EE5904- The Solution of the Homework 1

Q1.

The forward procedure of a perceptron is as $v = w^\top x$ and $\varphi(v) = y$.

1). Let the threshold $\epsilon \in (0,1)$,

$$\varphi(v) = \epsilon$$
$$e^{-v} = \frac{1-\epsilon}{\epsilon}$$
$$v = -ln(\frac{1-\epsilon}{\epsilon})$$

so, we have

$$w^\top x + b = -ln(\frac{1-\epsilon}{\epsilon})$$

Thus, the boundary is a hyper-plane.

2). Let the threshold $\epsilon \in (0,1)$,

$$\varphi(v) = \epsilon$$
$$v - m = \pm\sqrt{-2ln(\epsilon)}$$

so, we have

$$w^\top x + b = m \pm \sqrt{-2ln(\epsilon)}$$

Thus, the decision boundary is not a hyper-plane.

3).

$$\varphi(v) = \frac{v}{1+v} \in [0,1), \quad v \geq 0$$
$$\varphi(v) = \frac{v}{1+(-v)} \in (-1,0), \quad v < 0$$

It is easy to know that $\varphi(v)$ is monotonic(non-deceasing). For any threshold $\epsilon \in (-1,1)$, there is an unique solution $v = \varphi^{-1}(\epsilon)$. Thus, $w^\top x = \varphi^{-1}(\epsilon)$, the decision boundary is a hyper-plane.

Q2

To solve this question, the self-contradiction methold is a good choice. Firstly, we make an assumption that the linear separation boundary exsits, and the weight vector is $[w_1, w_2]$.

Considering one case: if $wx^T > 0$, the class label will be 1, otherwise, the class label is 0. According to XOR truth table,

$$[w_1, w_2][1, 0]^T > 0$$
$$[w_1, w_2][0, 1]^T > 0$$
$$[w_1, w_2][1, 1]^T < 0$$
$$[w_1, w_2][0, 0]^T < 0$$

However, $[1, 1] = [1, 0] + [0, 1]$, then $[w_1, w_2][1, 1]^T = [w_1, w_2][1, 0]^T + [w_1, w_2][0, 1]^T > 0$, It is self-contradictory.

Considering the other case: if $wx^T < 0$, the class label will be 1, otherwise, the class label is 0. According to XOR truth table,

$$[w_1, w_2][1, 0]^T < 0$$
$$[w_1, w_2][0, 1]^T < 0$$
$$[w_1, w_2][1, 1]^T > 0$$
$$[w_1, w_2][0, 0]^T > 0$$

On the other hand, $[w_1, w_2][1, 1]^T = [w_1, w_2][1, 0]^T + [w_1, w_2][0, 1]^T < 0$, it is self-contradictory. Therefore, the weight vector $[w_1, w_2]$ for linear separation does not exsit.

As a result, the XOR problem is not linearly separable.

Q3

3.1 a)

For AND, as shown in following figure, the 'o' means $y = 0$ and the '*' means $y = 1$. A line of function '$x1 + x2 = 1.5$' separate two set of points. The weights vector is $[1, 1]$, and the bias is '-1.5'. If $x1 + x2 - 1.5 > 0$, the value of y is 1, otherwise, the value of y is 0.

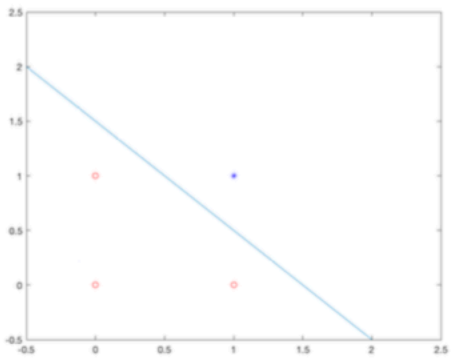


Figure 2: The classification for AND problem

For OR, with same method above, the two set of points can be separated by a line of function '$x_1 + x_2 = 0.5$'. Then the weight vector is still $[1, 1]$ and the bias is '-0.'5.

For COMPLEMENT, it is very easy to a polynomial of '$1 - x$', the weight is '-1', and the bias is '1'.

For NAND, obviously the line separating sets of AND still work in this situation. Just the decison strategy based on value 'y' should be reversed.

