National University of Singapore

Faculty of Engineering

Title	Homework 1
Module	EE5904
Name	Zavier Ong Jin Jie
Matriculation No.	A0138993L

Q1

1.1) $\varphi(v) = (v-a)^2 + c$

Input vector $= [+1, x_1, x_2, \dots, x_m]^T$

weight vector $= [b_k, w_{k_1}, w_{k_2}, \dots, w_{k_m}]^T$

Induced local field $= v_k = \sum_{j=0}^{m} w_{kj} x_j$

$\varphi(v) = \xi = (v-a)^2 + c$

$\sqrt{\xi - c} = v - a$

$v = a \pm \sqrt{\xi - c}$

$(b + w_{k_1}x_1 + \dots + w_{k_m}x_m) - (a \pm \sqrt{\xi - c}) = 0.$

$\therefore$ Hyperplane when $(\xi - c) \geq 0$

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

$1 - e^{-v} = \xi + \xi e^{-v}$

$1 - \xi = e^{-v}(1+\xi)$

$\dfrac{1-\xi}{1+\xi} = e^{-v}$

$\ln\left(\dfrac{1-\xi}{1+\xi}\right) = -v$

$v + \ln\left(\dfrac{1-\xi}{1+\xi}\right) = 0$

$(b + w_{k_1}x_1 + \dots + w_{k_m}x_m) + \ln\left(\dfrac{1-\xi}{1+\xi}\right) = 0$

Define constant C to be $\ln\left(\frac{1-\xi}{1+\xi}\right)$

$b + w_{k_1}x_1 + \dots w_{k_m}x_m + C = 0$

$\therefore$ Hyperplane where $\left(\dfrac{1-\xi}{1+\xi}\right) > 0$

1 3) $V_k = \sum_{j=0}^{m} w_{kj} x_j$

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

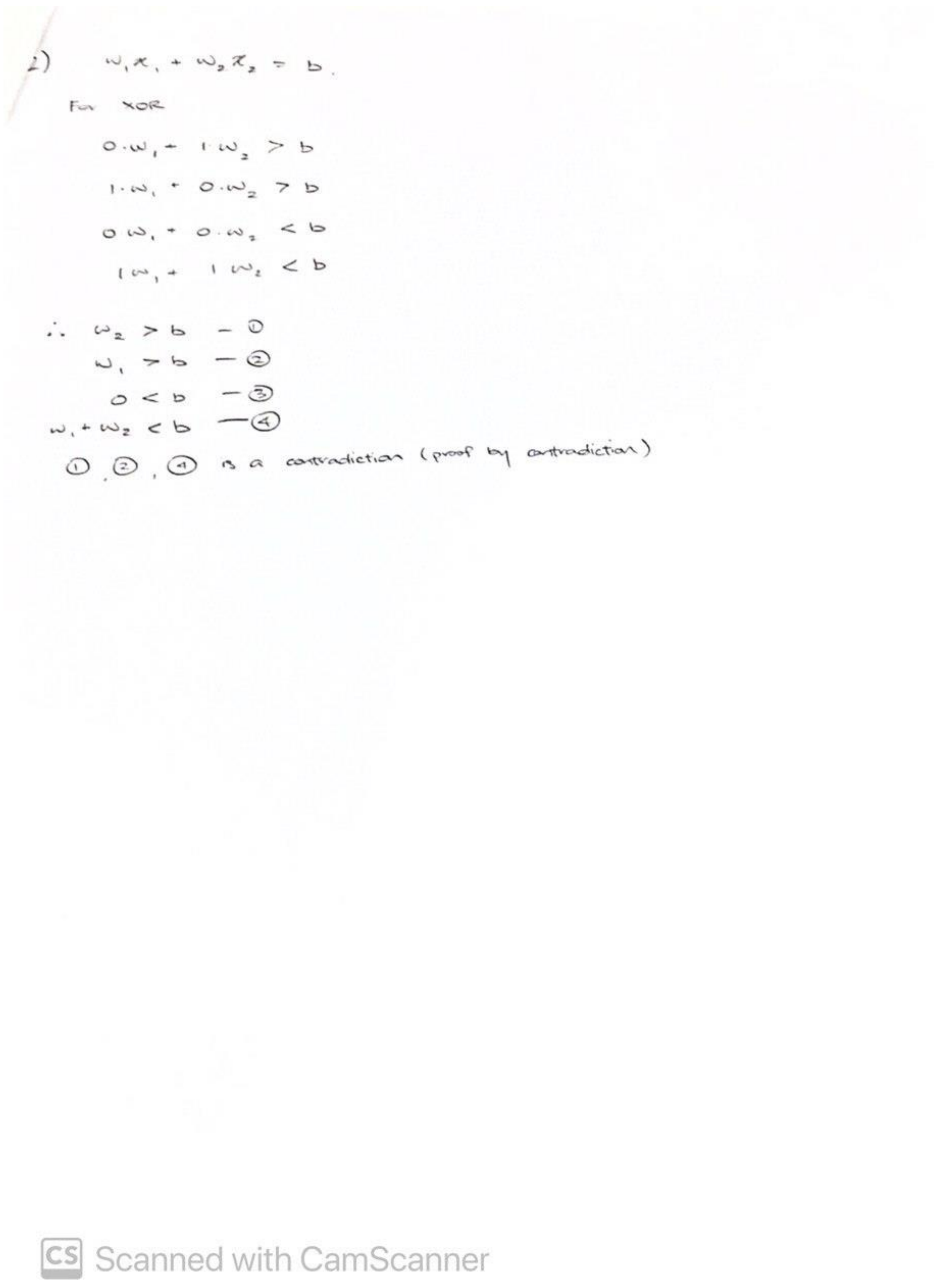
$$\ln \xi = -\frac{(v-m)^2}{2}$$

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

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

$\therefore$ Not a hyperplane

Q2



2) $w_1 x_1 + w_2 x_2 = b$.

For XOR

$0 \cdot w_1 + 1 \cdot w_2 > b$

$1 \cdot w_1 + 0 \cdot w_2 > b$

$0 w_1 + 0 \cdot w_2 < b$

$1 w_1 + 1 w_2 < b$

$\therefore \; w_2 > b$ — ①

$w_1 > b$ — ②

$0 < b$ — ③

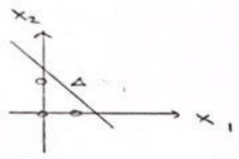
$w_1 + w_2 < b$ — ④

①, ②, ④ is a contradiction (proof by contradiction)

Q3

3a)

