuate all the time and boundary decision can not be figured out.

In [ ]:

Question 4

1

Refer to the slides, we have the cost function for LLS method.

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

where the error vector is $\boldsymbol{e} = \boldsymbol{d} - \boldsymbol{x}\boldsymbol{w}$ Thus, we have:

$$\begin{aligned} \frac{\alpha \boldsymbol{E}}{\alpha \boldsymbol{w}} &= \frac{\alpha \boldsymbol{E}}{\alpha \boldsymbol{e}} \frac{\alpha \boldsymbol{e}}{\alpha \boldsymbol{w}} \\ \frac{\alpha \boldsymbol{E}}{\alpha \boldsymbol{w}} &= -2\boldsymbol{e}^T \boldsymbol{X} = 0 \\ \boldsymbol{w} &= (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{d} \end{aligned}$$

Now we have the pairs and fed into the code, we can get the w = 0.38733906, 1.60944206. Thus the line can be written as y = 1.60944206 x + 0.38733906.

In [34]:

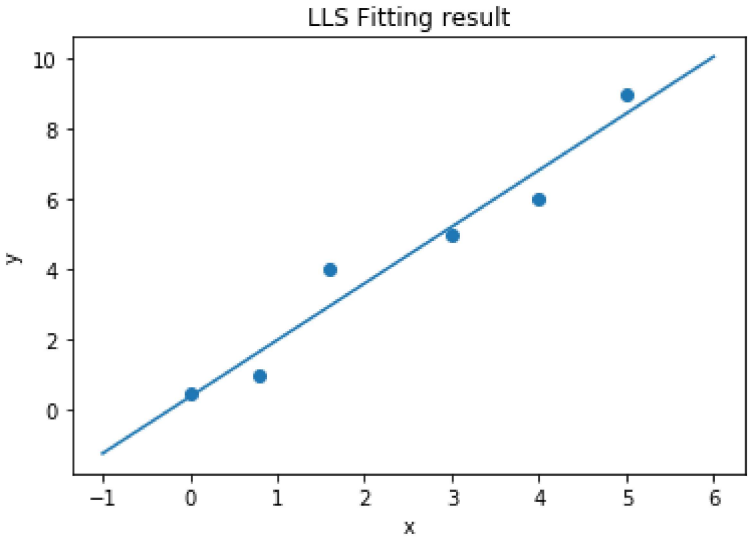
```
import numpy as np
import matplotlib.pyplot as plt
```

(1) Find the solution of w and b using the standard linear least-squares (LLS) method. Plot out the fitting result.

In [35]:

```
x = np.array([0, 0.8, 1.6, 3, 4.0, 5], dtype = float)
x_ = np.ones(x.shape[0])
x  = np.vstack((x_, x)).T
y = np.array([0.5, 1, 4, 5, 6, 9], dtype = float).T
w = np.linalg.inv(x.T.dot(x)).dot(x.T.dot(y))
print(w)
# plot
i = np.arange(-1, 7)
j = w[1]*i + w[0]
plt.scatter(x[:,1], y)
plt.plot(i, j)
plt.title("LLS Fitting result")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

