EE5904/ME5404 Neural Networks: Homework #1

Q1 Solution (10 Marks).

According to the signal-flow graph of the perceptron (ignoring the subscript k), the induced local field v can be written as

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

Consider the following three choices of activation function:

1) Quadratic function $\varphi(v) = (v-a)^2 + c$:
   As the observation vector $\boldsymbol{x} = [x_1, x_2, \ldots, x_m]^T$ is classified as C_1 (resp. C_2) when $y = \varphi(v) > \xi$ (resp. $y = \varphi(v) \leqslant \xi$), the decision boundary is
   $$\varphi(v) = \xi$$
   $$v = \pm\sqrt{\xi - c} + a$$
   $$x_1 w_1 + x_2 w_2 + \cdots . + x_m w_m = \hat{c}$$
   where $c \triangleq \pm\sqrt{\xi - c} + a - b$ is **not** a constant when $\xi > c$. The resulting decision boundary is therefore **not** a hyper-plane.
2) Hyperbolic tangent function $\varphi(v) = \frac{1-e^{-v}}{1+e^{-v}}$:
   Similarly, the decision boundary in this case can be obtained by $\varphi(v) = \xi$, i.e.
   $$\frac{1-e^{-v}}{1+e^{-v}} = \xi$$
   $$v = \ln\left(\frac{1+\xi}{1-\xi}\right)$$
   $$x_1 w_1 + x_2 w_2 + \cdots . + x_m w_m = c$$
   where $c \triangleq \ln\left(\frac{1+\xi}{1-\xi}\right) - b$ is a constant; and thus, the resulting decision boundary is a hyper-plane.
3) Bell-shaped Gaussian function $\varphi(v) = e^{-\frac{(v-m)^2}{2}}$:
   Also, the decision boundary in this case is $\varphi(v) = \xi$, i.e.
   $$e^{-\frac{(v-m)^2}{2}} = \xi$$
   $$v = \pm\sqrt{-2\ln\xi} + m$$
   $$x_1 w_1 + x_2 w_2 + \cdots . + x_m w_m = c$$
   where $c \triangleq \pm\sqrt{-2\ln\xi} + m - b$ is **not** a constant when $0 < \xi < 1$. The resulting decision boundary is therefore **not** a hyper-plane.

Q2 Solution (10 Marks).

Proof by contradiction: assume the XOR problem is linearly separable, then there must exist a weight vector $\boldsymbol{w}$ such that the perceptron $y = \varphi(v) = \begin{cases} 0, & v \leq 0 \\ 1, & v > 0 \end{cases}$ where $v = w_1 x_1 + w_2 x_2 + b$ can correctly perform the classification. In other words, the following four inequalities can be guaranteed simultaneously.

$$x_1 = 0, x_2 = 0, y = 0: \quad b \leq 0 \tag{1}$$
$$x_1 = 1, x_2 = 0, y = 1: \quad w_1 + b > 0 \tag{2}$$
$$x_1 = 0, x_2 = 1, y = 1: \quad w_2 + b > 0 \tag{3}$$
$$x_1 = 1, x_2 = 1, y = 0: \quad w_1 + w_2 + b \leq 0 \tag{4}$$

Combining (1) and (4), one can get $w_1 + w_2 + 2b \leqslant 0$ which is contradicted with the inequality obtained by (2) + (3): $w_1 + w_2 + 2b > 0$. Thus, formula (1)-(4) cannot be guaranteed simultaneously and the assumption is wrong, i.e. the XOR problem is not linearly separable.

Q3 Solution (10 Marks).

a) The off-line calculations can be done by plotting the truth table values and a decision boundary to separate the two classes. The implementation functions AND, OR, COMPLEMENT, and NAND with selection of weights by off-line calculations are demonstrated as follows: (note $\varphi(\cdot)$ is the hard limiter).