3a) i) AND where o = 0 & Δ = 1



$x_2 = -x_1 + 1.5$,

$V = x_2 + x_1 - 1.5 = 0$, $w_1 = 1$; $w_2 = 1$; $b = -1.5$

when $x_1 = 0$, $x_2 = 0$

$v = -1.5 \rightarrow y = 0$

when $x_1 = 1$, $x_2 = 0$

$v = -0.5 \rightarrow y = 0$

when $x_1 = 0$, $x_2 = 1$

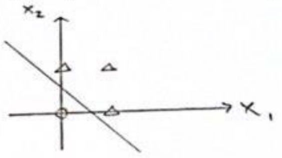
$v = -0.5 \rightarrow y = 0$

when $x_1 = 1$, $x_2 = 1$

$v = 0.5 \rightarrow y = 1$

∴ $v = x_1 + x_2 - 1.5 = 0$; $y = \begin{cases} 0 & \text{if } v < 0 \\ 1 & \text{if } v \geq 0 \end{cases}$

ii) OR , where o = 0, Δ = 1



$x_2 = -x_1 + 0.5$

$v = x_2 + x_1 - 0.5 = 0$, $w_1 = 1$; $w_2 = 1$, $b = -0.5$

∴

x_1	x_2	v		y
0	0	-0.5	$\rightarrow$	0
0	1	0.5	$\rightarrow$	1
1	0	0.5	$\rightarrow$	1
1	1	1.5	$\rightarrow$	1

∴ $v = x_2 + x_1 - 0.5 = 0$, $y \begin{cases} 0 & \text{if } v < 0 \\ 1 & \text{if } v \geq 0 \end{cases}$

iii) Complement

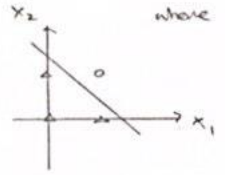
$x_1 = 0.5$

$v = -x_1 + 0.5 = 0$, $w_1 = -1$ $b = 0.5$

x_1	v		y
0	0.5	$\rightarrow$	1
1	-0.5	$\rightarrow$	0

∴ $v = -x_1 + 0.5 = 0$; $y \begin{cases} 0 & \text{if } v < 0 \\ 1 & \text{if } v \geq 0 \end{cases}$

3a) iv) NAND



where ○ = 0, Δ = 1

$x_2 = -x_1 + 1.5$

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

x_1	x_2	v	y
0	0	1.5	$\rightarrow 1$
0	1	0.5	$\rightarrow 1$
1	0	0.5	$\rightarrow 1$
1	1	-0.5	$\rightarrow 0$

$\therefore\ v = -x_1 - x_2 + 1.5 = 0$; $y = \begin{cases} 0 & \text{if } v < 0 \\ 1 & \text{if } v \geq 0 \end{cases}$

b)

3b)

