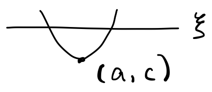
Wang Yizhuo A0225440R
Feb 6, 2021

Homework #1

Q1.

Since weights w can form a hyper-plane, the prerequisite of activate function is that $y = \varphi(wx)$ should be monotonic. Hence it can remain the property of linearly separable. But if we map it via a non-monotonic function, it will "fold" the decision boundary and it will no longer be a hyper-plane.

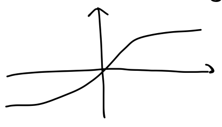
① $\varphi(v) = (v-a)^2 + c$



It is a quadratic function that when $(v-a)^2+c < \xi$ $\left(a-\sqrt{\xi-c} < v < a+\sqrt{\xi-c}\right)$, it belongs to C_2, C_1 otherwise.

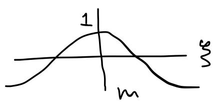
Hence, the decision boundary is not a linear plane.

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



$\dfrac{d\varphi(v)}{dv} = \dfrac{2e^{v}}{(e^{v}+1)^2} > 0$: monotonic, DB is a hyper-plane.

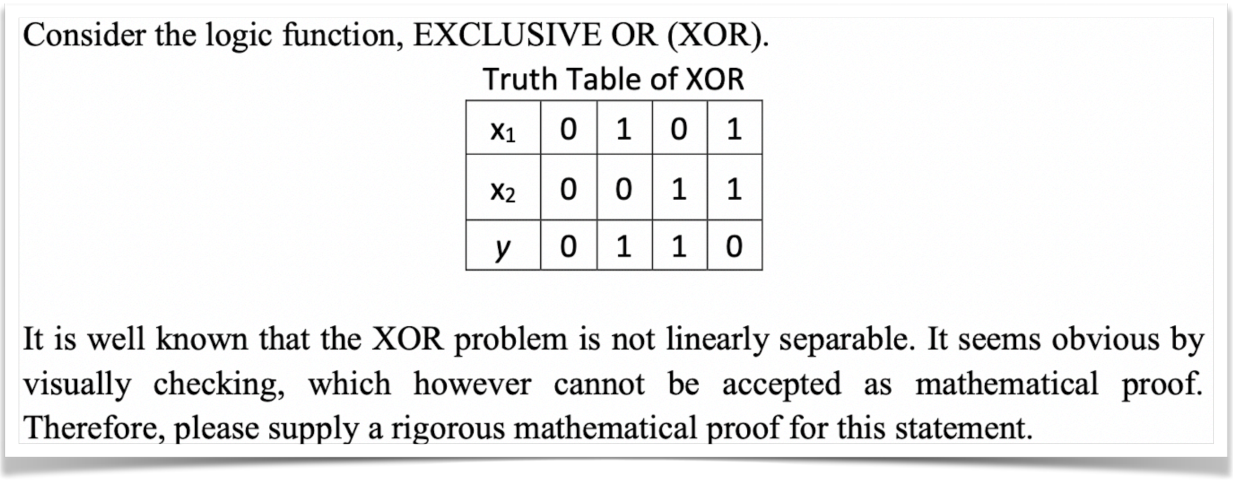
③ $\varphi(v) = e^{-\frac{(v-m)^2}{2}}$



When $\varphi(v) > \xi$ $\left(m-\sqrt{-2\ln\xi} < v < m+\sqrt{-2\ln\xi}\right)$, it belongs to C_1.

The decision boundry is not a hyper-plane.

Q2.



Consider the logic function, EXCLUSIVE OR (XOR).

Truth Table of XOR

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

It is well known that the XOR problem is not linearly separable. It seems obvious by visually checking, which however cannot be accepted as mathematical proof. Therefore, please supply a rigorous mathematical proof for this statement.

We can use reductio ad absurdum to rigorously prove this.

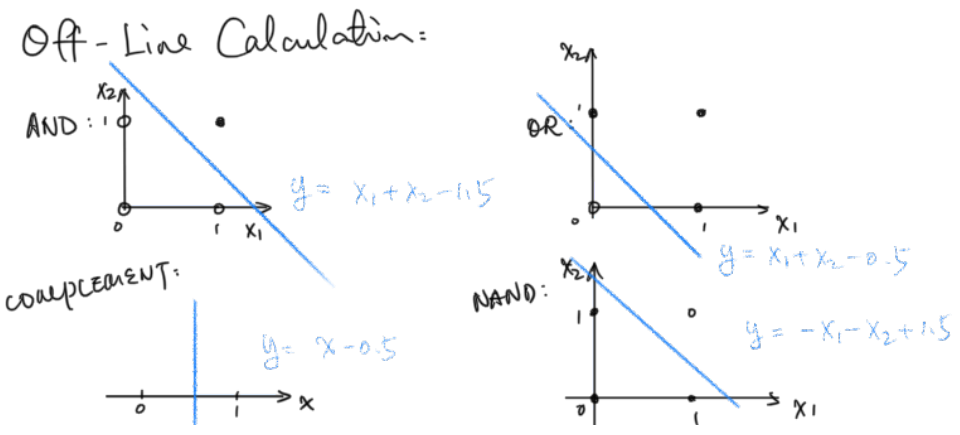
Assume the opposite of the statement is true, that is, XOR problem is linearly separable. So there must exist a line $w_0 + w_1x_1 + w_2x_2 = 0$ that can successfully classify all the points in the truth table. Then, the equation below holds.