```
[0.38733906 1.60944206]
```

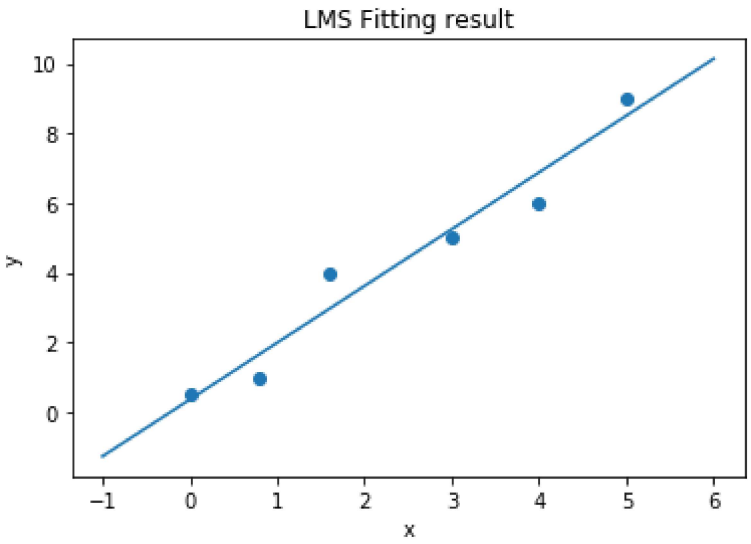


(2) Suppose that initial weight is chosen randomly and learning rate is 0.01. Find the solution of w and b using the least-mean-square (LMS) algorithm for 100 epochs. Plot out the fitting result and the trajectories of the weights versus learning steps. Will the weights converge?

In [45]:

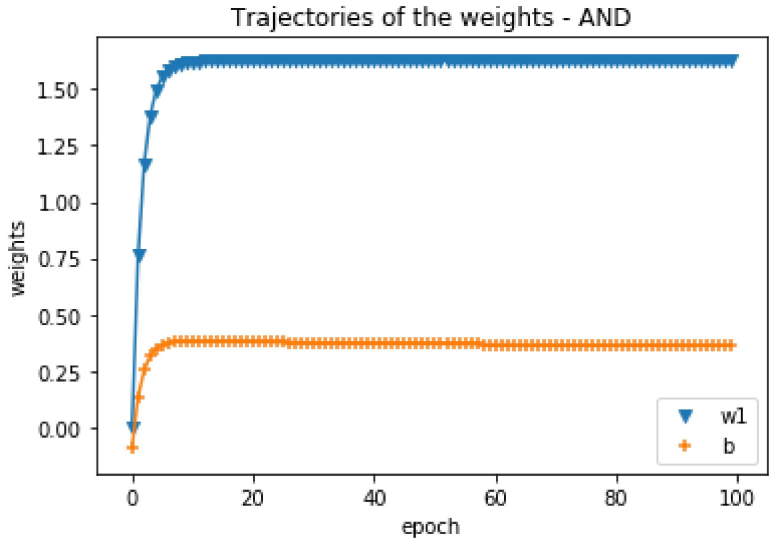
```
l_rate = 0.01
epoch = 100
w = np.random.randn(1,x.shape[1]).flatten()
w1_record = []
b_record = []

for i in range(epoch):
    w1_record.append(w[1])
    b_record.append(w[0])
    for j, label in enumerate(y):
        e = y[j] - x[j, :].dot(w)
        w = w + l_rate * e *(x[j])

print(w)
i = np.arange(-1, 7)
j = w[1]*i + w[0]
plt.scatter(x[:,1], y)
plt.plot(i, j)
plt.title("LMS Fitting result")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

```
[0.36294148 1.62940456]
```



In [46]:

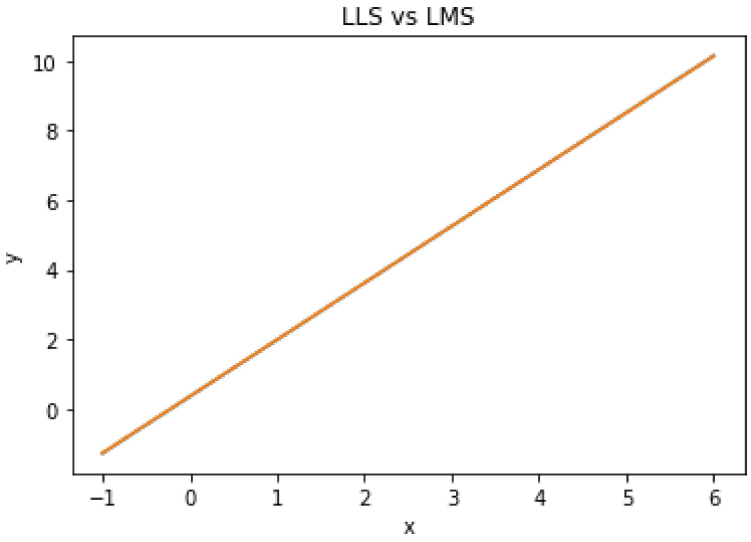
```
rge = np.array(range(100))
plt.scatter(rge,w1_record, label = 'w1', marker = 'v')
plt.plot(rge,w1_record)
plt.scatter(rge,b_record, label = 'b', marker = '+')
plt.plot(rge,b_record)
plt.title('Trajectories of the weights')
plt.xlabel("epoch")
plt.ylabel("weights")
plt.legend()
plt.show()
```



(3) Compare the results obtained by LLS and the LMS methods.