3.2 b)

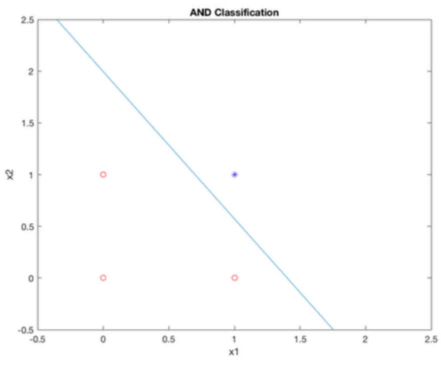
Perceptron Learning Algorithm solves linear separation effetively. During each iteration, the weight vector is updated as the following:

$$w(n + 1) = w(n) + \eta * e(n) * x(n)$$

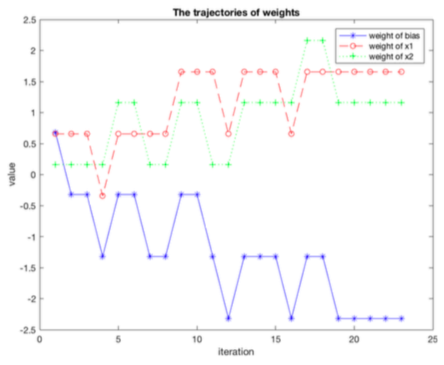
$$e(n) = d(n) - y(n)$$

where $\eta > 0$ is learning rate

AND: Firstly, the weight vector is generated randomly as [0.679702676853675 0.655098003973841 0.162611735194631], the classification results are shown in the following figures. To avoid too many iterations, double check is set: numbers of iteration and 5 non-error in consequence. To compare the effect of different learning rate, the code is run with same initial weight vector and different learning rate - $\eta = 1$ and $\eta = 0.5$. In result, the algorithm with leaning rate 1 needs 23 iterations, while the algorithm with learning rate 0.5 needs 27 iterations.

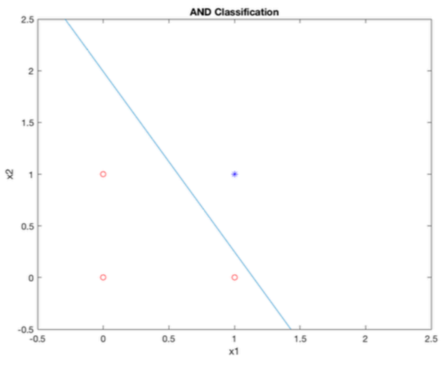


(a) Classification of AND

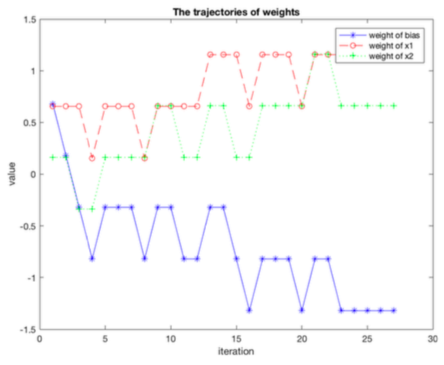


(b) The trajectories of weights

Figure 3: Results of AND problem with $\eta = 1$



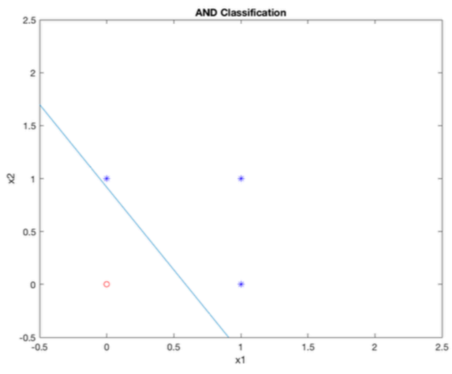
(a) Classification of AND



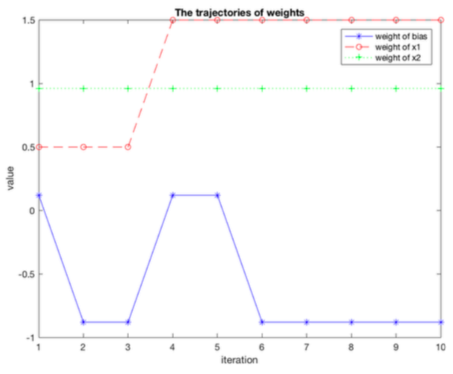
(b) The trajectories of weights

Figure 4: Results of AND problem with $\eta = 0.5$

OR: Firstly, the original weight vector is generated randomly as [0.118997681558377 0.498364051982143 0.959743958516081], classification results are shown in the following figures. The algorithm with leaning rate 0.5 needs 6 iterations, while the algorithm with learning rate 1 needs 10 iterations.