$$
\begin{aligned}
w_0 + 0 \times w_1 + 0 \times w_2 < 0 &\Rightarrow w_0 < 0 \\
w_0 + 1 \times w_1 + 0 \times w_2 > 0 &\Rightarrow w_0 > -w_1 \\
w_0 + 0 \times w_1 + 1 \times w_2 > 0 &\Rightarrow w_0 > -w_2 \\
w_0 + 1 \times w_1 + 1 \times w_2 < 0 &\Rightarrow w_0 > -w_1 - w_2
\end{aligned}
$$

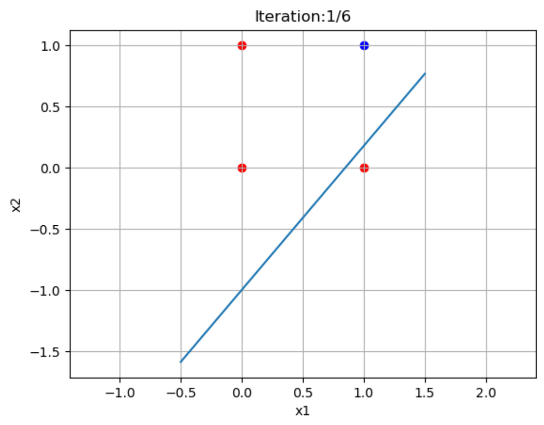
However, these equations are contradictory to each other. Hence, the assumption does not hold. XOR is not linearly separable.

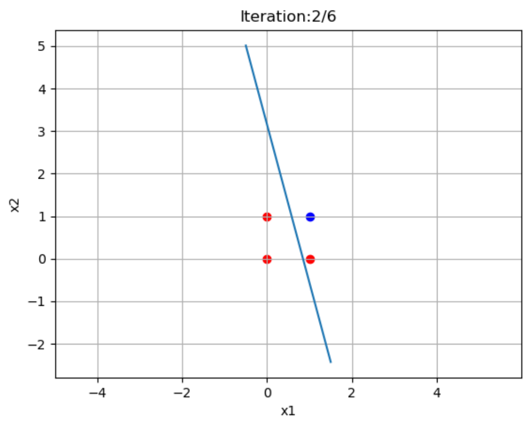
Q3

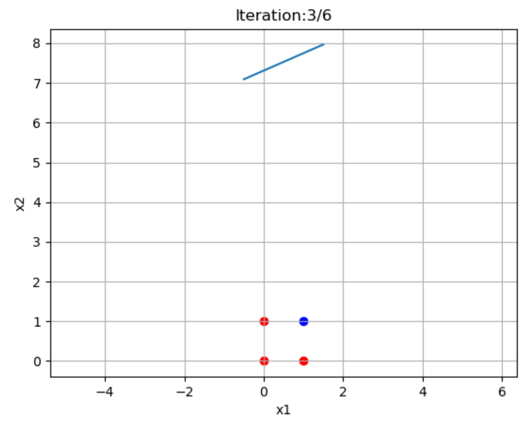
a).

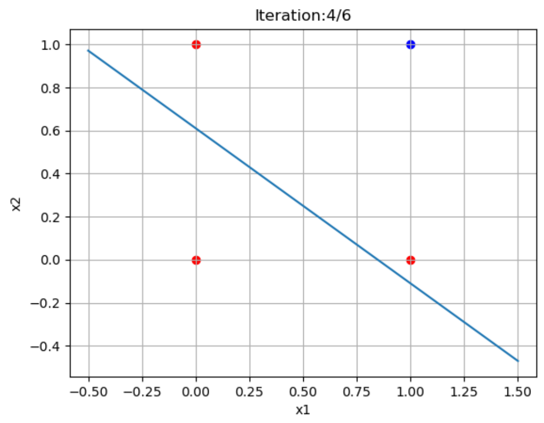


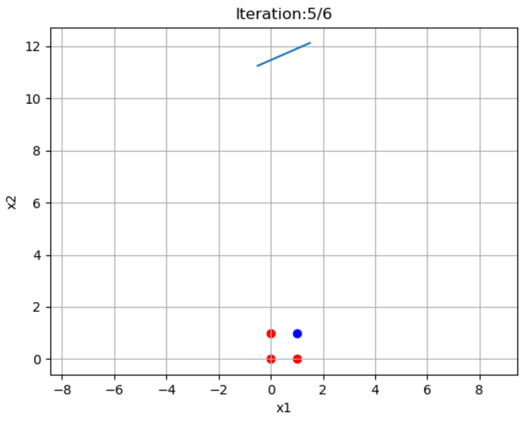
b). First, set learning rate to be 1.The iteration of boundary is shown below. Take AND logic function for example, the weights start at random values and end at $\mathbf{w} = [-1.758, 0.895, 1.240]^T$. Red dots represent value of 0, blue represent 1.

