EE5904 Neural Networks: Homework #1

A0295779Y Xu Yimian

Q1:

Define the input vector as $\boldsymbol{x} = [+1, x_1, x_2, \dots, x_m]^T$, and weight vector as $\boldsymbol{w}(\mathrm{n}) = [b, w_1, w_2, \dots, w_m]^T$. Accordingly, the induced local field: $\boldsymbol{v} = \sum_{i=0}^{m} w_i x_i = \boldsymbol{w}^T\boldsymbol{x}$.

(1) $\varphi(\boldsymbol{v}) = \frac{1}{1+e^{-v}} = \xi, v = -\ln\left(\frac{1-\xi}{\xi}\right)$, define $C = -\ln\left(\frac{1-\xi}{\xi}\right)$ is a constant. The decision boundary is written as $\boldsymbol{w}^T\boldsymbol{x} = C$, and it is a hyper-plane.

(2) $\varphi(\boldsymbol{v}) = e^{-\frac{(v-m)^2}{2}} = \xi, v = m \pm \sqrt{-2ln\xi}$. The equation defines 2 hyper-planes. The decision boundary is not a hyper-plane.

(3) $\varphi(\boldsymbol{v}) = \frac{1}{1+|v|} = \xi, v = \frac{\xi}{1-|\xi|}$. The decision boundary is a hyper-plane.

Q2:

Assume XOR is linearly separable, which means there exists a decision boundary $w_1x_1 + w_2x_2 + b = 0$ that can successfully separate these points.

We also assume a threshold for classification:

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

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

From the Truth Table we have 4 inequalities:

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

This system of inequalities has no solution, which means XOR problem is not linearly separable.

Q3:

a) AND:

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

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

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

OR:

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

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

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

COMPLEMENT:

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

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

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

NAND:

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

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

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

b) When $\eta = 1.0$, the trajectories of the weights are shown below:

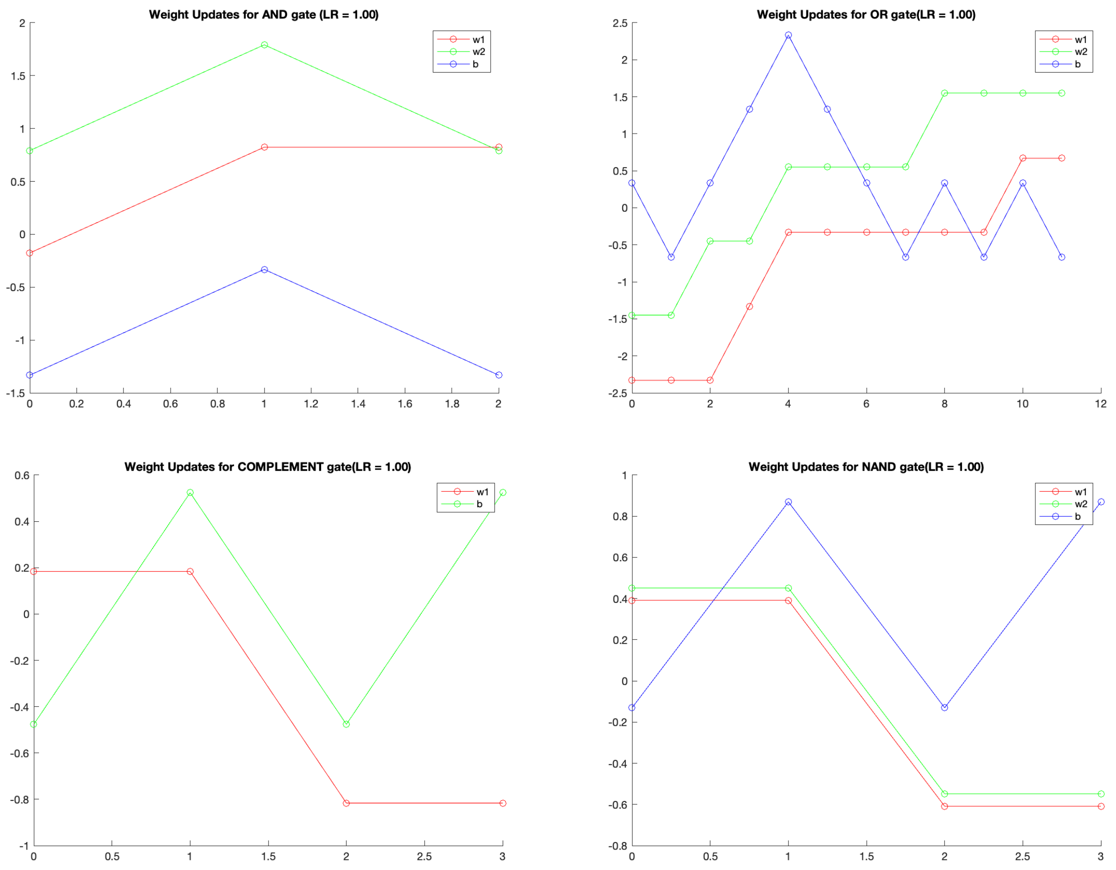
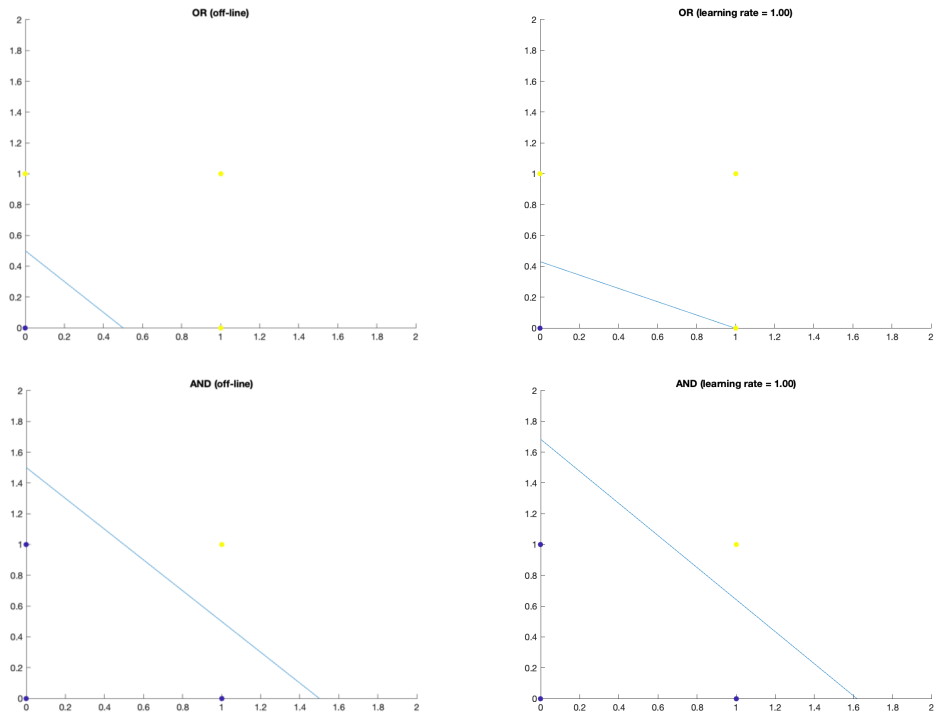


Figure 1 Weight update (LR=1.0)

Compare the result with those obtained in (a):