In [47]:

```
i = np.arange(-1, 7)
j = 1.63154731*i + 0.35465501
k = 1.62940456*i + 0.36294148
plt.plot(i, j)
plt.plot(i, k)
plt.title("LLS vs LMS")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

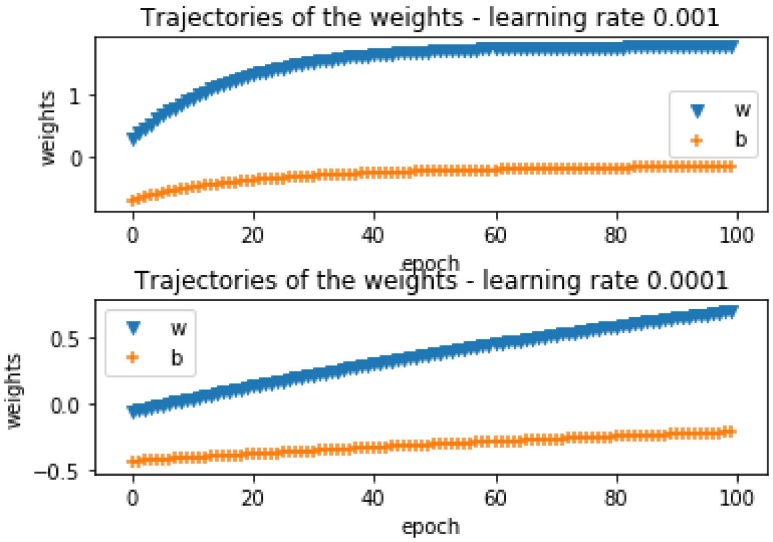


For LLS method, we have w = [0.38733906 1.60944206].In terms of LMS, weights are [0.36294148 1.62940456]. These two lines are actually very close to each other. In terms of computation time, LLS is much more efficient than LMS. However, for high dimension data, LMS method is better than LLS.

(4) Repeat the simulation study in b) with different learning rates , and explain your findings.

In [50]:

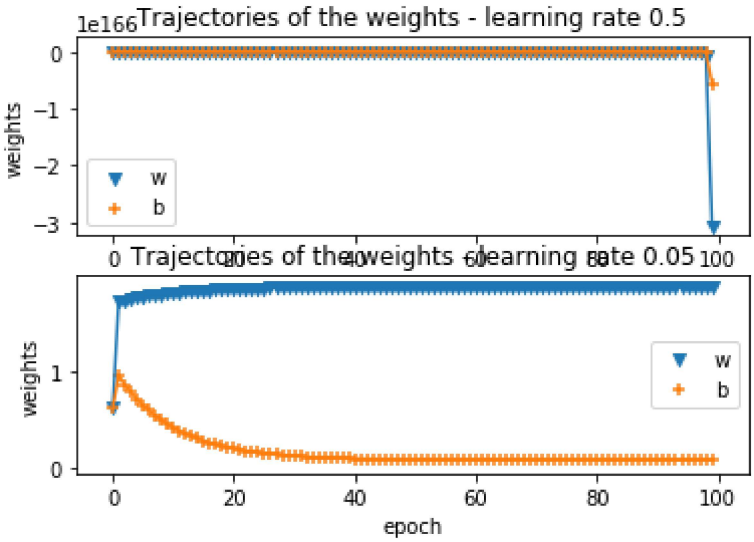
```
l_rates = [0.001, 0.0001, 0.5, 0.05, ]
epoch = 100
w_record = np.zeros([4, 100])
b_record = np.zeros([4, 100])

for num, l_rate in enumerate(l_rates):
    w = np.random.randn(1,x.shape[1]).flatten()
    for i in range(epoch):
        w_record[num, i] = (w[1])
        b_record[num, i] = (w[0])
        for j, label in enumerate(y):
            e = y[j] - x[j, :].dot(w)
            w = w + l_rate * e *(x[j])