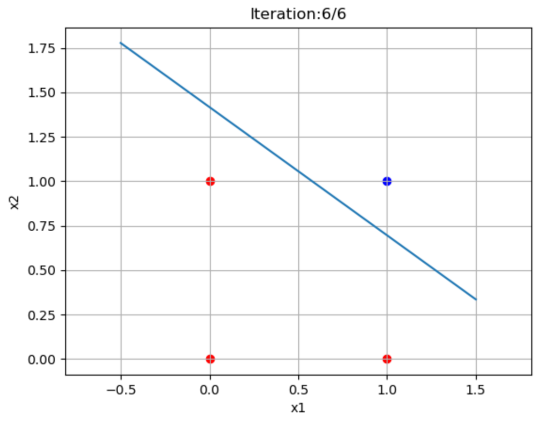




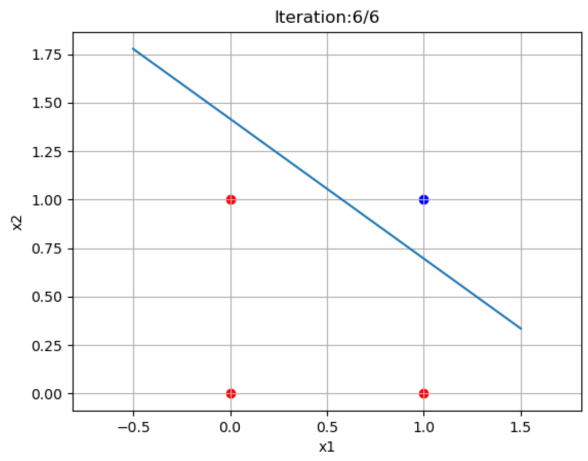


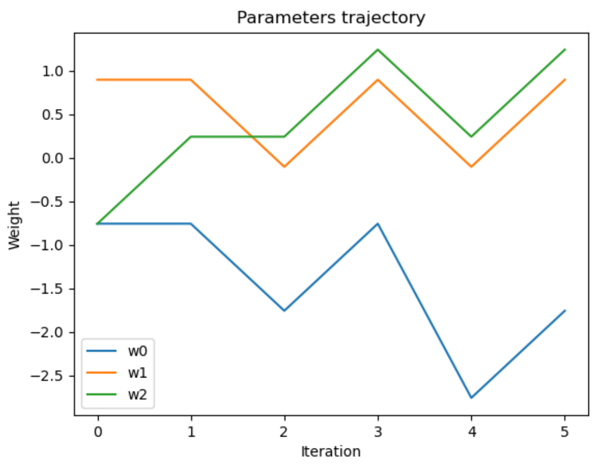




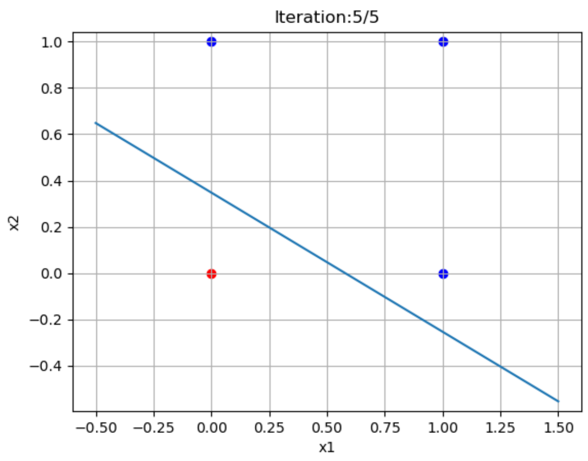


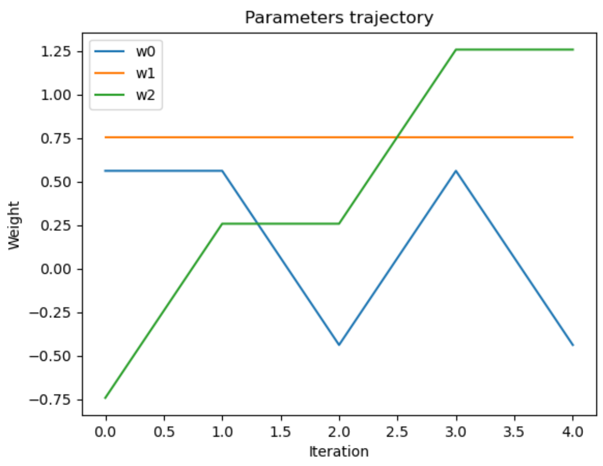
AND



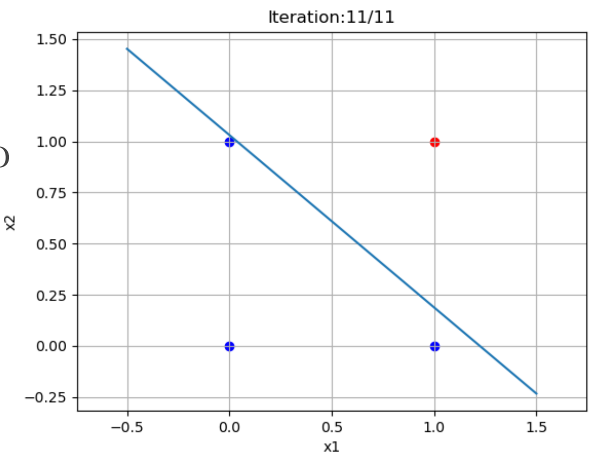


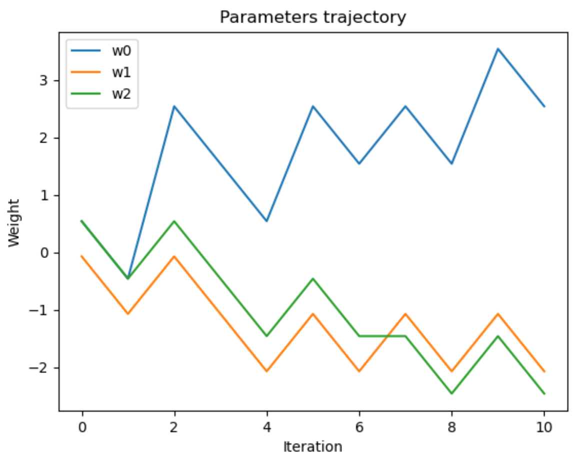
OR





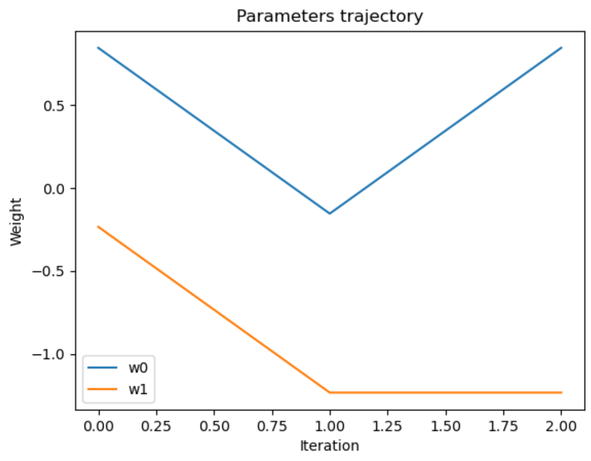
NAND





For logic functions AND, OR, COMPLEMENT, NAND, the parameters trajectories are shown above.

