Q3

a)To implement AND logic, we use weights w=[-1.5 1 1]T for bias, x1 and x2 respectively,

V= x1 + x2 – 1.5

The values yield the following table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x1 | x2 | bias | v | y |
| 0 | 0 | 1 | -1.5 | 0 |
| 0 | 1 | 1 | -0.5 | 0 |
| 1 | 0 | 1 | -0.5 | 0 |
| 1 | 1 | 1 | 0.5 | 1 |

To implement OR logic, we can use weights w=[-0.5 1 1]T for bias, x1 and x2 respectively,

V = x1 + x2 – 1.5

The values yield the following table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x1 | x2 | bias | v | y |
| 0 | 0 | -0.5 | -0.5 | 0 |
| 0 | 1 | -0.5 | -0.5 | 1 |
| 1 | 0 | -0.5 | -0.5 | 1 |
| 1 | 1 | -0.5 | 1.5 | 1 |

To implement COMPLEMENT logic, we can use weights w=[0.5 -1]T for bias and respectively,

V = -x + 0.5

The values yield the following table:

|  |  |  |  |
| --- | --- | --- | --- |
| X | Bias | V | Y |
| 0 | 0.5 | 0.5 | 1 |
| 1 | 0.5 | -0.5 | 0 |

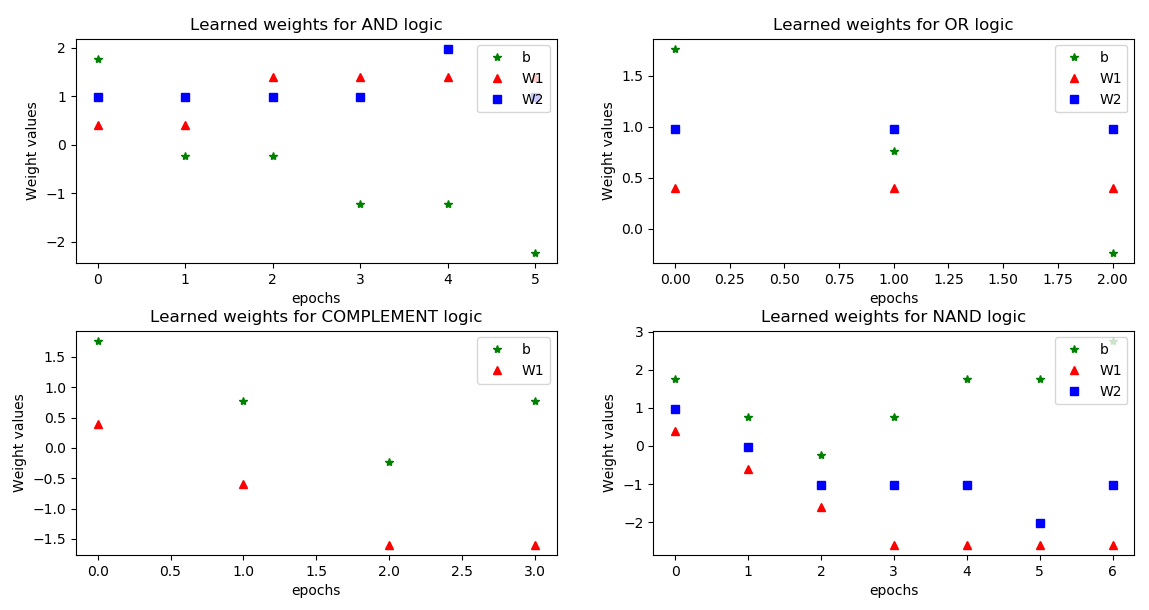
To implement NAND logic, we can use weights w=[1.5 -1 -1]T for bias, x1 and x2 respectively,