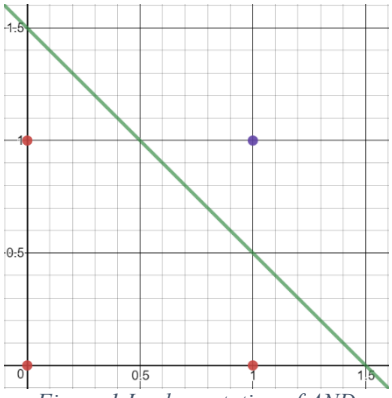


Figure 1 Implementation of AND

$\varphi(-1.5 + x_1 + x_2), \boldsymbol{w} = [-1.5,1,1\ ]^T$

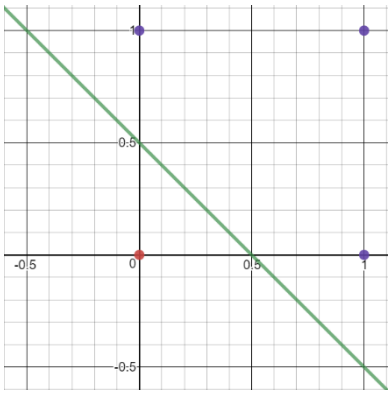


Figure 2 Implementation of OR

$\varphi(-0.5 + x_1 + x_2), \boldsymbol{w} = [-0.5,1,1\ ]^T$

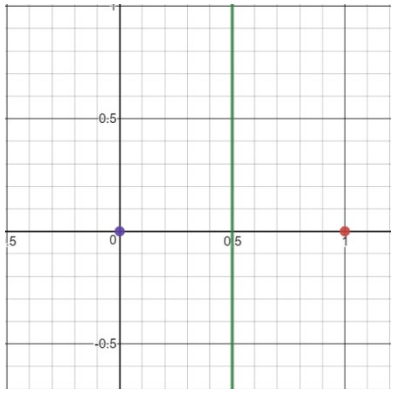


Figure 3 Implementation of COMPLEMENT

$\varphi(0.5 - x_1), \boldsymbol{w} = [0.5, -1\ ]^T$

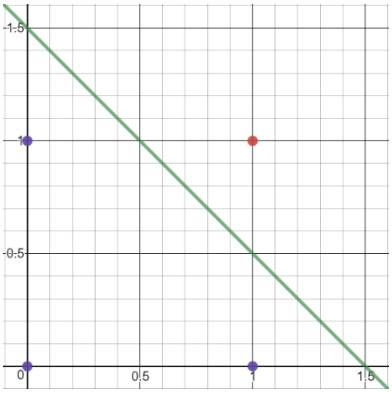


Figure 4 Implementation of NAND

$\varphi(1.5 - x_1 - x_2), \boldsymbol{w} = [1.5, -1, -1\ ]^T$

b) The learning procedure is implemented by the following code based on MATLAB:

```
flag = 1;
    while flag
    flag = 0;
        for i = length(d)
        y = (X(i, :) * w) >= 0;          % y = 0 is defined to be positive
        e = d(i) - y;
        if e ~= 0
            w = w + e * X(i, :)';        % update weight vector
            flag = 1;
        end
    end
end
```

Function AND: Initialize the learning procedure with

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad d = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad w_0 = \begin{bmatrix} 0.2 \\ 0.5 \\ 0.8 \end{bmatrix}$$

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

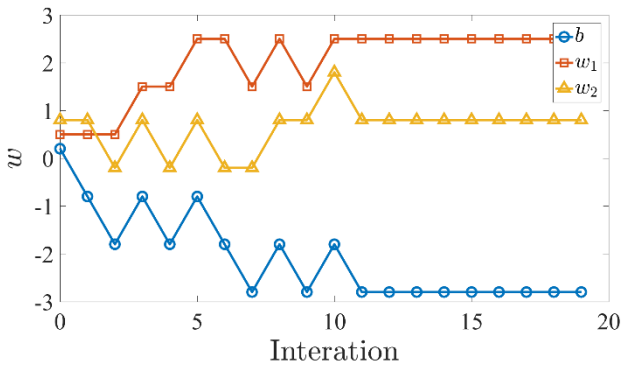


Figure 5 Trajectory of weights

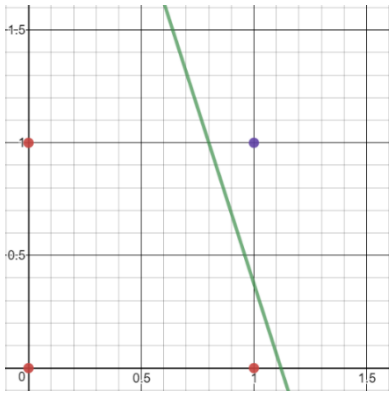


Figure 6 Implementation of AND

$$\varphi(-2.8 + 2.5x_1 + 0.8x_2), \boldsymbol{w} = [-2.8,2.5,0.8\ ]^T$$

Function OR: Initialize the learning procedure with

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad d = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \qquad w_0 = \begin{bmatrix} 0.2 \\ 0.4 \\ 0.6 \end{bmatrix}$$