AND

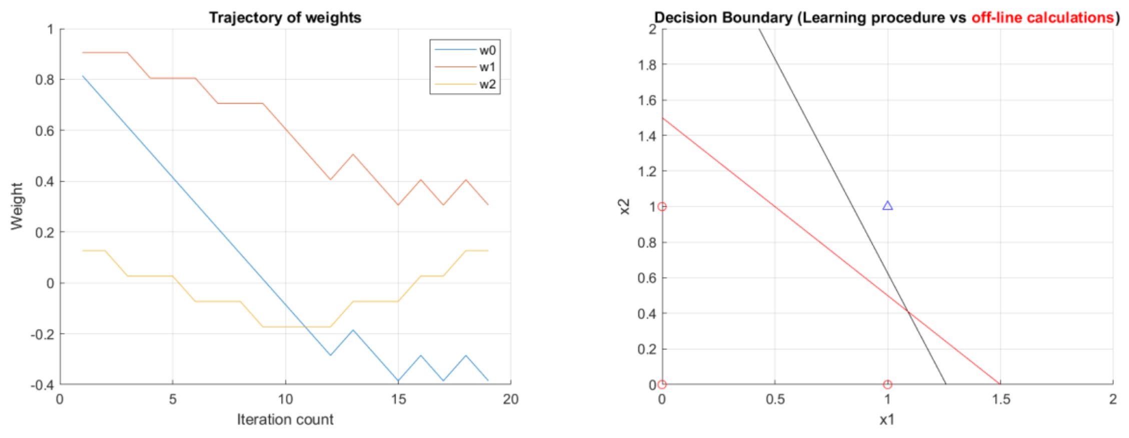


Figure 1. AND Learning Rate of 0.1

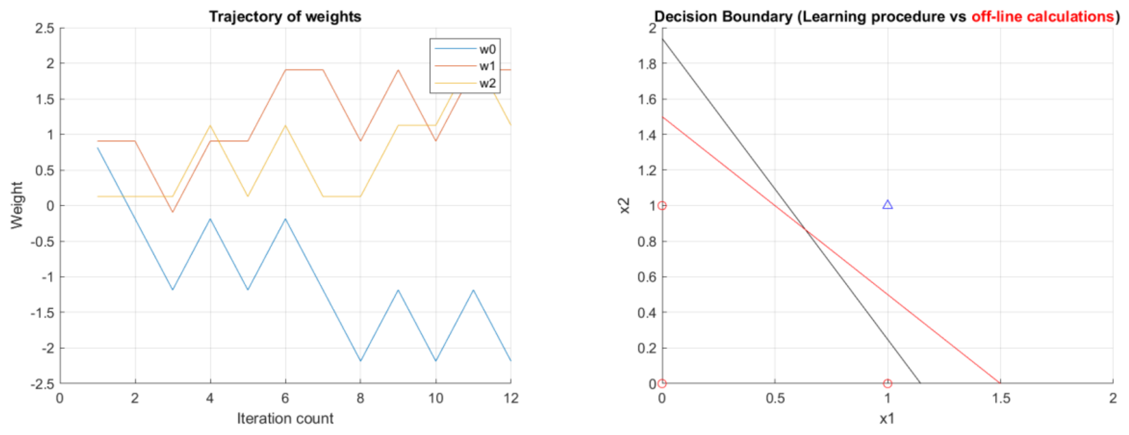


Figure 2. AND Learning Rate of 1.0

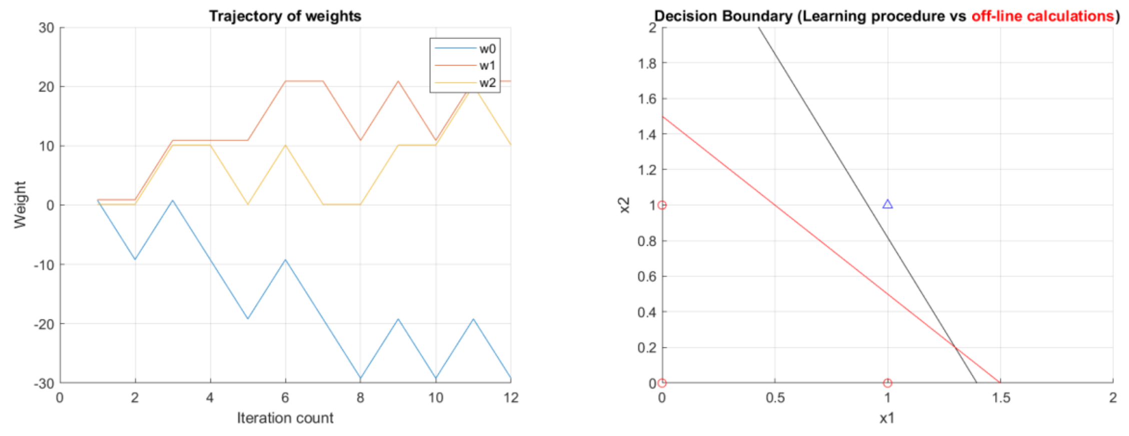


Figure 3. AND Learning Rate of 10.0

OR

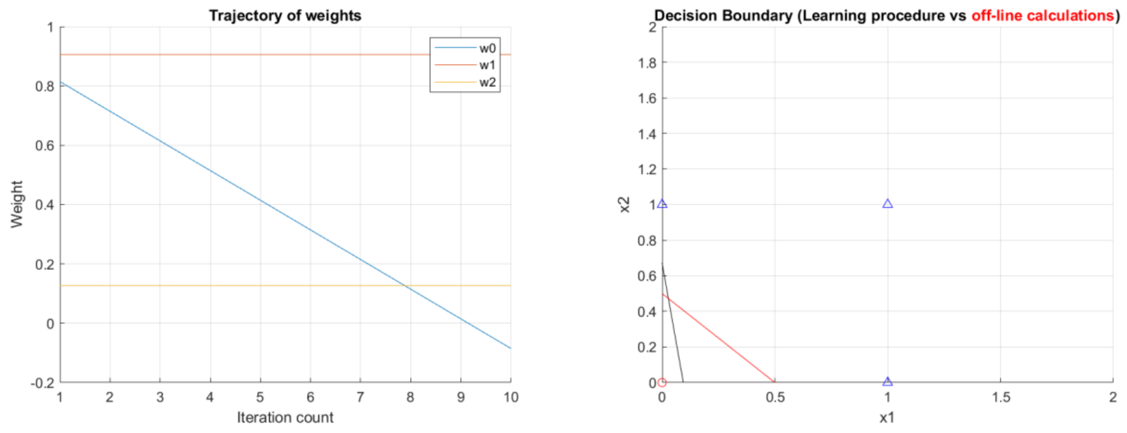


Figure 4. OR Learning Rate of 0.1

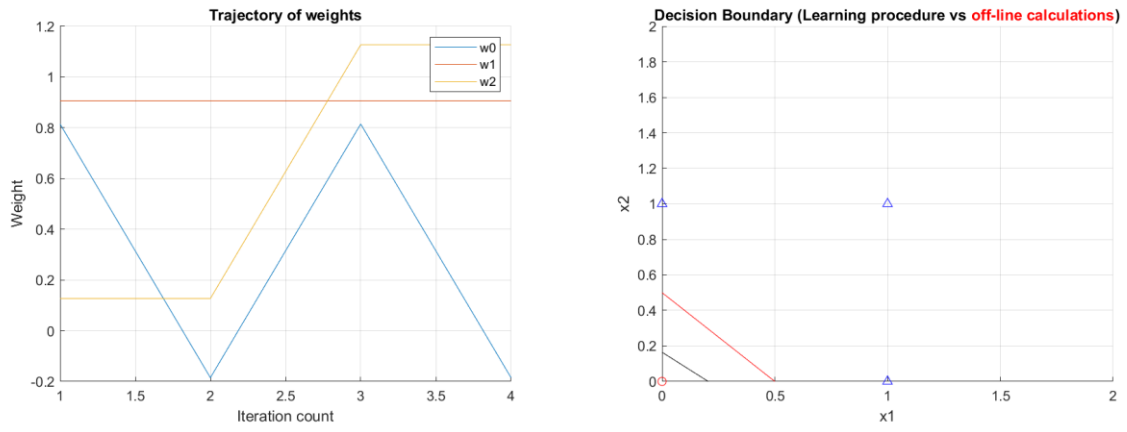


Figure 5. OR Learning Rate of 1.0

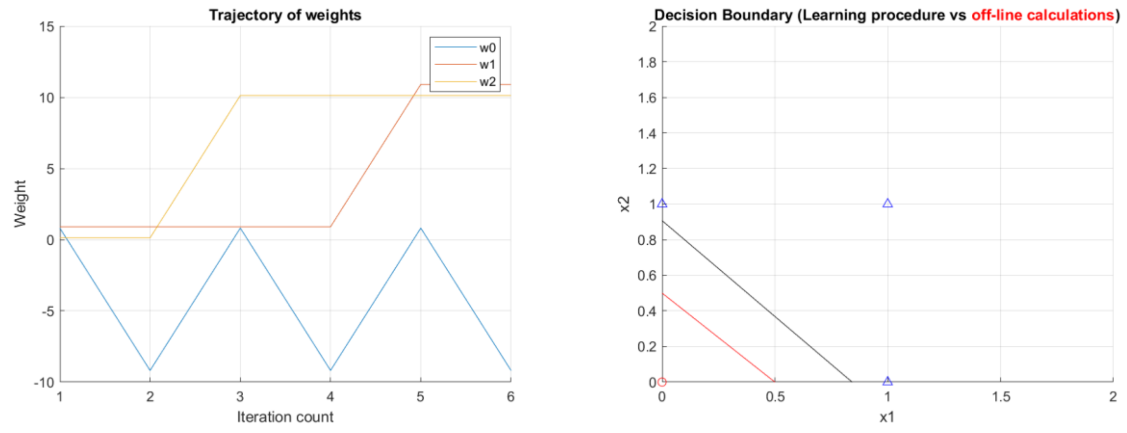


Figure 6. OR Learning Rate of 10.0

COMPLEMENT

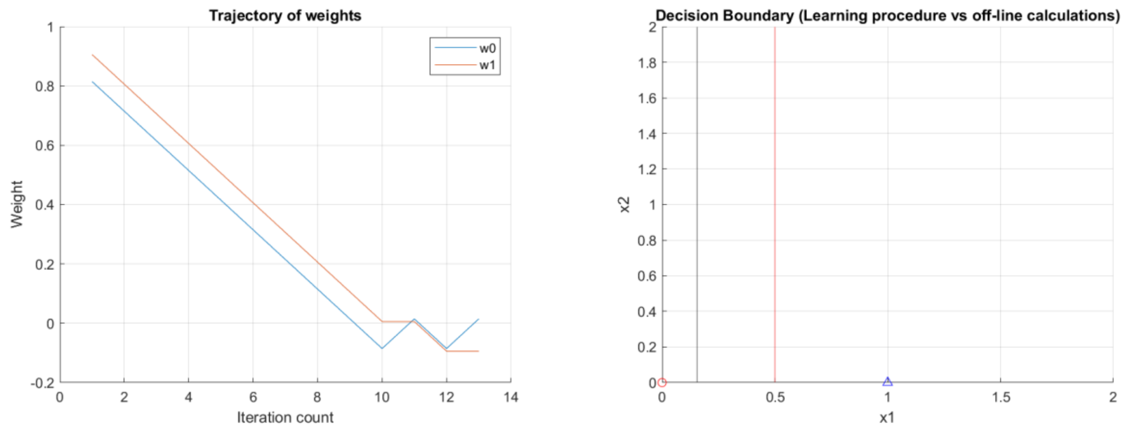


Figure 7. COMP Learning Rate of 0.1

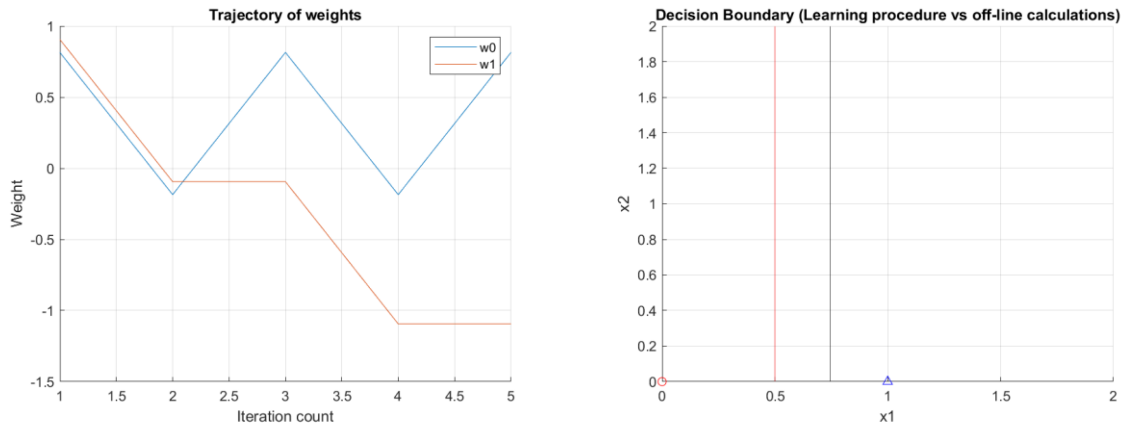


Figure 8. COMP Learning Rate of 1.0

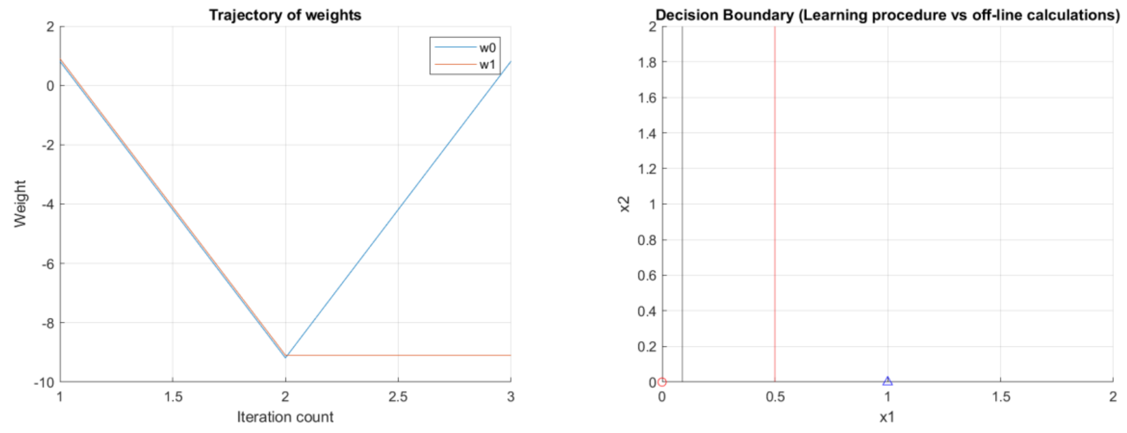


Figure 9. COMP Learning Rate of 10.0

NAND

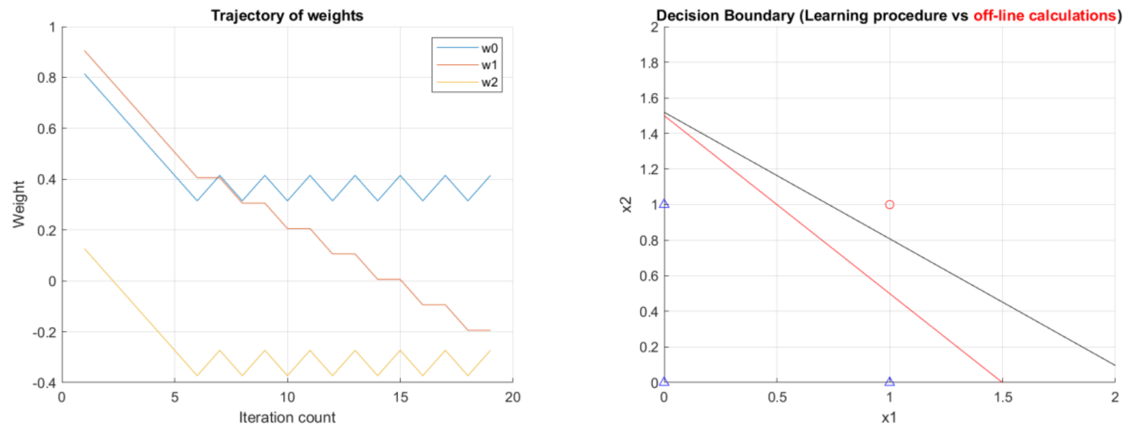


Figure 10. NAND Learning Rate of 0.1

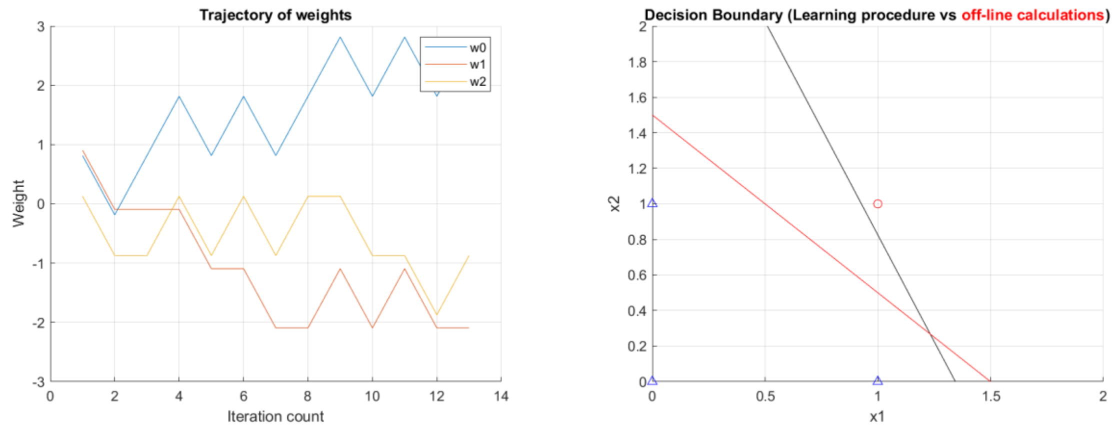


Figure 11. NAND Learning Rate of 1.0

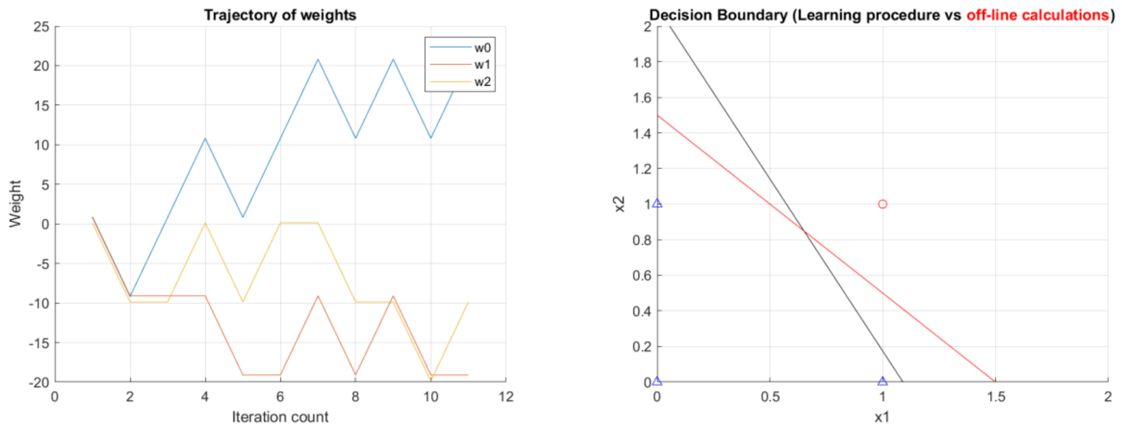


Figure 12. NAND Learning Rate of 10.0