V = -x1  - x2 + 1.5

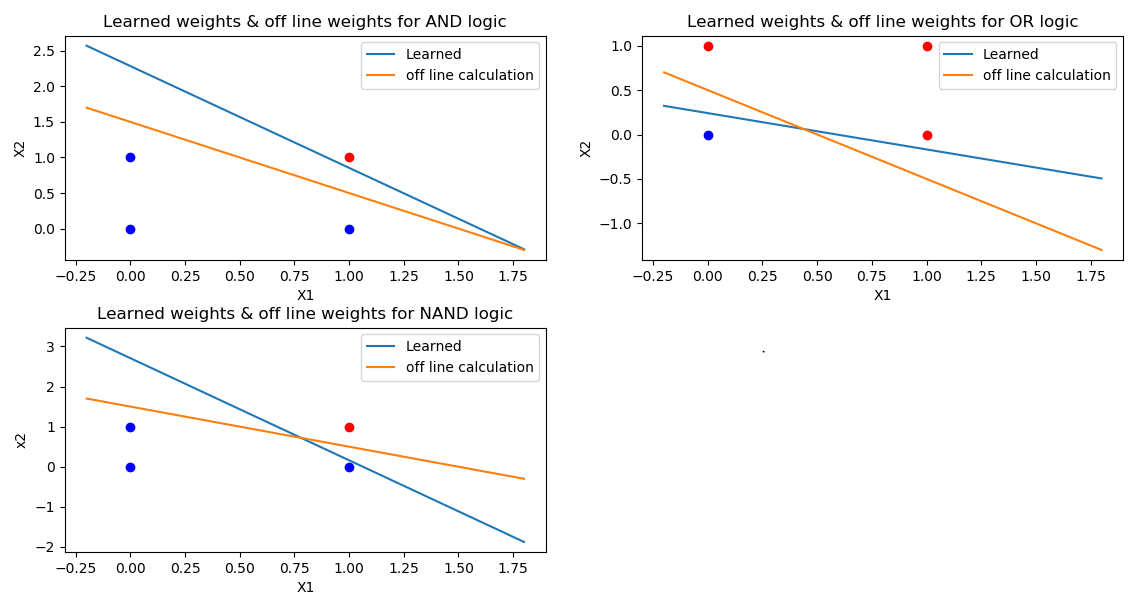
The values yield the following table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x1 | x2 | bias | v | y |
| 0 | 0 | 1.5 | 1.5 | 1 |
| 0 | 1 | 1.5 | 0.5 | 0 |
| 1 | 0 | 1.5 | 0.5 | 0 |
| 1 | 1 | 1.5 | -0.5 | 0 |

b1.Trajectories of weights for each cases at learning rate = 1



The initial weights will update when there is an error signal. The weights will converge when there is no more error signal produced. The final obtained weights in each case are very close to the off-line calculated weights. Both off-line calculated weights and iterated weights can implement AND, OR & NAND logics.



For Complement logic, ignore w2 since w2 is 0. the final learned weights is [0.76 -1.60] : v= -1.6x +0.76

When x = 0, v = 0.76 which greater than 0, y = f(v) = 1; when x = 1, v = -0.84, y = f(v) = 0

Both the learned weights and calculate off line weights model can implement complement logic.

b2.

From my observation on different learning rate, it controls how quickly the model is adapted to the logic gate problem. Smaller learning rates require more training epochs. As the smaller rates make smaller changes on the weights for each update. Whereas, larger learning rates can cause a rapid change in weights and require fewer epochs. If the learning rate is too large can result the model to converge too fast to a suboptimal result; if the learning rate is too small can cause the process to get stuck.

c. The computer simulation results show that the training process will keep on iterating, the training perceptron will not converge, no matter how weights updated, it is not able to find a linear line or hyper-plane to separate the two class.

##############################Q3 source code#################################

import numpy as np

import matplotlib.pyplot as plt

# inputs initializing

# columns = (bias, x1, x2, y)

AND\_set = np.array([[1,0,0,0],

                    [1,0,1,0],

                    [1,1,0,0],

                    [1,1,1,1]])

OR\_set = np.array([[1,0,0,0],

                   [1,0,1,1],

                   [1,1,0,1],

                   [1,1,1,1]])

COMPLEMENT\_set = np.array([[1,0,1],

                           [1,1,0]])

NAND\_set =np.array([[1,0,0,1],

                    [1,0,1,1],

                    [1,1,0,1],

                    [1,1,1,0]])

XOR\_set =np.array([[1,0,0,0],

                    [1,0,1,1],

                    [1,1,0,1],

                    [1,1,1,0]])

# learning rate

learn\_rate = 1

def activation\_fn(x):

    return 1 if x >=0 else 0

def train(x):

    dim = x.shape[1]

    col = x.shape[0]

    iterate = True

    iteration = 0

    #initial weights

    np.random.seed(0)

    weights = np.random.randn(1,dim -1)

    W = weights

    print("Initial weights is " + str(weights))

    # for i in range(iteration):

    while iterate:

        iterate = False

        for j in range(col):

            v = weights.dot(x[j,:-1])

            v\_out = activation\_fn(v)

            error = x[j,-1] - v\_out

            if error != 0 :

                # print("wrong weights is " + str(weights))

                weights = weights + learn\_rate\*x[j,:-1]\*error

                # W = np.append(W,weights,axis = 0)

                # print("Updated weights is " + str(weights))

                iterate = True

        if iterate == True:

            W = np.append(W,weights,axis = 0)

            print("Updated weights is " + str(weights))

        iteration += 1

    print(str(iteration) + " times of epochs")

    return W

def plotweights(W\_AND,W\_OR,W\_NAND,W\_COMPLEMENT):

    iter\_range = range(W\_AND.shape[0])

    iter\_OR = range(W\_OR.shape[0])

    iter\_NAND = range(W\_NAND.shape[0])

    iter\_COMPLEMENT = range(W\_COMPLEMENT.shape[0])

    plt.figure(num=3, figsize=(10, 3),)

    plt.subplots\_adjust(wspace =0.2, hspace =0.3)

    plt.subplot(2,2,1)

    plt.plot(iter\_range,W\_AND[:,0:1],'g\*',iter\_range,W\_AND[:,1:2],'r^',iter\_range,W\_AND[:,2:3],'bs')