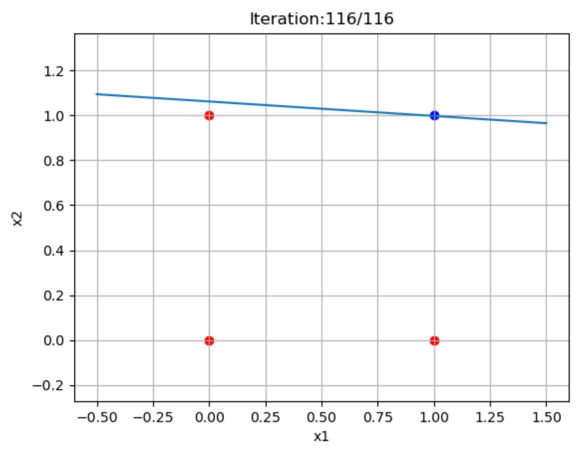
As we can see, the perceptron can successfully classify the logic functions AND, OR, COMPLEMENT and NAND, just as we had classified in a).

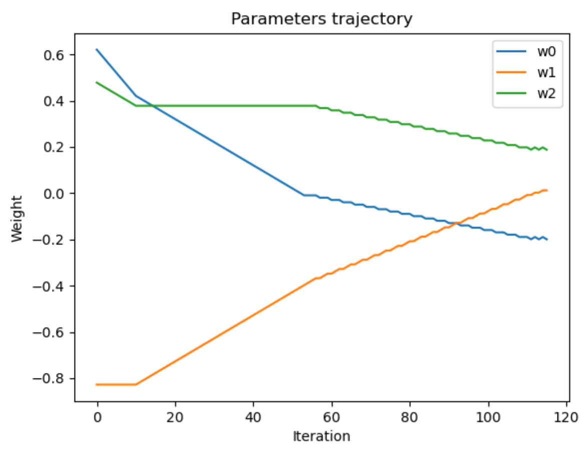


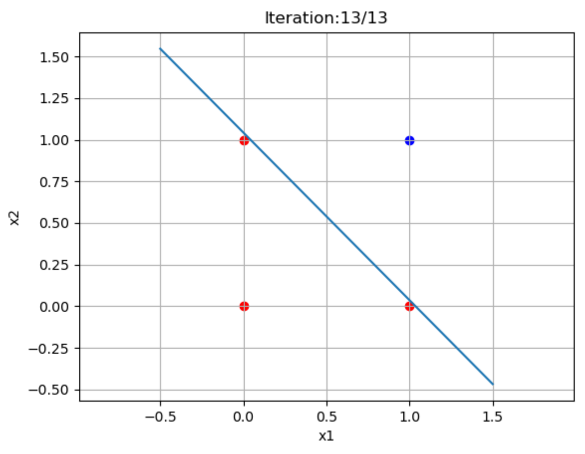
COMPLEMENT

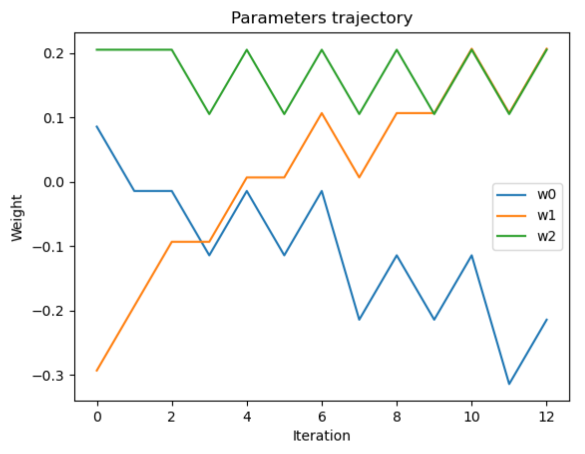
Then, we tune the learning rate $\eta = 0.01,0.1,10$, and see how the trajectories varies (take AND logic for instance).