Figure 1 to 12 displays the plots of the implemented logic functions AND, OR, COMPLEMENT and NAND respectively at three different learning rates (0.1, 1 and 10).

By comparing the results with those obtained in (a), we can observe that the decision boundary acquired from off-line calculations and learning procedure differs. However, both methods are still able to classify the input vectors correctly.

In this implementation, all the logic functions are experimented with three different learning rates at 0.1, 1 and 10. We can observe that with different learning rates, the number of iterations needed for the learning procedure to converge is also different. For smaller learning rates (0.1), it constantly requires more iterations to converge. This is because in this equation to update the weight vector:

$$w(n+1) = w(n) + \eta\mathrm{e}(\mathrm{n})\mathrm{x}(\mathrm{n})$$

the new updated weight is relatively smaller as compared to the other learning rates of 1 and 10. For relatively larger learning rates, it does not necessarily mean that it is able to converge with lesser iterations. This is because at each iteration, a large learning rate would cause the decision boundary to shift by a large amount, missing the correct decision boundary. As a result, leading to a slight oscillation around the correct decision boundary.

3c)

From question 2, we have proved that the XOR problem is not linearly separable. As a result, by applying the selection of weights by learning procedure on the XOR functions, the program will not be able to converge and thus, run endlessly. By running my code in debug mode, I have plotted the trajectory of weights for the XOR function (Figure 13).

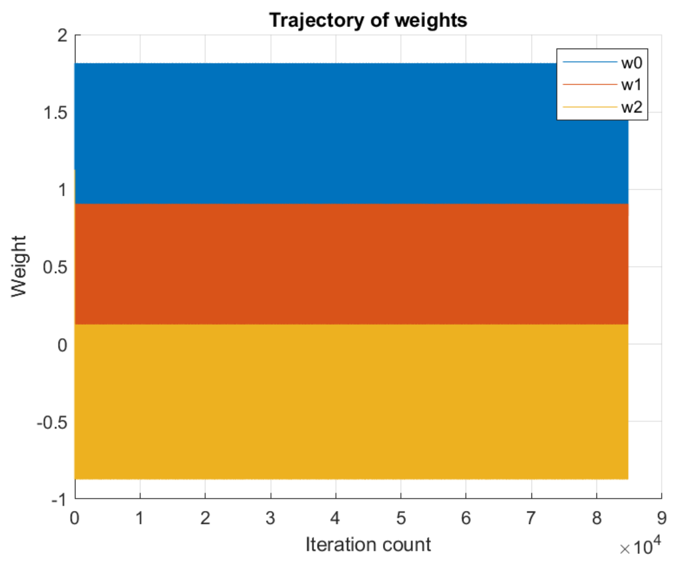


Figure 13. XOR trajectory of weights

As observed from Figure 13, we can observe that the number of iteration at the point on pausing the program is already more than 8000. We can also observe that the 3 weights are not converging but rather oscillating around 1.75 to -0.8.

4)

4a)

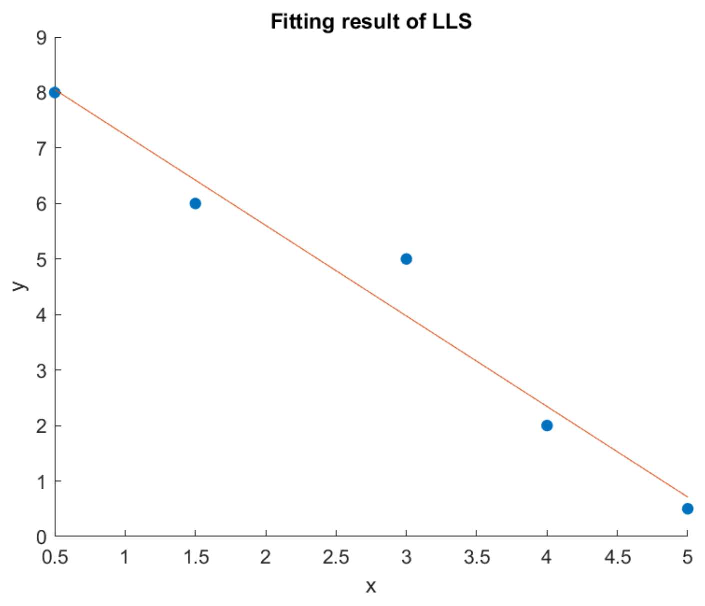


Figure 14. Fitting Result of Linear Least-Squares

Solution of w using LLS: -1.6316

Solution of b using LLS: 8.8684

4b)