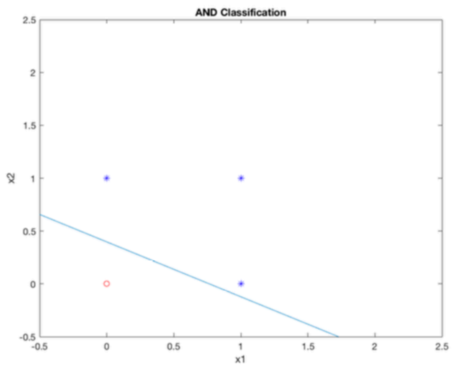


(a) Classification of OR

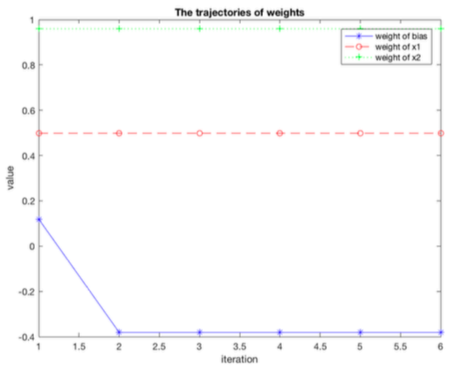


(b) The trajectories of weights

Figure 5: Results of OR problem with $\eta = 1$



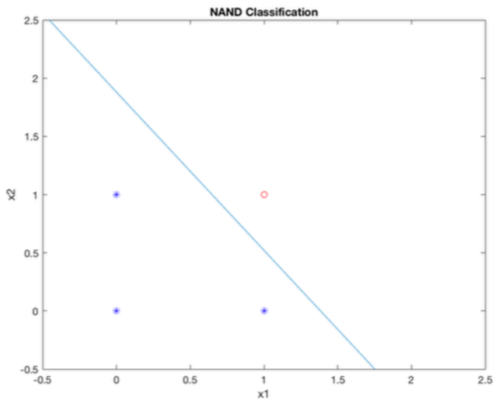
(a) Classification of OR



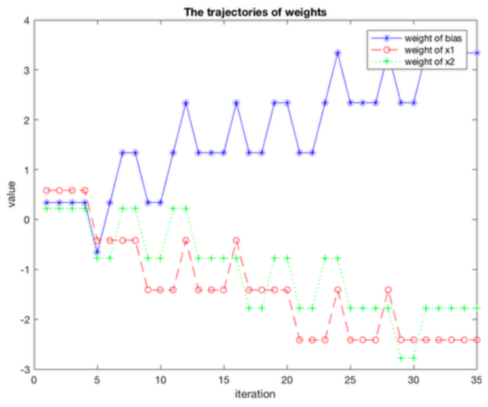
(b) The trajectories of weights

Figure 6: Results of OR problem with $\eta = 0.5$

NAND: The original weight vector is generated as [0.340385726666133 0.585267750979777 0.223811939491137], all the results are shown as following figures. The algorithm with learning rate 1 needs 35 iterations, while the algorithm with learning rate 0.5 needs 27 iterations.

