```
import numpy as np
import matplotlib.pyplot as plt
```

Assignment 1. Given a perceptron with weight vector (w1, w2)

T = (1, 1)T and bias w0 = 2, plot

the partition of R 2 that is realized by this perceptron and mark the area where the

perceptron outputs 1.

Answer

The linear equation for a perceptron is

Here i took bias as W_3

> $\sum_{i=1}^{n} w_i . x_i + w_n +_1 = 0$

for two input perceptron

> $w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 = 0$

this is equivalent with

> $x_2 = -\frac{w_1}{w_2} \cdot x_1 - \frac{w_3}{w_2}$

Therefore

> $m = -\frac{w_1}{w_2}$

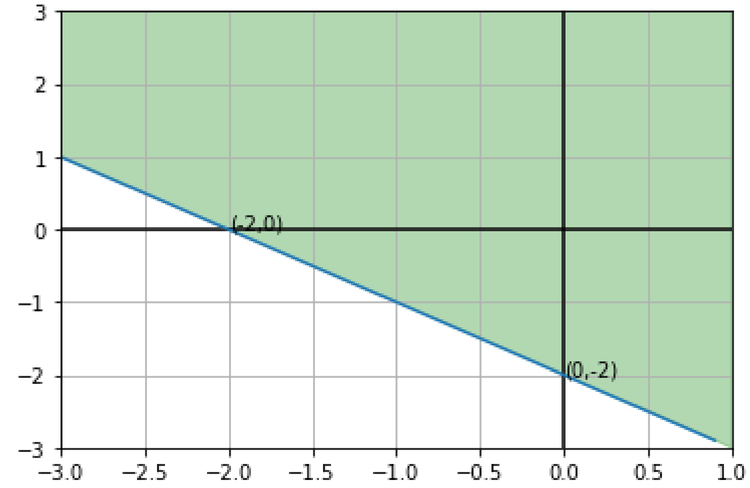
and

> $c = -\frac{w3}{w2}$

```
W1 = 1
W2 = 1
W3 = 2


fig, ax = plt.subplots()

xmin, xmax = -3, 1
X = np.arange(xmin, xmax, 0.1)

ax.set_xlim([xmin, xmax])
ax.set_ylim([-3, 3])

plt.axhline(0, color='black')
plt.axvline(0, color='black')
```

```
ax.grid(True)


m = -W1/W2
c= -W3/W2
ax.plot(X, m * X + c, label="decision boundary" )
ax.text(-2,0,s="(-2,0)")
ax.text(0,-2,s="(0,-2)")

plt.fill_between([-3,1], [1,-3], 4,
                 color='green',
                 alpha=0.3)
plt.plot()
```

```
[]
```

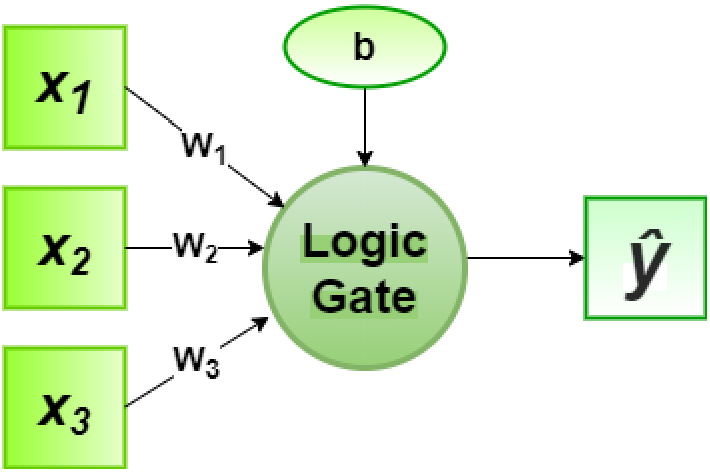


Assignment 2 . Implement OR logic function of three variables: y = f(x1, x2, x3), where y ∈ {−1, +1},

xi ∈ {0, 1} using a single perceptron. Give a boolean expression describing your function. Derive the weight vector using two methods: (Create a data set by just listing the truth table - represent FALSE by -1 and TRUE by +1)

(a) by working out the equation of the decision plane,

(b) by training using the perceptron learning algorithm that we discussed in the first lecture.