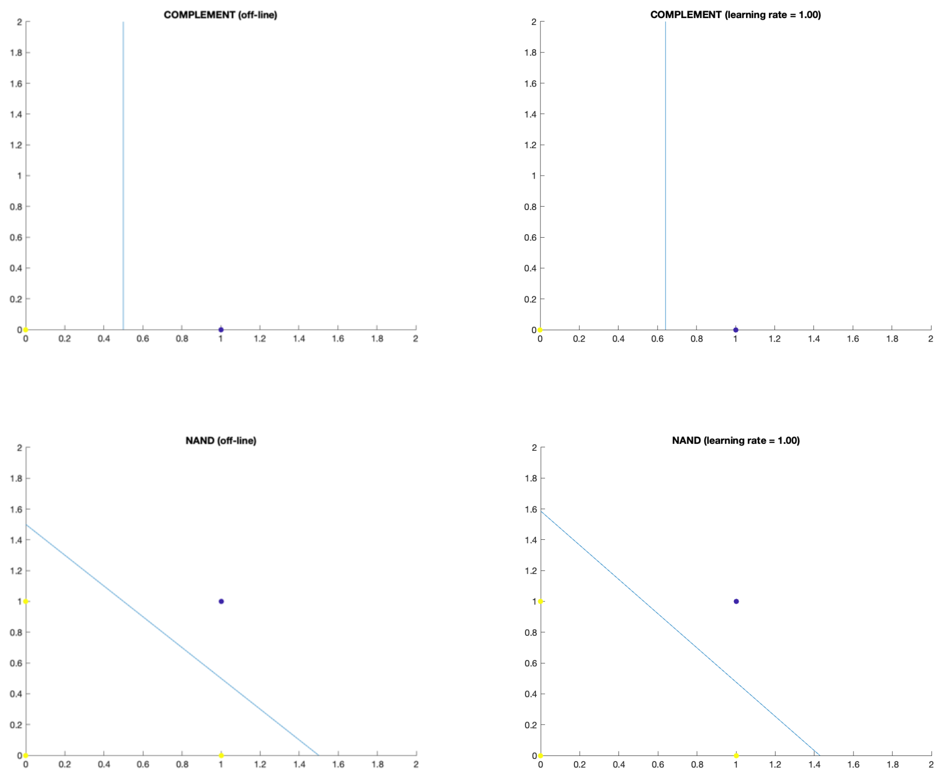


Figure 2 decision boundary comparison (off-line computation vs. learning LR=1.0)

When $\eta = 0.1$, the trajectories of the weights are shown below:

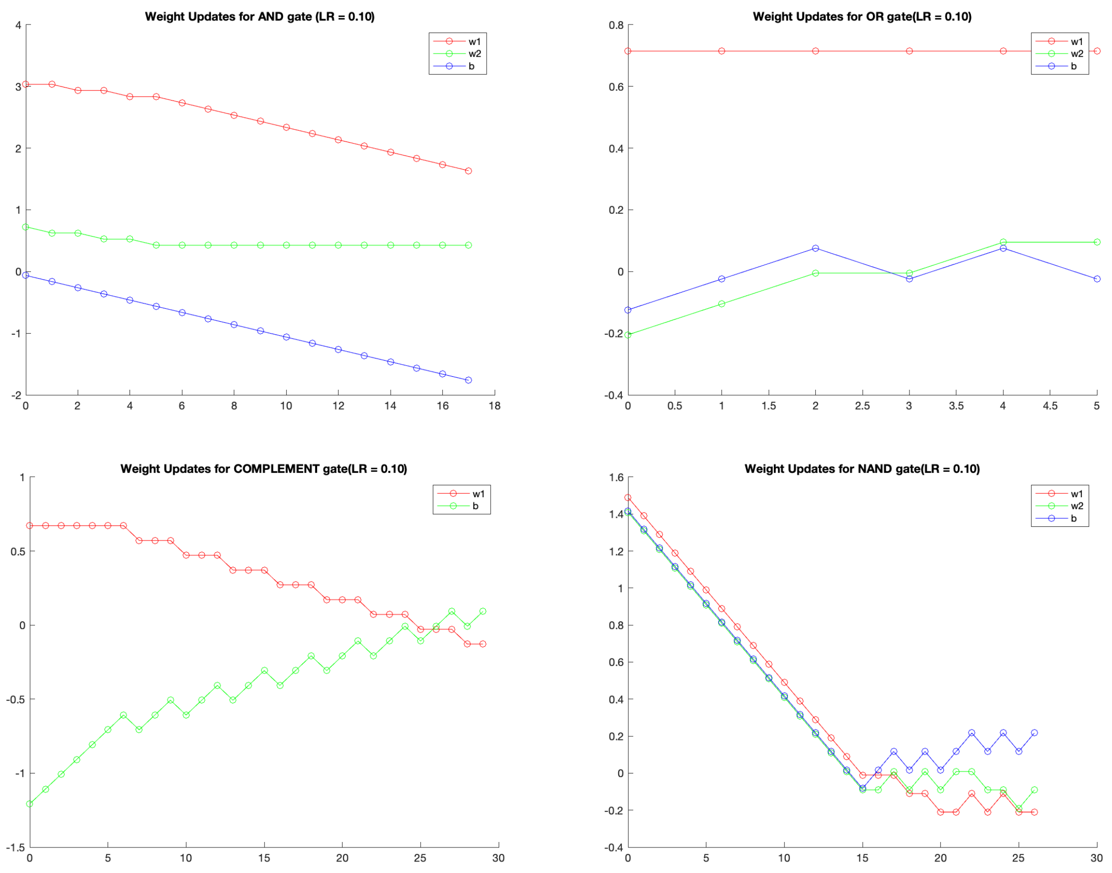


Figure 3 Weight update (LR=0.1)

When $\eta = 5$, the trajectories of the weights are shown below:

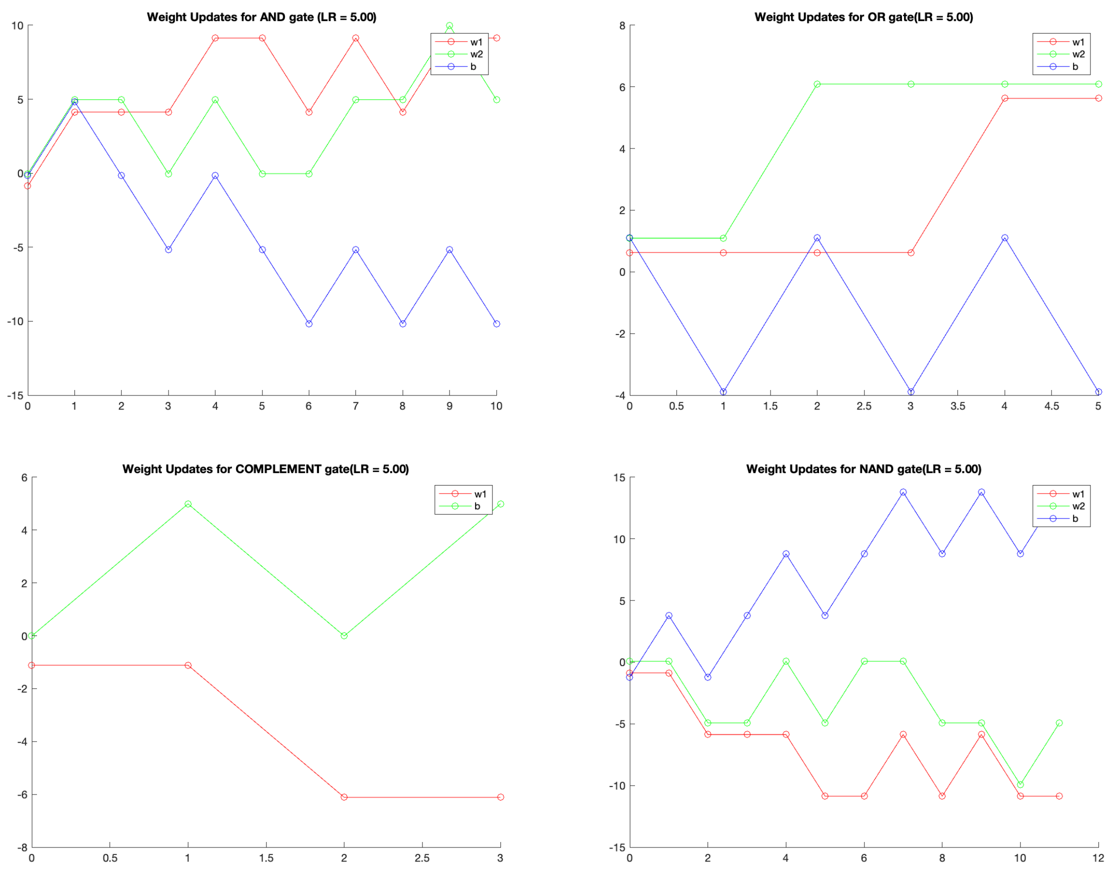


Figure 4 Weight update (LR=5.0)

From the comparison of the weight update procedures with different learning rates, we can see:

For AND Gate, the learning rate 1.0 is the best one. Too small or too big learning rates will increase the iteration times to converge. For OR Gate, the learning rate 0.1 and 5.0 have smaller converge iteration then 1.0, similar to COMPLEMENT Gate. For NAND Gate, the learning rate 1.0 is the best one, increasing or decreasing the learning rate causes more iteration to converge.

Analysis:

The perceptron convergence theorem guarantees convergence for linearly separable data, but the **initial weight vector** and **learning rate** affect the path to the solution.

If the initial weights are "closer" to the optimal separating hyperplane (e.g., aligned with the true weight vector), fewer iterations are typically needed. If the initial weights are poorly aligned (e.g., orthogonal or opposing the true direction), more iterations may be required. In terms of learning rate, larger steps per update can accelerate convergence if the initial weights are far from the solution, but in risk of overshooting near the decision boundary, causing oscillations and potentially increasing iterations. Small learning rates avoid oscillations but require more iterations to reach the solution.

In conclusion, the perceptron's simplicity ensures convergence for linearly separable problems, but moderate learning rate and understanding initialization effects can optimize training efficiency.

c) Consider applying the perceptron to implement the XOR function:

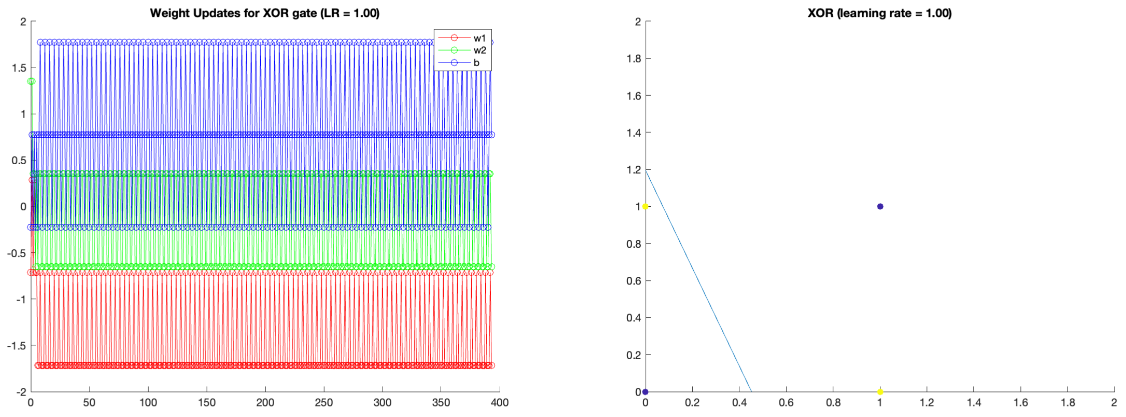


Figure 5 Weight update for XOR (LR=1.0)

The perceptron fails to learn the XOR function because XOR problem is not linearly separable, since perceptron can only generate linear decision boundary.