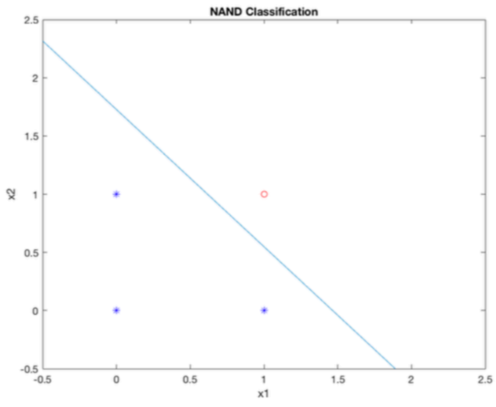


(a) Classification of NAND

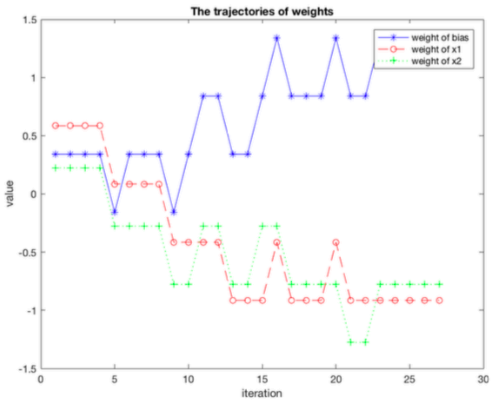


(b) The trajectories of weights

Figure 7: Results of NAND problem with $\eta = 1$



(a) Classification of NAND



(b) The trajectories of weights

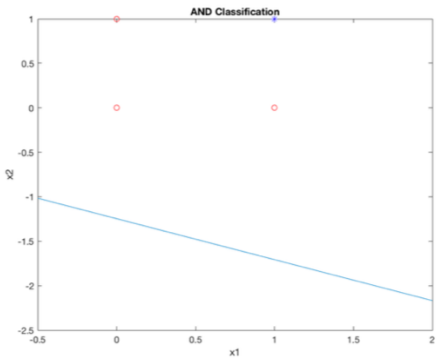
Figure 8: Results of NAND problem with $\eta = 0.5$

For COMPLEMENT problem, it is easier than three above. With same algorithm and initial weight vector [0.814723686393179 0.905791937075619], the results come out, from which algorithm with learning rate 1 needs 8 iterations and the other needs 10 iterations.

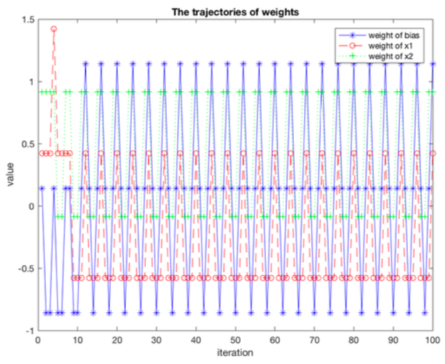
In conclusion, when the initial weight vector is really far away from the suitable one, algorithm with bigger learning rate can converge faster. However, if the initial weight vector is near from the suitable one, bigger learning rate may lead the update weight jump over the suitable one and turn back for searching. In that case, bigger learning rate may need more iterations than smaller.

3.3 c)

The results are shown in the following figures, the perceptron can not do classification for XOR problem. The algorithm will not converge. When the iterations end, the weight vector generated still can not separate two sets. As shown in figure9b, the weight vector will vibrate between some specific values and will not converge.



(a) Classification of XOR



(b) The trajectories of weights

Figure 9: Results of XOR problem with $\eta = 0.5$

Q4

4.1 a)

In this problem, input values are x and bias 1. The respective weights are weight of x and bias weight b. Firstly, I use standard linear least-squares method to solve this question. LLS aims to find the least value of $E(w) = \sum e(w)^2 = \sum(d - Xw)^2$, where X is regression matrix. The first step is to calculate regression matrix,

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 0.8 \\ 1 & 1.6 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{bmatrix}$$

$$d = \begin{bmatrix} 0.5 \\ 1 \\ 4 \\ 5 \\ 6 \\ 8 \end{bmatrix}$$

$$w = (X^\top X)^{-1} X^\top d = [0.5554, 1.47]^\top$$

Thus, $y = 0.5554 + 1.47x$

4.2 b)

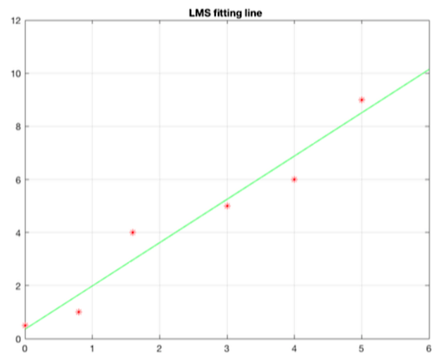
Different from LLS method, LMS pays more attention to instaneous mean error. The error cost function at step n is $E(w) = \frac{1}{2}e(n)^2$. Differentiate the matrix function, and we get that the gradient of cost function is