LR=0.01

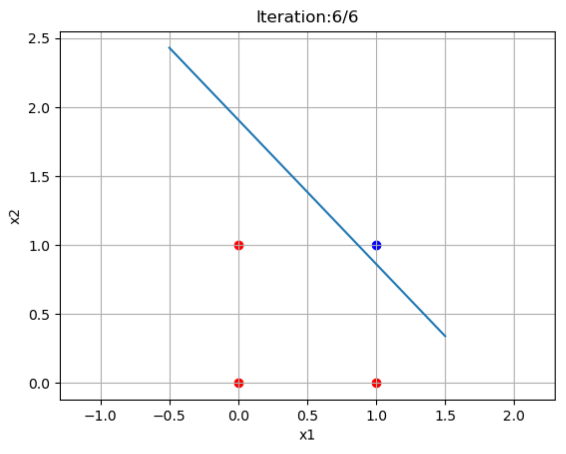


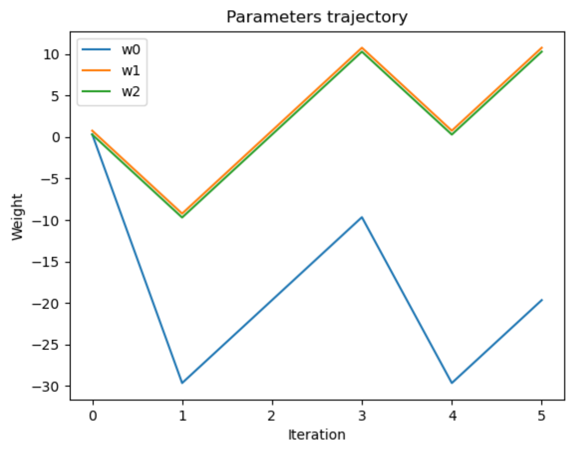






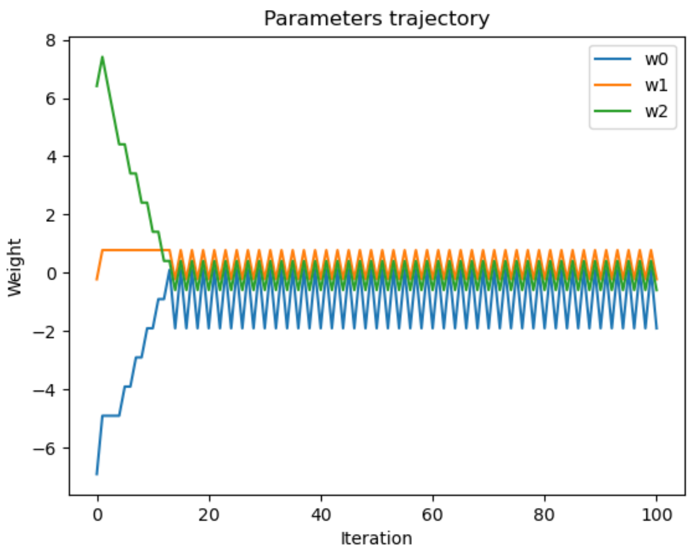
LR=10





As we can see, in general, if we set learning rate smaller, e.g. 0.01, 0.1 in this question, it needs more iterations to converge, and the decision boundary is close to the edges of the optimal one. If we set learning rate higher, the weights will change a lot in a single iteration, making parameters high and similar in value, thus make the equation ill-conditioned. Hence, the learning rate should set properly to make a good learning process.

c). If EXCLUSIVE OR is applied to a single perception, it will fail to converge, because XOR is theoretically not linearly separable. The parameter trajectory will oscillate and never converge. (The weights are initialized uniformly between -10 and 10.)



Q4.

a). Use linear least-squares (LLS) method to calculate the weights.

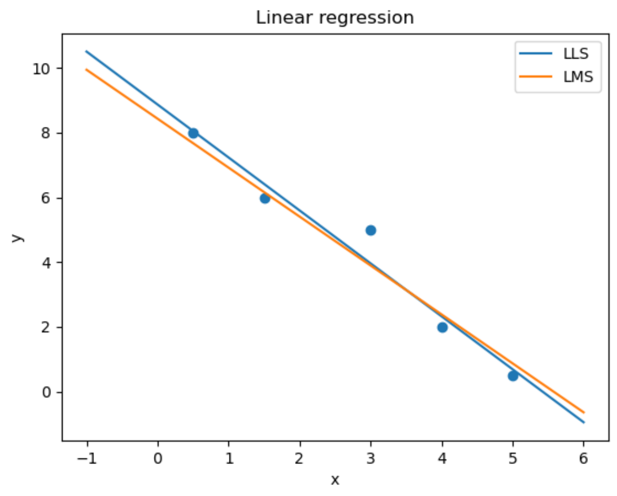
$$X = \begin{bmatrix} 1 & 0.5 \\ 1 & 1.5 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{bmatrix}$$, and the pseudo-inverse matrix is

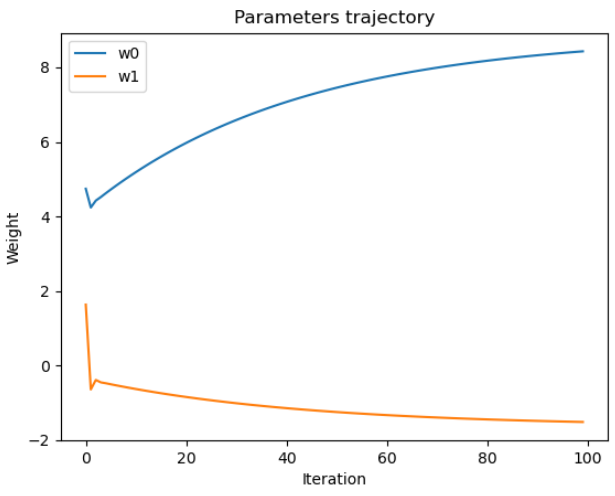
$$X^{\dagger} = \begin{bmatrix} 0.6842 & 0.4737 & 0.1579 & -0.0526 & -0.2632 \\ -0.1729 & -0.0977 & 0.0150 & 0.0902 & 0.1654 \end{bmatrix}$$

The ground truth (label) matrix is $d = [8,6,5,2,0.5]^T$. Hence, the weighs should be $w = X^{\dagger}d = [8.87, -1.64]^T$.