    # plt.plot(iter\_range,W[:,1:2],'r^','w1')

    # plt.plot(iter\_range,W[:,2:3],'bs')

    plt.title("Learned weights for AND logic")

    plt.xlabel("epochs")

    plt.ylabel("Weight values")

    plt.legend(('b','W1','W2'),loc = 'upper right')

    plt.subplot(2,2,2)

    plt.plot(iter\_OR,W\_OR[:,0:1],'g\*',iter\_OR,W\_OR[:,1:2],'r^',iter\_OR,W\_OR[:,2:3],'bs')

    # plt.scatter(iter\_OR,W3,c='r')

    # plt.scatter(iter\_OR,W4,c='b')

    plt.title("Learned weights for OR logic")

    plt.xlabel("epochs")

    plt.ylabel("Weight values")

    plt.legend(('b','W1','W2'),loc = 'upper right')

    plt.subplot(2,2,3)

    plt.plot(iter\_COMPLEMENT,W\_COMPLEMENT[:,0:1],'g\*',iter\_COMPLEMENT,W\_COMPLEMENT[:,1:2],'r^')

    # plt.scatter(iter\_COMPLEMENT,W7,c='r')

    plt.title("Learned weights for COMPLEMENT logic")

    plt.xlabel("epochs")

    plt.ylabel("Weight values")

    plt.legend(('b','W1'),loc = 'upper right')

    plt.subplot(2,2,4)

    plt.plot(iter\_NAND,W\_NAND[:,0:1],'g\*',iter\_NAND,W\_NAND[:,1:2],'r^',iter\_NAND,W\_NAND[:,2:3],'bs')

    plt.title("Learned weights for NAND logic")

    plt.xlabel("epochs")

    plt.ylabel("Weight values")

    plt.legend(('b','W1','W2'),loc = 'upper right')

    plt.show()

def compare\_weights(W\_AND,W\_OR,W\_NAND):

    X = np.arange(-0.2,2,0.5)

    y\_AND\_learn = -(W\_AND[W\_AND.shape[0]-1,1]\*X + W\_AND[W\_AND.shape[0]-1,0])/W\_AND[W\_AND.shape[0]-1,2]

    y\_AND\_off = -X + 1.5

    y\_OR\_learn = -(W\_OR[W\_OR.shape[0]-1,1]\*X + W\_OR[W\_OR.shape[0]-1,0])/W\_OR[W\_OR.shape[0]-1,2]

    y\_OR\_off = -X + 0.5

    y\_NAND\_learn = -(W\_NAND[W\_NAND.shape[0]-1,1]\*X + W\_NAND[W\_NAND.shape[0]-1,0])/W\_NAND[W\_NAND.shape[0]-1,2]

    y\_NAND\_off = -X + 1.5

    plt.figure(num=3, figsize=(10, 3),)

    plt.subplots\_adjust(wspace =0.2, hspace =0.3)

    plt.subplot(2,2,1)

    plt.plot([0,0,1],[0,1,0],'bo',[1],[1],'ro')

    plt.plot(X,y\_AND\_learn,label = "Learned")

    plt.plot(X,y\_AND\_off,label = "off line calculation")

    plt.title("Learned weights & off line weights for AND logic")

    plt.xlabel("X1")

    plt.ylabel("X2")

    plt.legend()

    plt.subplot(2,2,2)

    plt.plot([0],[0],'bo',[0,1,1],[1,0,1],'ro')

    plt.plot(X,y\_OR\_learn,label = "Learned")

    plt.plot(X,y\_OR\_off,label = "off line calculation")

    plt.title("Learned weights & off line weights for OR logic")

    plt.xlabel("X1")

    plt.ylabel("X2")

    plt.legend()

    plt.subplot(2,2,3)

    plt.plot([0,0,1],[0,1,0],'bo',[1],[1],'ro')

    plt.plot(X,y\_NAND\_learn,label = "Learned")

    plt.plot(X,y\_NAND\_off,label = "off line calculation")

    plt.title("Learned weights & off line weights for NAND logic")

    plt.xlabel("X1")

    plt.ylabel("x2")

    plt.legend()

    plt.show()

W\_AND = train(AND\_set)

W\_OR = train(OR\_set)

W\_NAND = train(NAND\_set)

W\_COMPLEMENT = train(COMPLEMENT\_set)

Plotweights = plotweights(W\_AND,W\_OR,W\_NAND,W\_COMPLEMENT)

# Compare\_weights = compare\_weights(W\_AND,W\_OR,W\_NAND)

# X\_OR =train(XOR\_set)

##############################End of Q3#####################################