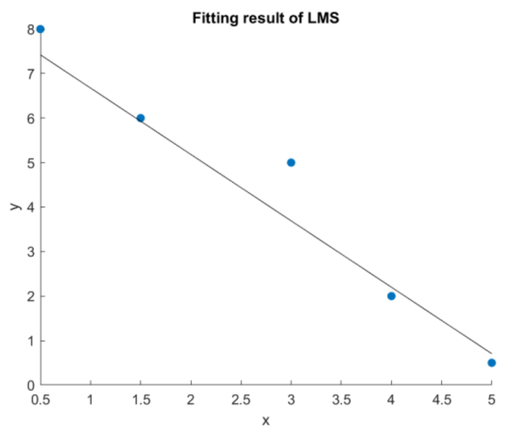


Figure 15. Fitting result of LMS

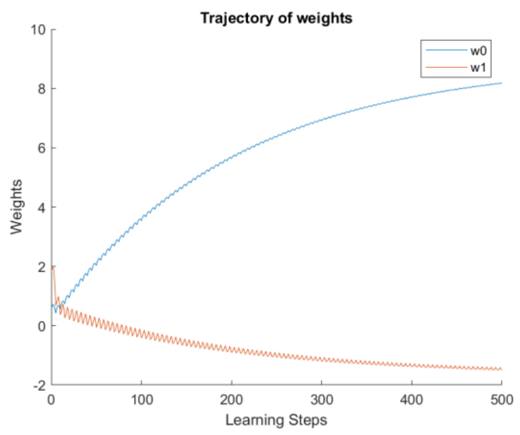


Figure 16. Trajectory of weights

Figure 15 and 16 depicts the fitting result and the trajectories of the weights for the least-mean-square (LMS) algorithm for 100 epochs and the learning rate of 0.02. As shown in Figure 16, we can observe that the weights converge.

Solution of w using LMS: -1.4925

Solution of b using LMS: 8.1683

4c)

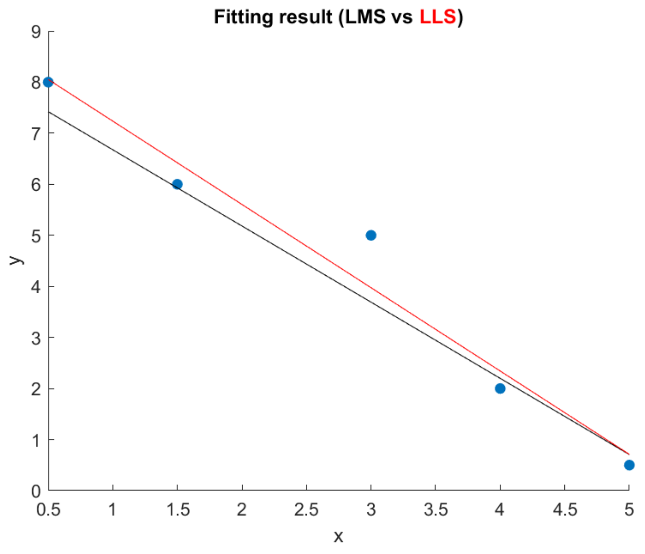


Figure 17. Fitting result (LLS vs LMS) 100 epochs

Figure 17 depicts the fitting results of the linear least-squares(LLS) and least-mean-squares(LMS). LLS is colored red while LMS is color black. This fitting result is the result of the learning rate η is 0.02 at 100 epochs. We can observe in Figure 17, the LMS fitting differs from the LLS result. This is because the LMS algorithm has yet to complete its learning. By changing the number of epochs to 500, we are able to recreate a fitting result similar to the LLS result as shown in Figure 18 below.

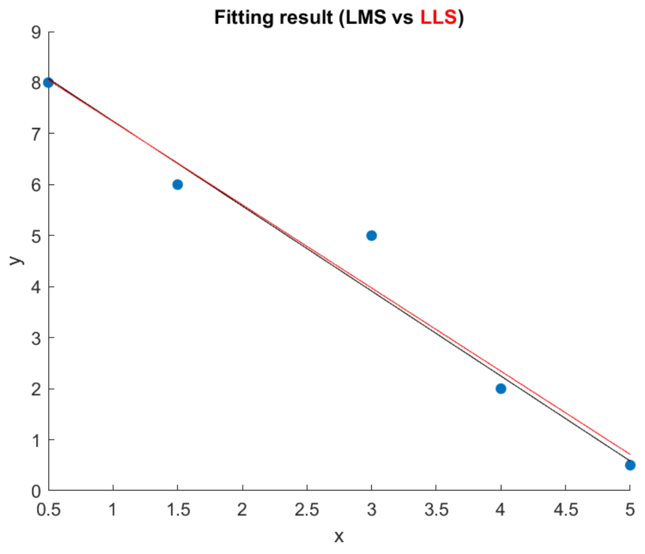


Figure 18. Fitting result (LLS vs LMS) 500 epochs

4d)

In this experiment, we will repeat the same simulation but with different learning rates (0.2, 0.1, 0.002)

Learning Rate 0.002

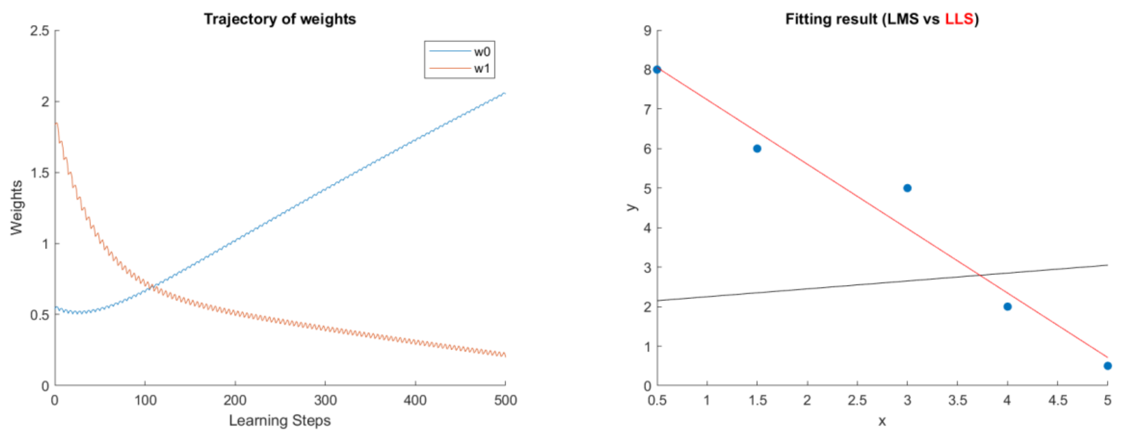


Figure 19. Learning Rate of 0.002, 100 Epochs

As depicted in Figure 19. We can observe that the LMS fitting result differs from the LLS results greatly. This is due to the small learning rate, causing the single layer perceptron to learn much slower. Therefore, to achieve a similar fitting result to LLS, we can increase the number of epochs. This is displayed in Figure 20, where the number of epochs is increase to 2000.