$$g(n) = \frac{\partial E(w)}{\partial w(n)} = -e(n)x(n) = d(n) - x(n)^T w(n)$$

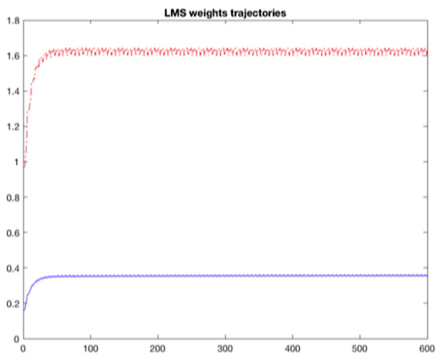
Applying steepest descent method, we have

$$w(n+1) = w(n) + \eta * e(n) * x(n)$$

For this question, iteration steps are fixed at 100 epochs times 6. And the results are shown in the follwoing figure. Obviously, the weights converge and the last value is [0.358, 1.631].



(a) LMS fitting line



(b) The trajectories of weights

Figure 10: Results of LMS with $\eta = 0.5$ and 100 epochs

4.3 c)

Comparing two method, the LLS cares much about the global optimal solution. While the LMS keeps minimizing the loss of a mini-batch. Thus, the trajectories of the learned weights may oscillate. The performance of LMS method significantly depends on the learning rates. Too large learning rates may leads to unstable training.

4.4 d)

To compare the results of different learning rates, we choose same original weight vector randomly as [0.158 ; 0.971] and the iteration steps are still 100epochs times 6. The fitting line and total error squares $\sum e(n)^2$ are shown in the following figure and table.

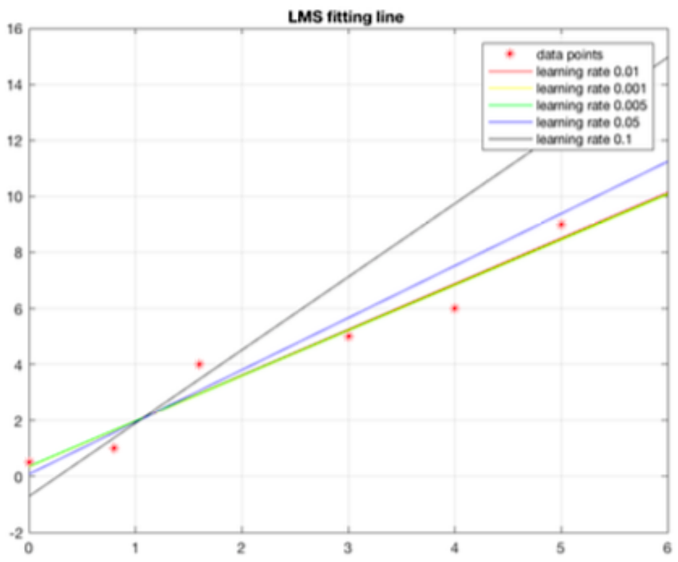


Figure 12: Fitting lines of different learing rates in 100 epochs

learning rate	0.01	0.001	0.005	0.05	0.1
error	2.603	2.595	2.595	4.291	31.615

As shown in the experiments, small learning rate may lead to good performance. While the big learning rate may result in long-step jump and large error. So, when doing a project, it is better to try different learning rates and apply the rate-decaying technique to adjust the learning rate dynamically.

Q 5.

Firstly, let

$$R = \begin{bmatrix} r_1^2 & & & & & \\ & r_2^2 & & & & \\ & & \cdot & & & \\ & & & \cdot & & \\ & & & & \cdot & \\ & & & & & r_n^2 \end{bmatrix}$$

Let $X = [x(1), x(2), ..., x(n)]^\top$ where $x(i)$ is a data sample of dimension m, so the size of X is (n, m). Let $w = [w_1, w_2, ..., w_m]^\top$,then $e = d - Xw$.

Then, we have

$$J = \frac{1}{2}e^\top Re + \frac{1}{2}\lambda w^\top w$$

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

$$(d - Xw)^\top R(-X) + \lambda w^\top = 0$$

$$w^\top X^\top RX - d^\top RX + \lambda w^\top = 0$$

$$w^\top (X^\top RX + \lambda I) = d^\top RX$$

$$w^\top = d^\top RX(X^\top RX + \lambda I)^{-1}$$