Q4:

a) Using the standard linear least square method, the fitting result is shown below:

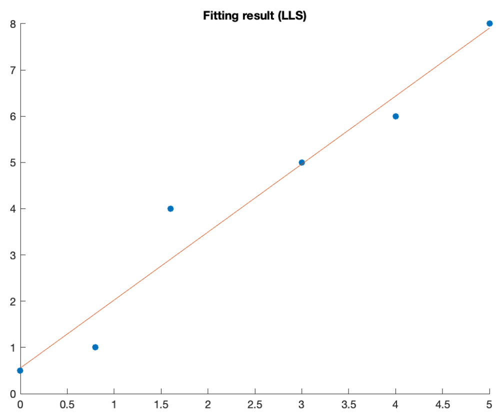


Figure 6 LLS fitting result

b) Using the least-mean-square algorithm, the fitting result and weight update procedure are shown below:

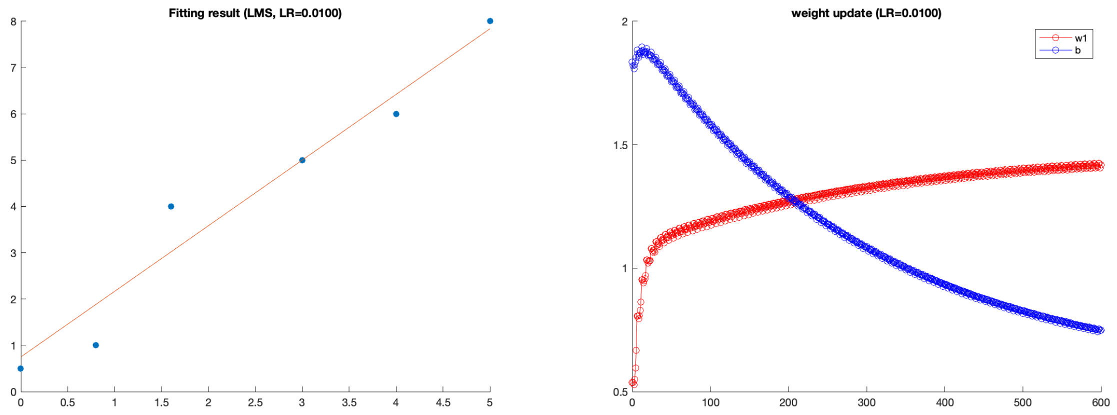
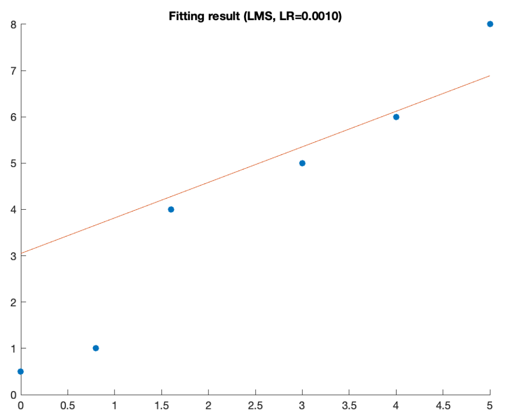


Figure 7 LMS fitting result and weight update (learning rate = 0.01)

c) LLS provides the globally optimal solution (assuming no numerical issues, i.e., $x^T x$ is non-singular). LMS converges to the same solution as LLS only if learning rate is properly tuned and iterations are sufficient. However, LLS cannot solve large data, because the inverse operation is computation expensive.

d) Choose learning rates η_1= 0.001 and η_2= 0.1 and repeat the simulation in (b).