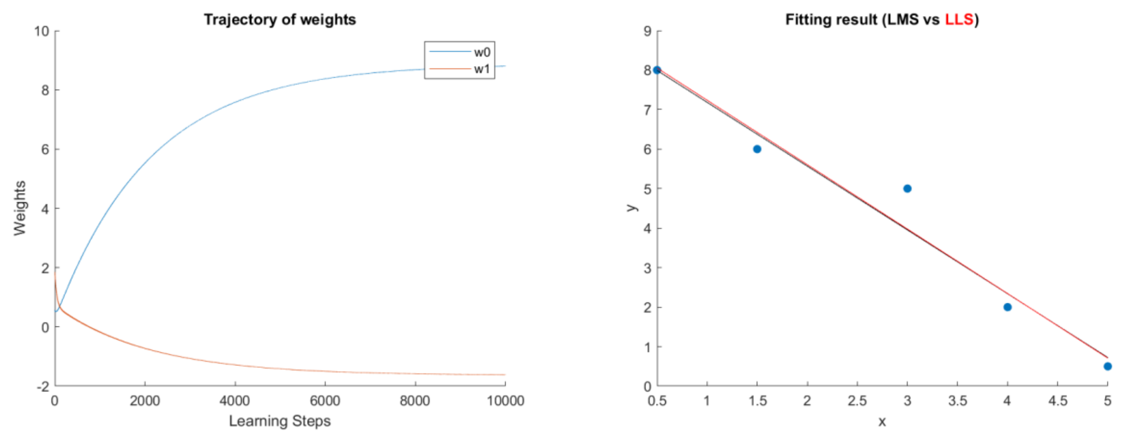


Figure 20. Learning Rate of 0.002, 2000 Epochs

Learning Rate 0.1

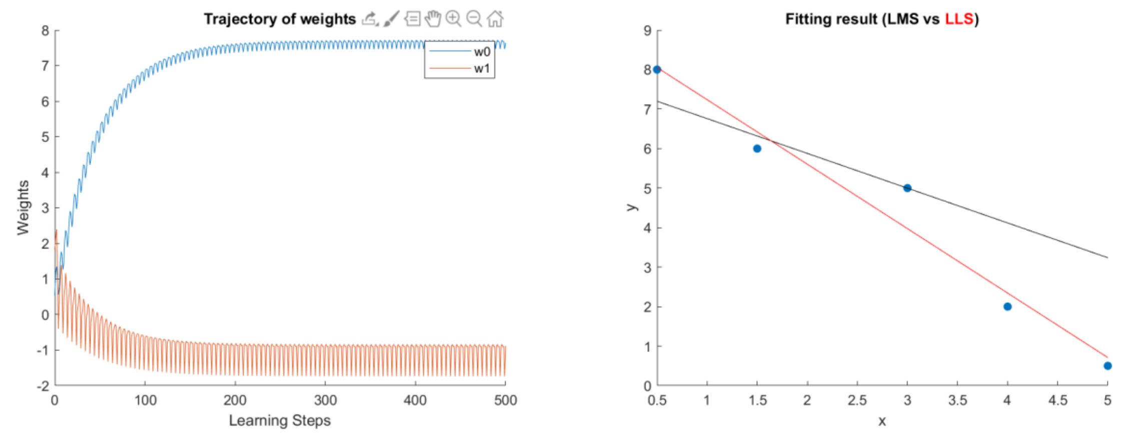


Figure 21. Learning Rate of 0.1, 100 Epochs

For Figure 21, we can observe from the Trajectory of weights plot that the weights w0 and w1 does not converge but rather oscillating between the minimum. This is most likely due to the step size η being too large, causing the gradient descent to "jump over" the minima.

Learning Rate 0.2

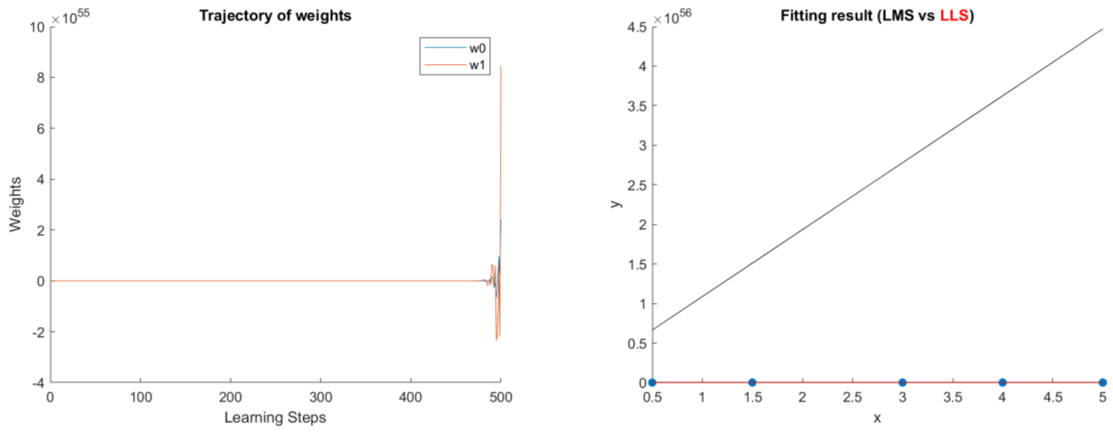


Figure 22. Learning Rate of 0.2, 100 Epochs

In Figure 22, it is also an example of step size η being too large. However, it leads to another cause where the weights increase exponentially causing a divergence instead of convergence. This is because when the oversized step η is taken, the new point has a worse cost function than that original point. Causing the gradient descent algorithm to recalculate the gradient and take an even larger step to hopefully correct its new cost. As a result, causing the weights to increasingly become larger and larger.

Q5

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

$e(n) = d(n) - w^T(n)x(n)$

$\frac{d}{dw}\left(\frac{1}{2}e^2(n) + \frac{1}{2}\lambda\|w(n)\|^2\right)$

By chain rule

$\frac{d}{dw}\left[\frac{1}{2}e^2(n)\right] = e(n) \cdot -x^T(n) = -e(n)x^T(n)$

$\frac{d}{dw}\left[\frac{1}{2}\lambda w^T(n)w(n)\right] = 2\lambda w^T(n)$

$\therefore \frac{dE}{dw} = -e(n)x^T(n) + 2\lambda w^T(n)$

Gradient $g(n) = \left(\frac{dE}{dw}\right)^T = \left[-e(n)x^T(n) + 2\lambda w^T(n)\right]^T$

$= -e(n)x(n) + 2\lambda w(n)$

Applying steepest descent method

$w(n+1) = w(n) - \eta g(n)$

$= w(n) - \eta[-e(n)x(n) + 2\lambda w(n)]$

$= (1-2\lambda)w(n) + \eta e(n)x(n)$ (shown).