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

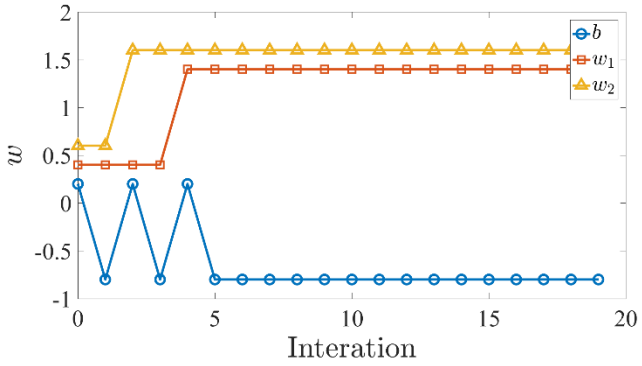


Figure 7 Trajectory of weights

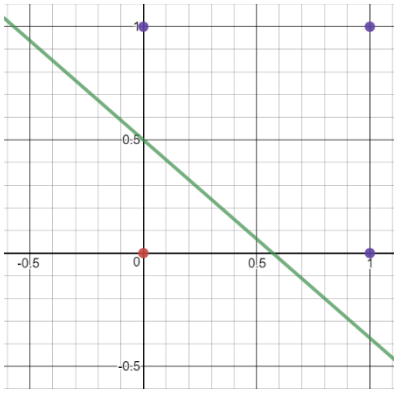


Figure 8 Implementation of OR

$$\varphi(-0.8 + 1.4x_1 + 1.6x_2), w = [-0.8,1.4,1.6\ ]^T$$

Function COMPLEMENT: Initialize the learning procedure with

$$X = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \qquad d = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad w_0 = \begin{bmatrix} 0.3 \\ 0.6 \end{bmatrix}$$

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

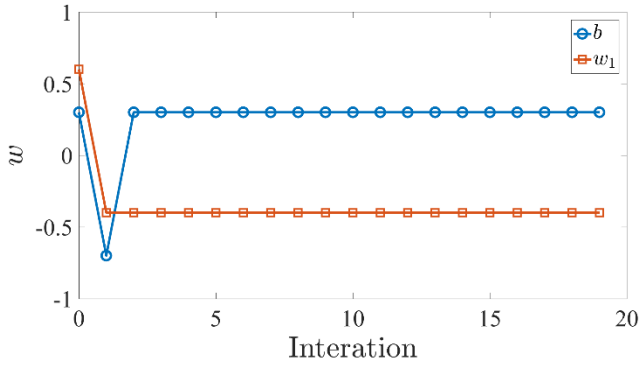


Figure 9 Trajectory of weights

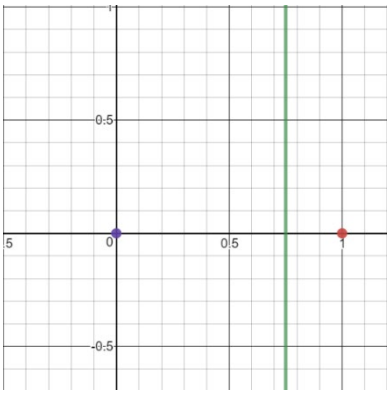


Figure 10 Implementation of COMPLEMENT

$$\varphi(0.3 - 0.4x_1), w = [0.3, -0.4\ ]^T$$

Function NAND: Initialize the learning procedure with

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad d = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} \qquad w_0 = -\begin{bmatrix} 0.4 \\ 0.6 \\ 0.8 \end{bmatrix}$$