rge = np.array(range(100))
plt.subplots_adjust(wspace =0.2, hspace =0.5)
plt.subplot(2,1,1)
plt.scatter(rge,w_record[0, :], label = 'w', marker = 'v')
plt.plot(rge,w_record[0, :])
plt.scatter(rge,b_record[0, :], label = 'b', marker = '+')
plt.plot(rge,b_record[0, :])
plt.title('Trajectories of the weights - learning rate %r' %(l_rates[0]))
plt.xlabel("epoch")
plt.ylabel("weights")
plt.legend()
plt.subplot(2,1,2)
plt.scatter(rge,w_record[1, :], label = 'w', marker = 'v')
plt.plot(rge,w_record[1, :])
plt.scatter(rge,b_record[1, :], label = 'b', marker = '+')
plt.plot(rge,b_record[1, :])
plt.title('Trajectories of the weights - learning rate %r' %(l_rates[1]))
plt.xlabel("epoch")
plt.ylabel("weights")
plt.legend()
plt.show()
```



For the smaller learning rate like 0.001 and 0.0001. It can converge like 0.001 but probably needs more epoches like 0.0001.

In [53]:

```
plt.subplot(2,1,1)
plt.scatter(rge,w_record[2, :], label = 'w', marker = 'v')
plt.plot(rge,w_record[2, :])
plt.scatter(rge,b_record[2, :], label = 'b', marker = '+')
plt.plot(rge,b_record[2, :])
plt.title('Trajectories of the weights - learning rate %r' %(l_rates[2]))
plt.xlabel("epoch")
plt.ylabel("weights")
plt.legend()
plt.subplot(2,1,2)
plt.scatter(rge,w_record[3, :], label = 'w', marker = 'v')
plt.plot(rge,w_record[3, :])
plt.scatter(rge,b_record[3, :], label = 'b', marker = '+')
plt.plot(rge,b_record[3, :])
plt.title('Trajectories of the weights - learning rate %r' %(l_rates[3]))
plt.xlabel("epoch")
plt.ylabel("weights")
plt.legend()
plt.show()
```



For the larger learning rate, it might fail to converge like 0.5. The reason can be concluded as the Taylor approximation doesn't hold if a very large learning rate is set.

In [ ]:

Question 5

$r(i)$ is the weighted factors for each output error $e(i)$, we can express it as the matrix formulation.

$$\boldsymbol{R_{nxn}} = diag(r(1), r(2), r(3), ..., r(n))$$

We also know error can be expressed as $\boldsymbol{e} = \boldsymbol{d} - \boldsymbol{X}\boldsymbol{w}$ Thus, the cost function can be written as below:

$$\begin{aligned} J(W) &= \sum_{i=1}^{n} r(i)e(i) \\ &= \boldsymbol{e}^T \boldsymbol{R}\boldsymbol{e} \end{aligned}$$

In order to calculate the optimal parameter w^* such that the following cost function J(w) is minimized. We have:

$$\begin{aligned} \frac{\alpha J(w)}{\alpha w} &= \frac{\alpha J(w)}{\alpha e}\frac{\alpha e}{\alpha w} = 0 \\ &= -2\boldsymbol{e^T}\boldsymbol{R}\boldsymbol{X} = 0 \end{aligned}$$

Replace the $\boldsymbol{e}$, we have:

$$\begin{aligned} -2(\boldsymbol{d} - \boldsymbol{X}\boldsymbol{w})^T \boldsymbol{R}\boldsymbol{X} &= 0 \\ \boldsymbol{w}^T \boldsymbol{X}^T \boldsymbol{R}\boldsymbol{X} &= \boldsymbol{d}^T \boldsymbol{R}\boldsymbol{X} \\ \boldsymbol{w} &= (\boldsymbol{X}^T \boldsymbol{R}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{R}\boldsymbol{d} \end{aligned}$$

Thus the optimal parameter $w^* = (\boldsymbol{X}^T \boldsymbol{R}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{R}\boldsymbol{d}$, and cost funtion is minimized.