Ordinary LMS algo is a special case of the leaky LMS algorithm when

$\lambda = 0$

Appendix

Q3b Code

```
clc
clear

and = [1 1 1 1; 0 0 1 1; 0 1 0 1; 0 0 0 1];
or = [1 1 1 1; 0 0 1 1; 0 1 0 1; 0 1 1 1];
nand = [1 1 1 1; 0 0 1 1; 0 1 0 1; 1 1 1 0];
complement = [1 1; 0 1; 1 0];


func = or;
l_rate = 0.1;
[dim, num_inputs] = size(func);
%setting seed to 0 to ensure reproducibility
rng('default')
s = rng;

weights = rand(1, dim-1);
count = 1;

classified = false;

while ~(classified)
    classified = true;
    for i = 1: num_inputs
        y = (dot(weights(count, 1:dim-1), func(1:dim-1, i)) >= 0);
        err = func(dim, i) - y;
        if (~(err == 0))
            classified = false;
            weights(count+1, 1:dim-1) = weights(count, 1:dim-1) + (l_rate *
err * func(1:dim-1, i))';
            count = count + 1;
        end
    end
end

if dim-1 == 2 %handling complement case
    fig1 = figure;
    title('Trajectory of weights');
    hold on;
    xlabel("Iteration count");
    ylabel("Weight");
    plot(weights(1:size(weights, 1), 1));
    plot(weights(1:size(weights, 1), 2));
    legend({'w0', 'w1'});
    grid on;
    hold off;
    saveas(fig1, sprintf('COMP_%.1f.png', l_rate));

    fig2 = figure;
    title('Decision Boundary (Learning procedure vs off-line
calculations)')
    hold on;
    xlim([0,2])
    ylim([0,2])
    %plot truth table
    for i=1:num_inputs
```

```
        if func(end, i) == 1
            plot(func(end, i),0, 'b^');
        else
            plot(func(end, i),0, 'ro');
        end
    end
    %plot learning procedure
    x = -weights(end,1)/weights(end,2);
    xline(x,'k');
    %plot COMPLEMENT off line calculation
    xline(0.5, 'r');
    xlabel("x1");
    ylabel("x2");
    grid on
    hold off
    saveas(fig2, sprintf('COMP_db_%.1f.png', l_rate));
end

if dim-1 == 3
    fig1 = figure;
    title('Trajectory of weights');
    hold on;
    xlabel("Iteration count");
    ylabel("Weight");
    plot(weights(1:size(weights,1),1));
    plot(weights(1:size(weights,1),2));
    plot(weights(1:size(weights,1),3));
    legend({'w0','w1','w2'});
    grid on
    hold off
    saveas(fig1, sprintf('OR_%.1f.png', l_rate));

    fig2 = figure;
    title('Decision Boundary (Learning procedure vs \color{red}off-line
calculations\color{black})')
    hold on;
    x = -10:100;
    m = -weights(end,2)/weights(end,3);
    c = -weights(end,1)/weights(end,3);
    xlim([0,2])
    ylim([0,2])
    %plot truth table
    for i = 1:num_inputs
        if func(end,i)  == 1
            plot(func(2,i),func(3,i),'b^');
        else
            plot(func(2,i),func(3,i),'ro');
        end
    end
    %plot learning procedure
    y = m * x + c;
    plot(x, y,'k')
    %plot off line calculation
    y_off_and = -x + 1.5;
    y_off_or = -x + 0.5;
    y_off_nand = -x + 1.5;
    plot(x, y_off_or, 'r');
    xlabel("x1");
    ylabel("x2");
    grid on
    hold off
    saveas(fig2, sprintf('OR_db_%.1f.png', l_rate));
end
```

Q3c Code

```
clc
clear

xor = [1 1 1 1; 0 1 0 1; 0 0 1 1; 0 1 1 0];

l_rate = 1;
[dim, num_inputs] = size(xor);
%setting seed to 0 to ensure reproducibility
rng('default')
s = rng;

weights = rand(1, dim-1);
count = 1;

classified = false;

while ~(classified)
    classified = true;
    for i = 1: num_inputs
        y = (dot(weights(count, 1:dim-1), xor(1:dim-1, i)) >= 0);
        err = xor(dim, i) - y;
        if (~(err == 0))
            classified = false;
            weights(count+1, 1:dim-1) = weights(count, 1:dim-1) + (l_rate *
err * xor(1:dim-1, i))';
            count = count + 1;
        end
    end
end

fig1 = figure;
title('Trajectory of weights');
hold on;
xlabel("Iteration count");
ylabel("Weight");
plot(weights(1:size(weights,1),1));
plot(weights(1:size(weights,1),2));
plot(weights(1:size(weights,1),3));
legend({'w0','w1','w2'});
grid on
hold off
saveas(fig1, "xor.png");

figure;
title('Decision Boundary (Learning procedure vs off-line calculations)')
hold on;
x = -10:100;
m = -weights(end,2)/weights(end,3);
c = -weights(end,1)/weights(end,3);
xlim([0,2])
```

```
ylim([0,2])
%plot truth table
for i = 1:num_inputs
    if xor(end,i) == 1
        plot(xor(2,i),xor(3,i),'b^');
    else
        plot(xor(2,i),xor(3,i),'ro');
    end
end
%plot learning procedure
y = m * x + c;
plot(x, y,'k')
%plot off line calculation
y_off_and = -x + 1.5;
y_off_or = -x + 0.5;
y_off_nand = -x + 1.5;
plot(x, y_off_nand, 'r');
xlabel("x1");
ylabel("x2");
grid on
hold off
```

Q4a Code

```
clc;
clear;

x = [1 0.5; 1 1.5; 1 3; 1 4.0; 1 5.0];
d = [8.0; 6.0; 5; 2; 0.5];

%inv(x'*x) * x' == (x'*x) \ x'
w = (x'*x) \ x' *d;
y = x*w;

fig1 = figure;
xlabel('x');
ylabel('y');
title("Fitting result of LLS");
hold on;
scatter(x(:,2), d, 'filled')
p = plot(x(:,2), y);
disp(w);
%saveas(fig1, 'q4a.png');
```

Q4bcd Code

```matlab
clear;
clc;

x = [1 0.5; 1 1.5; 1 3; 1 4.0; 1 5.0];
d = [8.0, 6.0, 5, 2, 0.5];
rng('default');
s = rng;

weights = randn(1, 2);
trajectory = weights;
l_rate = 0.002;
total_epochs = 2000;

for epoch = 1:total_epochs
    for i = 1:length(x(:,1))
        if (x(i, 1:2)*weights') == d(i)
            continue
        else
            err = d(i) - x(i, 1:2)*weights';
            weights = weights + l_rate*err*x(i, 1:2);
            trajectory = [trajectory;weights];
        end
    end
end

y_lms = x*weights';
w = (x'*x) \ x' *d';
y_lls = x*w;
disp(weights);

fig1 = figure;
title('Fitting result (LMS vs \color{red}LLS\color{black})');
%title('Fitting result of LMS');
xlabel('x');
ylabel('y');
hold on;
scatter(x(:, 2), d, 'filled');
plot(x(:, 2), y_lms, 'k');
plot(x(:,2), y_lls, 'r');
hold off;
saveas(fig1, sprintf('4d_fit_%.3f_%d.png', l_rate, total_epochs));

fig2 = figure;
hold on;
title('Trajectory of weights');
xlabel("Learning Steps");
ylabel("Weights");
plot(0:length(trajectory)-1,trajectory(:,1))
plot(0:length(trajectory)-1,trajectory(:,2))
legend({'w0','w1'})
hold off;
saveas(fig2, sprintf('4d_traj_%.3f_%d.png', l_rate, total_epochs));
```