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

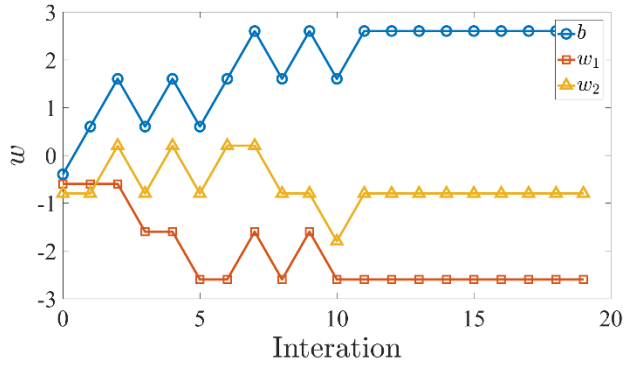


Figure 11 Trajectory of weights

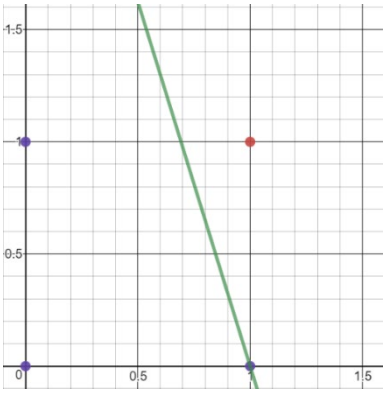
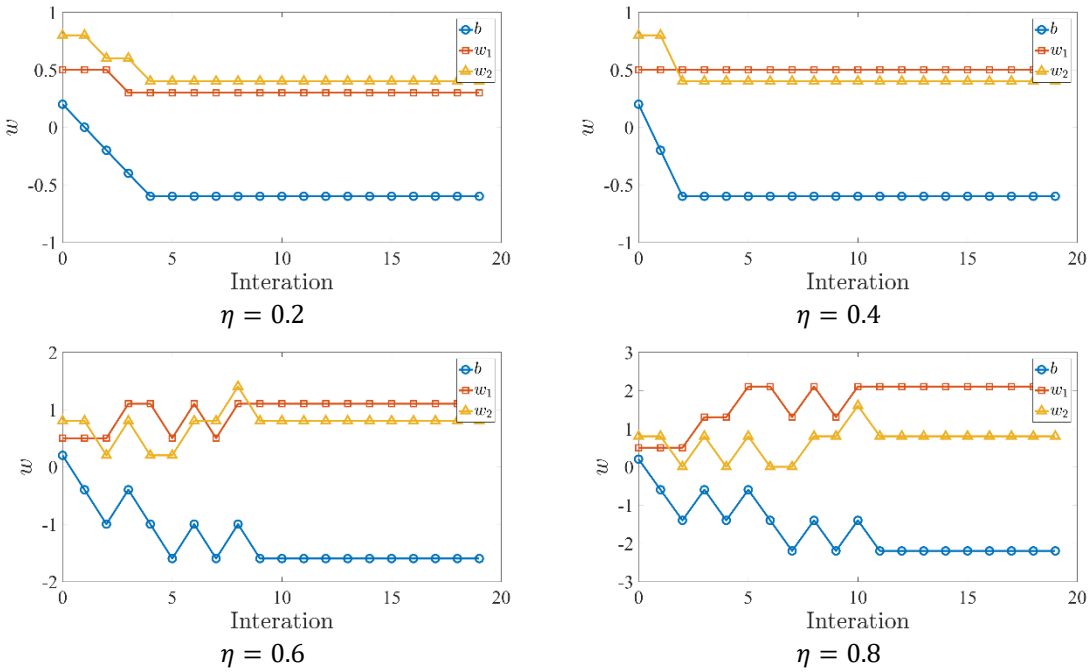


Figure 12 Implementation of NAND

$$\varphi(2.6 - 2.6x_1 - 0.8x_2), w = [2.6, -2.6, -0.8\ ]^T$$

Comparing Fig. 6,8,10,12 with Fig. 1-4, it can be found that all the decision boundaries could implement the concerned logic functions correctly, even though the weight vectors obtained through off-line calculation and learning procedure are different.

Take function AND as example, the weight vector could converge under various learning rate. In this special case, the weight vector converges fastest when $\eta = 0.4$.



c) When apply perception learning algorithm to the EXCLUSIVE OR function, the algorithm could not stop once started and the weight vector keeps oscillating as shown in Fig. 13. This is because the points that represents the EXCLUSIVE OR function are not linear separable and at least two straight lines are required to divide them. Hence, the algorithm could not converge in finite time and no correct decision boundary could be obtained.

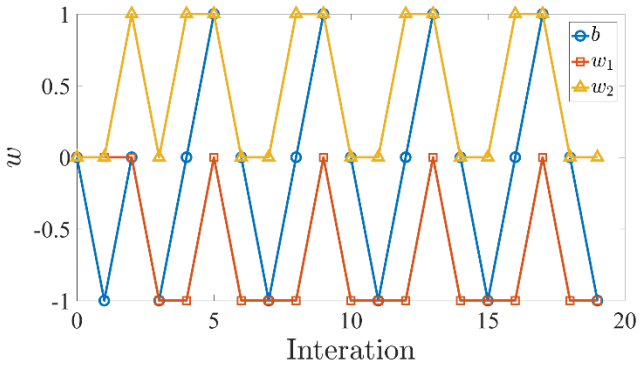


Figure 13 Trajectory of weights

Q4 Solution (10 Marks).