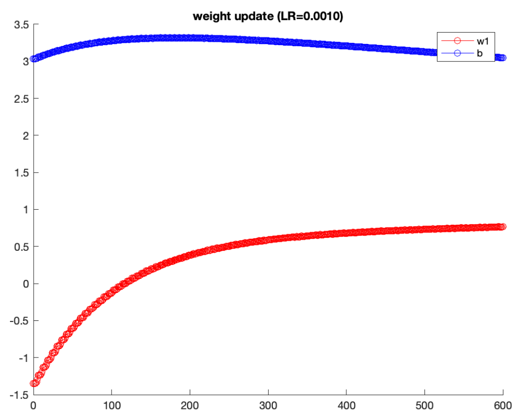
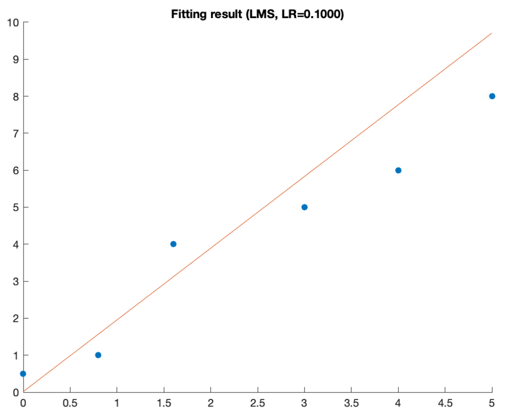


Figure 8 LMS fitting result and weight update (learning rate = 0.001)



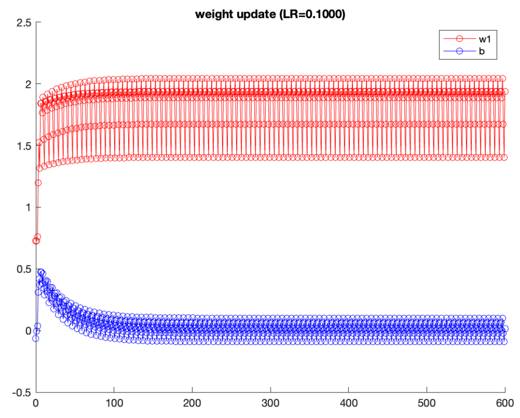


Figure 9 LMS fitting result and weight update (learning rate = 0.01)

According to the simulation results, larger steps per update can accelerate convergence if the initial weights are far from the solution, while large learning rate may cause oscillations. Small learning rates avoid oscillations but require more iterations to reach the solution. For this fitting task, when we choose learning rate η_1= 0.001, 100 epochs are insufficient to converge. In conclusion, both LLS and LMS minimize the same squared error objective, but their approaches differ fundamentally, LLS is a deterministic, one-step method ideal for exact solutions, LMS is a stochastic, iterative method suited for scalability and adaptability.

Q5:

The original cost function is $J(w) = \frac{1}{2}\sum_{i=1}^{n} r^2(i)(d(i) - y(x(i)))^2 + \frac{1}{2}\lambda w^T w$.

Define matrix R as a diagonal matrix of r(i): $R = diag\big(r(1), r(2), \ldots, r(n)\big)$

Then we can rewrite the cost function J(w) as follow:

$$J(w) = \frac{1}{2}e^T Re + \frac{1}{2}\lambda w^T w = \frac{1}{2}(d - Xw)^T R^T R(d - Xw) + \frac{1}{2}\lambda w^T w$$

Where $e = d - y,$ and $y = Xw.$

$$J(\mathbf{w}) = \frac{1}{2}d^T R^T Rd - w^T X^T R^T Rd + \frac{1}{2}w^T X^T R^T RXw + \frac{1}{2}\lambda w^T w,$$

Take the derivative of $J(\mathbf{w})$ with respect to $\mathbf{w}$

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = -X^T R^T Rd + X^T R^T RXw + \lambda w$$

Solve $\nabla_{\mathbf{w}} J(\mathbf{w}) = 0$, rearrange terms: $(X^T R^T RX + \lambda I)w = X^T R^T Rd.$

$$\boldsymbol{w}^* = (X^T R^T RX + \lambda I)^{-1} X^T R^T Rd$$

When $R = I$, the solution reduces to ridge regression: $\boldsymbol{w}^* = (X^T X + \lambda I)^{-1} X^T d.$