The regression equation is $y = 8.87 - 1.64x$.

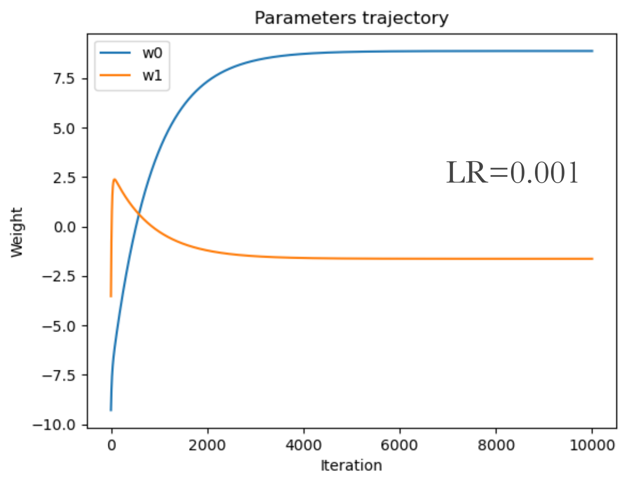
b). The initial weighs are chosen uniformly randomly between -10 and 10, and the learning rate is 0.02. The weights of learning which lasts for 100 epochs is $w = [8.44, -1.52]$. The figures below illustrate the fitting result and learning trajectory of LMS algorithm and the LLS.

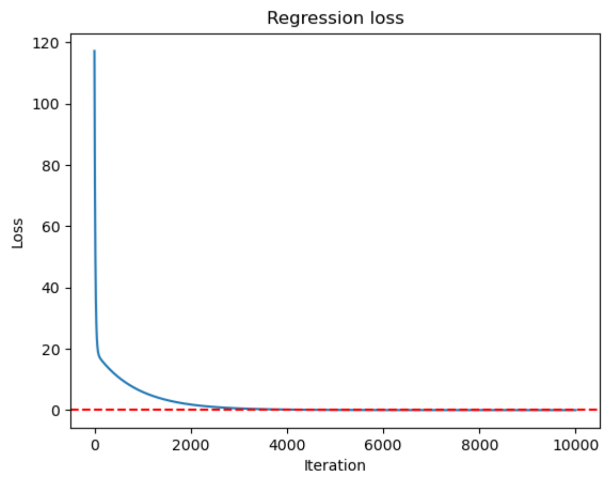


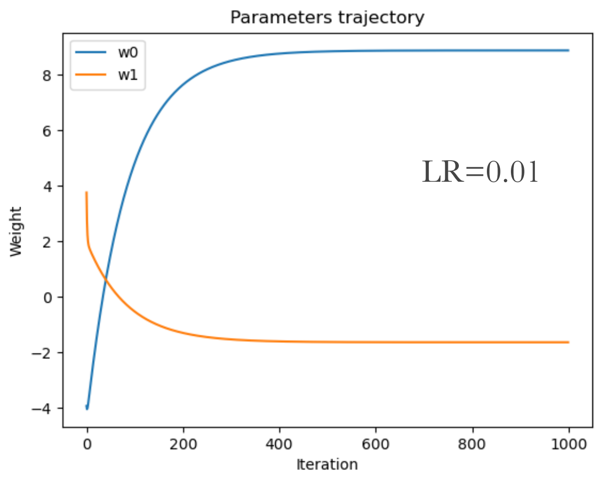


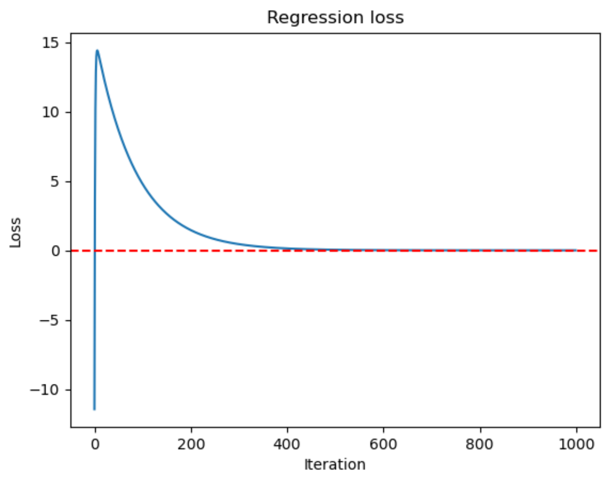
c). The LLS can derive the result directly using the least square method, which is suitable for small/medium scale data fitting. But the calculation expense will grow rapidly with scale of the problem, in other words, the number of weights and sampled points. The dimension of the matrix will be extremely large. In comparison, the LMS can solve the problem step by step. The accuracy (loss) of the results is related to learning epochs, longer it trains, more accurate the result is. Therefore, it is a powerful tool in solving problems with high dimensions, which is intractable to calculate directly.