a) First, construct the cost function:

$$E = \sum_{i=1}^{n} \frac{1}{2} e_i^2 = \frac{1}{2} \boldsymbol{e}^T \boldsymbol{e}$$

where the error vector is defined as $\boldsymbol{e} = \boldsymbol{d} - X\boldsymbol{w}$ with the desired response vector $\boldsymbol{d}$, sample matrix X and weight vector $\boldsymbol{w}$ denoted as

$$\boldsymbol{d} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} \qquad X = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_n^T \end{bmatrix} \qquad \boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}$$

Then, it follows

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

When $\nabla_{\boldsymbol{w}} E = 0$, we have $\boldsymbol{w} = (X^T X)^{-1} X^T \boldsymbol{d}$. Now, instantiate the sample matrix and desired response vector as follow:

$$X = \begin{bmatrix} 1 & 0.5 \\ 1 & 1.5 \\ 1 & 3.0 \\ 1 & 4.0 \\ 1 & 5.0 \end{bmatrix} \qquad \boldsymbol{d} = \begin{bmatrix} 8.0 \\ 6.0 \\ 5.0 \\ 2.0 \\ 0.5 \end{bmatrix}$$

According to the derived LLS formula, the closed-form weight vector can be calculated:

$$\boldsymbol{w} = \begin{bmatrix} b \\ w \end{bmatrix} = (X^T X)^{-1} X^T \boldsymbol{d} = \begin{bmatrix} +8.8684 \\ -1.6316 \end{bmatrix}$$

Thus, the optimal fitting function obtained by LLS is $y = -1.6316x + 8.8684$, which is plotted in Fig. 14.

b) The LMS algorithm is implemented by the following code based on MATLAB:

```
N = 100;
eta = 0.02;
for n = 0 : N
    for i = 1 : length(d)
```

```
        e = d(i) - X(i, :) * w;
        w = w + eta * e * X(i, :)';      % update weight vector
    end
end
```

and initialized with the aforementioned X, d and $w_0 = [4, 2]^T$. After 100 epochs, the weight vector becomes $w_{100} = [8.4378, -1.5555]^T$ and the fitting result is $y = -1.5555x + 8.4378$ as shown in Fig. 14. In addition, the trajectory of weight vector versus learning steps and the average of the cost $\frac{e^2}{2}$ versus epochs are illustrated in Fig. 15 and Fig. 16, respectively. It can be found from Fig. 15 and 16 that the weights and error do have a converging tendency; however, the weights will not converge to a certain value in finite time due the fact that the given sample points are not perfectly located on a straight line. Therefore, the error will always exist, and the algorithm will not converge to a constant value.

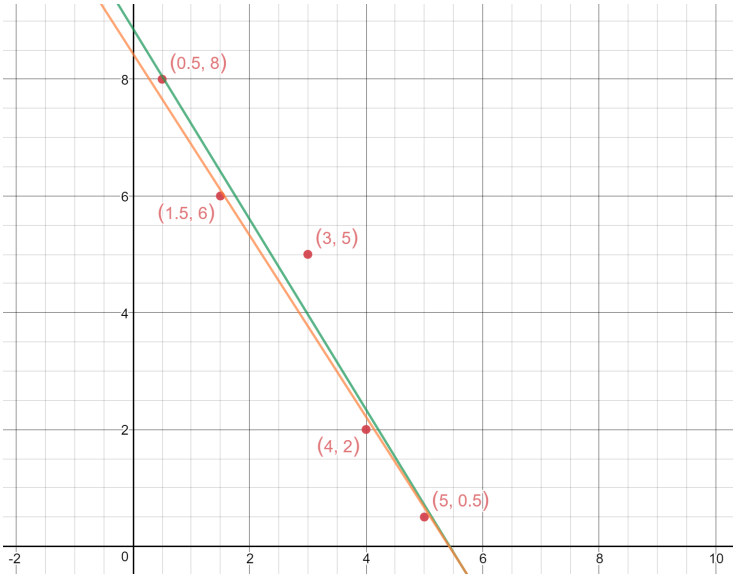


Figure 14 Fitting curve obtained by LLS (green) and LMS (orange)

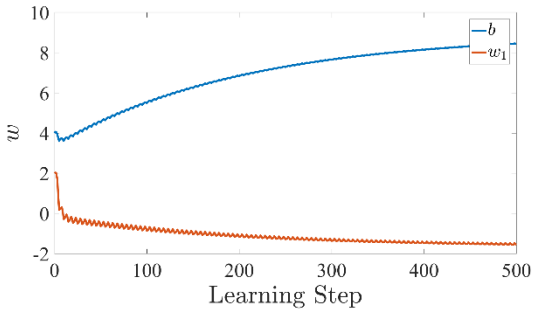


Figure 15 Trajectory of weights, $\eta = 0.02$

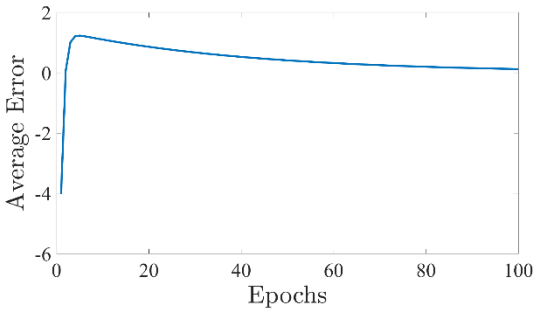


Figure 16 Average error versus epochs, $\eta = 0.02$

c) It is clearly from Fig. 14 that the fitting result determined by LMS algorithm is very close to that obtained from LLS method, which demonstrates the correctness of LMS algorithm.

Additionally, the LMS algorithm requires a long training procedure to converge, while the computation load would will be high when applying LLS method to high-dimensional data.

d) When the learning rate is set as $\eta = 0.2$, the weights of the fitting function as well as the estimation error will diverge as shown in Fig. 17 and Fig. 18. This is because the LMS algorithm is valid only when the Taylor approximation is effective, which is guaranteed by a fairly small learning rate (corresponding to a small neighborhood of the expansion point).

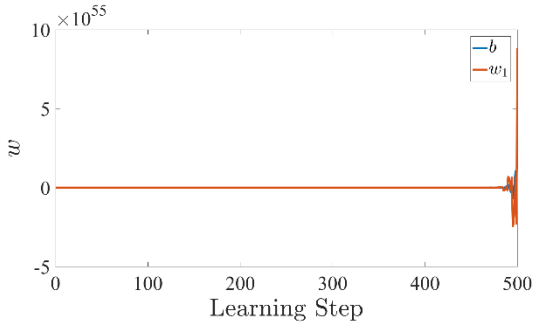


Figure 17 Trajectory of weights, $\eta = 0.2$

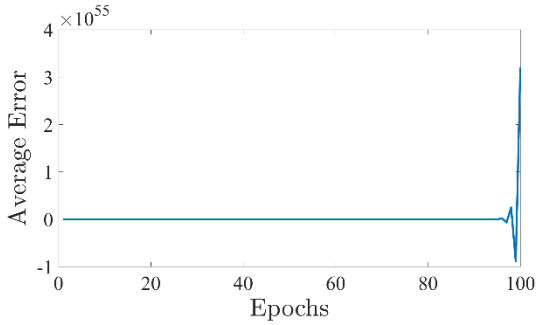


Figure 18 Average error versus epochs, $\eta = 0.2$

Q5 Solution (10 Marks).

The cost function is defined as the instantaneous cost at each step:

$$E(n) = \frac{1}{2}e^2(n) + \frac{\lambda}{2}\boldsymbol{w}^T(n)\boldsymbol{w}(n)$$

where $\boldsymbol{e}(n) = \boldsymbol{d}(n) - \boldsymbol{x}^T(n)\boldsymbol{w}(n)$ is the error signal at step n.

Take the derivative of E(n) with respect to w(n),

$$\frac{\partial \mathrm{E(n)}}{\partial w(n)} = e(n)\frac{\partial e(n)}{\partial w(n)} + \lambda w^T(n) = -x^T(n)e(n) + \lambda w^T(n)$$

Therefore, by gradient descent method, the update equation of $\boldsymbol{w}$ is

$$\begin{aligned} \boldsymbol{w}(n+1) &= \boldsymbol{w}(n) - \eta\nabla_{\boldsymbol{w}}E(n) \\ &= \boldsymbol{w}(n) - \eta\left(\frac{\partial E(n)}{\partial \boldsymbol{w}(n)}\right)^T \\ &= (1-\lambda\eta)\boldsymbol{w}(n) + \eta e(n)x(n) \end{aligned}$$

When $\lambda = 0$, we have $\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \eta e(n)x(n)$ that reduces to the LMS algorithm.