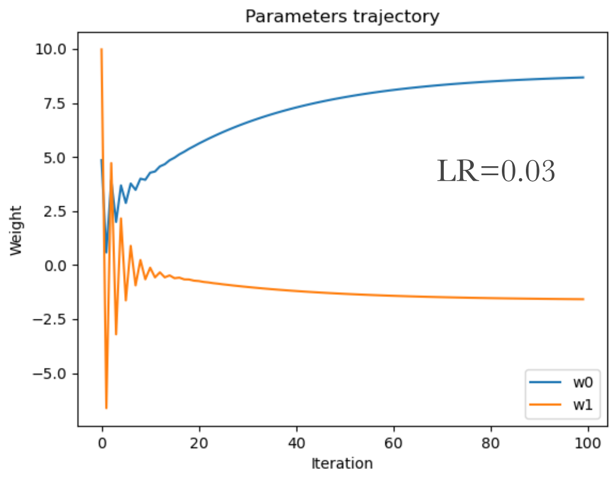
d). To see the influence of learning rate, we set $\eta = 0.001, 0.01, 0.03, 0.035, 0.04, 0.1$, respectively. Note that we will adjust the number of epoch accordingly, to make it fully trained.

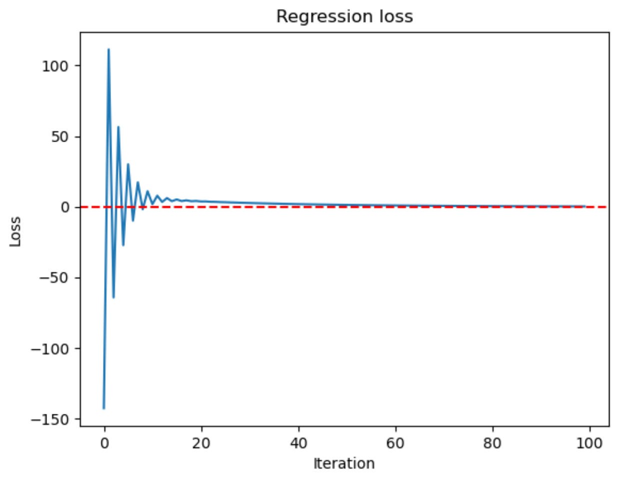

Parameters trajectory
w0
w1
LR=0.001
Weight
Iteration
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
0
2000
4000
6000
8000
10000

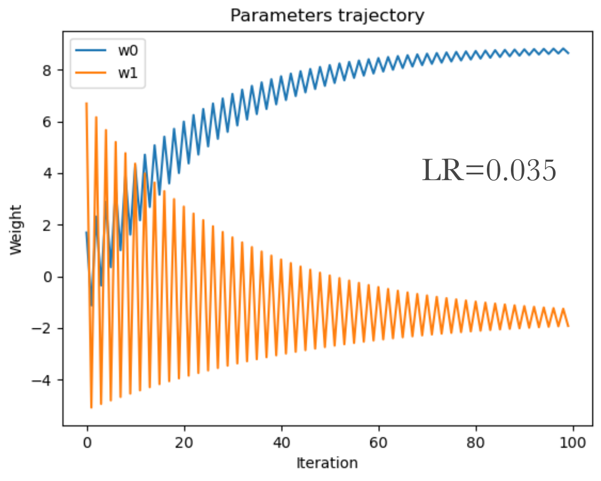

Regression loss
Loss
Iteration
120
100
80
60
40
20
0
0
2000
4000
6000
8000
10000

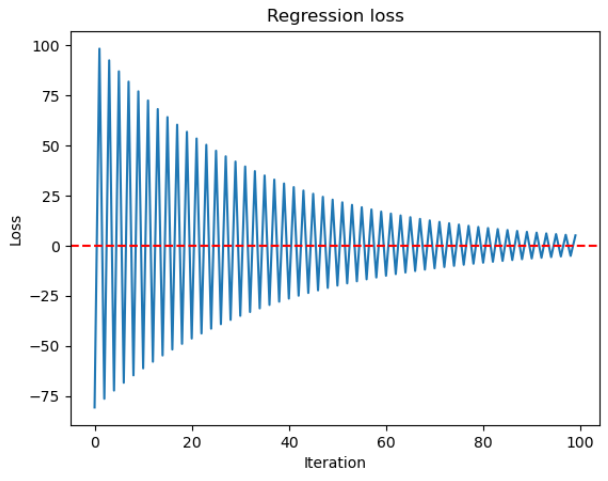

Parameters trajectory
w0
w1
LR=0.01
Weight
Iteration
8
6
4
2
0
−2
−4
0
200
400
600
800
1000

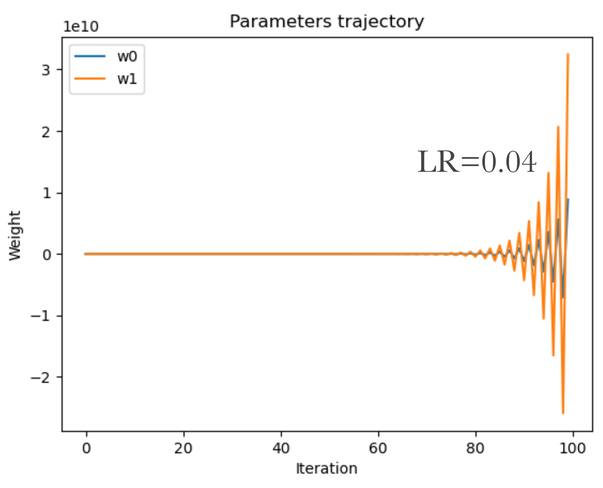

Regression loss
Loss
Iteration
15
10
5
0
−5
−10
0
200
400
600
800
1000

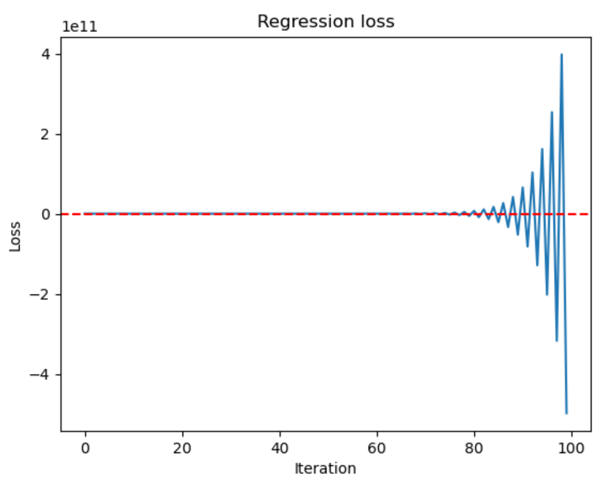

Parameters trajectory
LR=0.03
w0
w1
Weight
Iteration
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
0
20
40
60
80
100

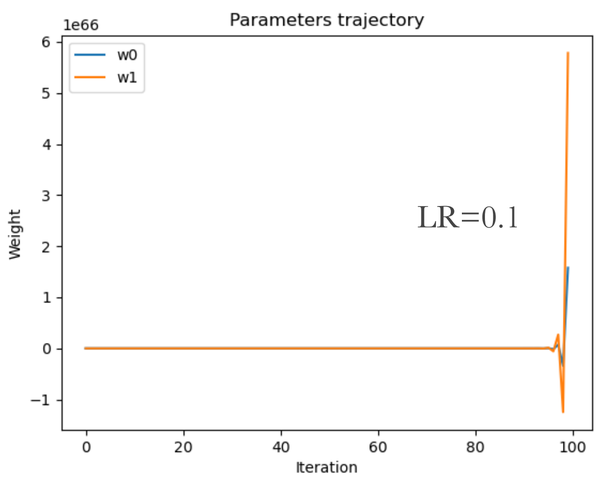

Regression loss
Loss
Iteration
100
50
0
−50
−100
−150
0
20
40
60
80
100

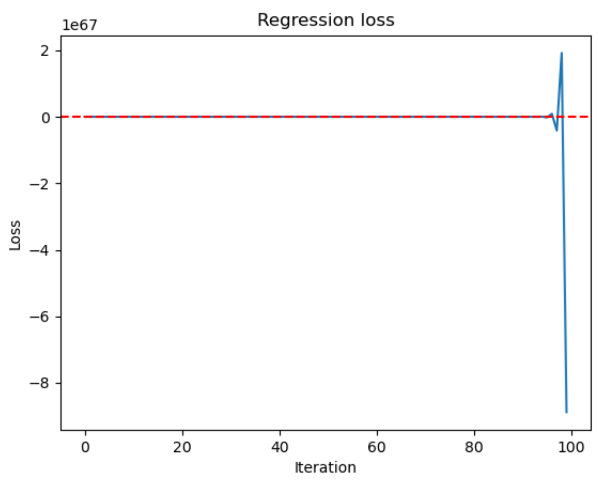

Parameters trajectory
w0
w1
LR=0.035
Weight
Iteration
8
6
4
2
0
−2
−4
0
20
40
60
80
100


Regression loss
Loss
Iteration
100
75
50
25
0
−25
−50
−75
0
20
40
60
80
100









The figures on the left show the parameters trajectory, and figures on the right show the relationship between the loss and learning epoch. It shows that if we tune the learning rate smaller, we have to increase the number of epoch to make it fully trained. It also shows that when learning rate is around 0.035, the convergence will behave like a zig-zag function, which tells that the step size is a bit higher than optimum. When learning rate is 0.4, the function will diverge, which is caused by the exorbitant step size. The adjustment of weights will overshoot, and makes the next iteration even away from the minima, finally result in explosion. In linear regression problems, given data points, there exist a threshold learning rate η_0, that for every $\eta \geq \eta_0$ the function will diverge.

Q5.

Similar proof can be applied to leaky LMS algorithm $E(n) = \frac{1}{2}e^2(n) + \frac{1}{2}\lambda\|w(n)\|^2$.

In LMS, $g_{LMS}(n) = \left(\frac{\partial E_{LMS}(w)}{\partial w(n)}\right)^T = -\,e(n)x(n)$, so

$$g(n) = g_{LMS}(n) + \left(\frac{\partial E_{leaky}(w)}{\partial w}\right)^T = -\,e(n)x(n) + \left(\frac{\partial(\frac{1}{2}\lambda w^T(n)w(n))}{\partial w}\right)^T = -\,e(n)x(n) + \lambda w(n)$$

Applying gradient descent method, gives

$w(n+1) = w(n) - \eta g(n) = w(n) - \eta(-e(n)x(n) + \lambda w(n)) = (1-\eta\lambda)w(n) + \eta e(n)x(n)$ QED