(b)

Percepron Algorithm

```
class Perceptron(object):
    """Implements a perceptron network"""
    def __init__(self, input_size, lr=1, epochs=100):
        self.W = np.zeros(input_size+1)
        # add one for bias
        self.epochs = epochs # no of iterations
        self.lr = lr # learning rate

    def activation_fn(self, x): # simple step function
        return 1 if x >= 0 else 0

    def predict(self, x):
        z = np.dot(self.W,x)
        a = self.activation_fn(z)
        return a

    def fit(self, X, d):
        for _ in range(self.epochs):
            for i in range(d.shape[0]):
                x = np.insert(X[i], 0, 1) # for setting the bias input to 1
                y = self.predict(x)
                e = d[i] - y
                self.W = self.W + self.lr * e * x
```

Main function to Train

```
if __name__ == '__main__':
 #FOR TRAINING
    X = np.array([
      [0, 0, 0],
      [0, 0, 1],
      [0, 1, 0],
      [0, 1, 1],
      [1, 0, 0],
      [1, 0, 1],
      [1, 1, 0],
      [1, 1, 1]
    ])
    d = np.array([0, 1, 1, 1, 1,1,1,1]) # TRUTH TABLE FOR THE OR FUNCTION

    perceptron = Perceptron(input_size=3)
    perceptron.fit(X, d) #training
    # print(perceptron.W)
    weights = perceptron.W # wieghts after training
    #y = mx +c
    # m = slope

    print("The Weights and Bias after training are : \n")
    print("bias= {}, w1= {} , w2= {}, w3= {}".format(weights[0], weights[1], weights[2],we
```

```
The Weights and Bias after training are :

bias= -1.0, w1= 1.0 , w2= 1.0, w3= 1.0
```

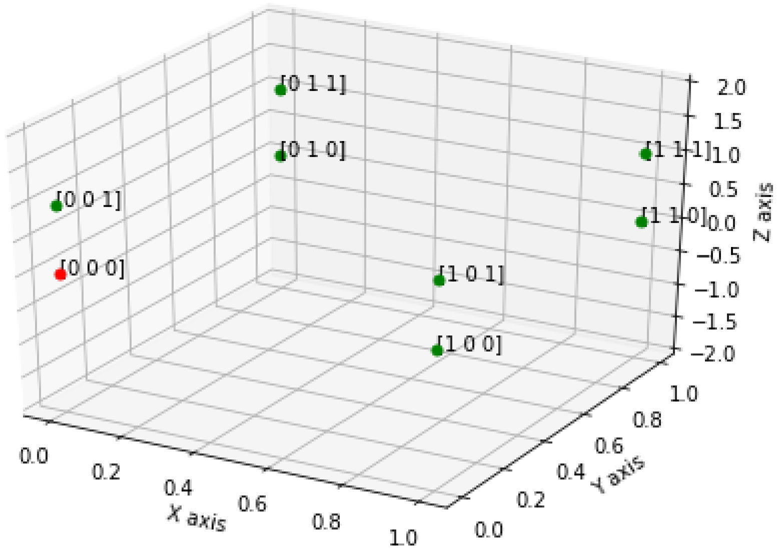
Output / Prediction

```
#Testing

def TruthTable(X):
  x = np.insert(X,0,1)
  y=perceptron.predict(x)
  return y

dataset = np.array([
     [0, 0, 0],
     [0, 0, 1],
     [0, 1, 0],
     [0, 1, 1],
     [1, 0, 0],
     [1, 0, 1],
     [1, 1, 0],
     [1, 1, 1]
   ])
print("Inputs  Predicted_Output")
for i in range(len(dataset)):
  print(" {}   {}".format(dataset[i], TruthTable(dataset[i])))
```

```
Inputs  Predicted_Output
 [0 0 0]   0
 [0 0 1]   1
 [0 1 0]   1
 [0 1 1]   1
 [1 0 0]   1
 [1 0 1]   1
 [1 1 0]   1
 [1 1 1]   1
```

```


# Trying Prediction

# This function predicts the output and depicts the point in red color if
# its not included and green vice versa

def setColor(X):
  x = np.insert(X,0,1)
  y=perceptron.predict(x)
  if y ==0 :
    return "r"
  else:
    return "g"
```

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D



fig = plt.figure()
ax = Axes3D(fig)

for i in range(len(dataset)):
    ax.scatter(dataset[i][0],dataset[i][1],dataset[i][2],c=setColor(dataset[i]),s=25, mark
    ax.text(dataset[i][0],dataset[i][1],dataset[i][2],s=str(dataset[i]))
ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('Z axis')


# ax.plot_surface(X, Y, Z, rstride=1, cstride=1)
# ax.contourf(X, Y, Z, zdir='z', offset=-2, cmap=plt.cm.hot)

ax.set_zlim(-2, 2)

plt.show()
```

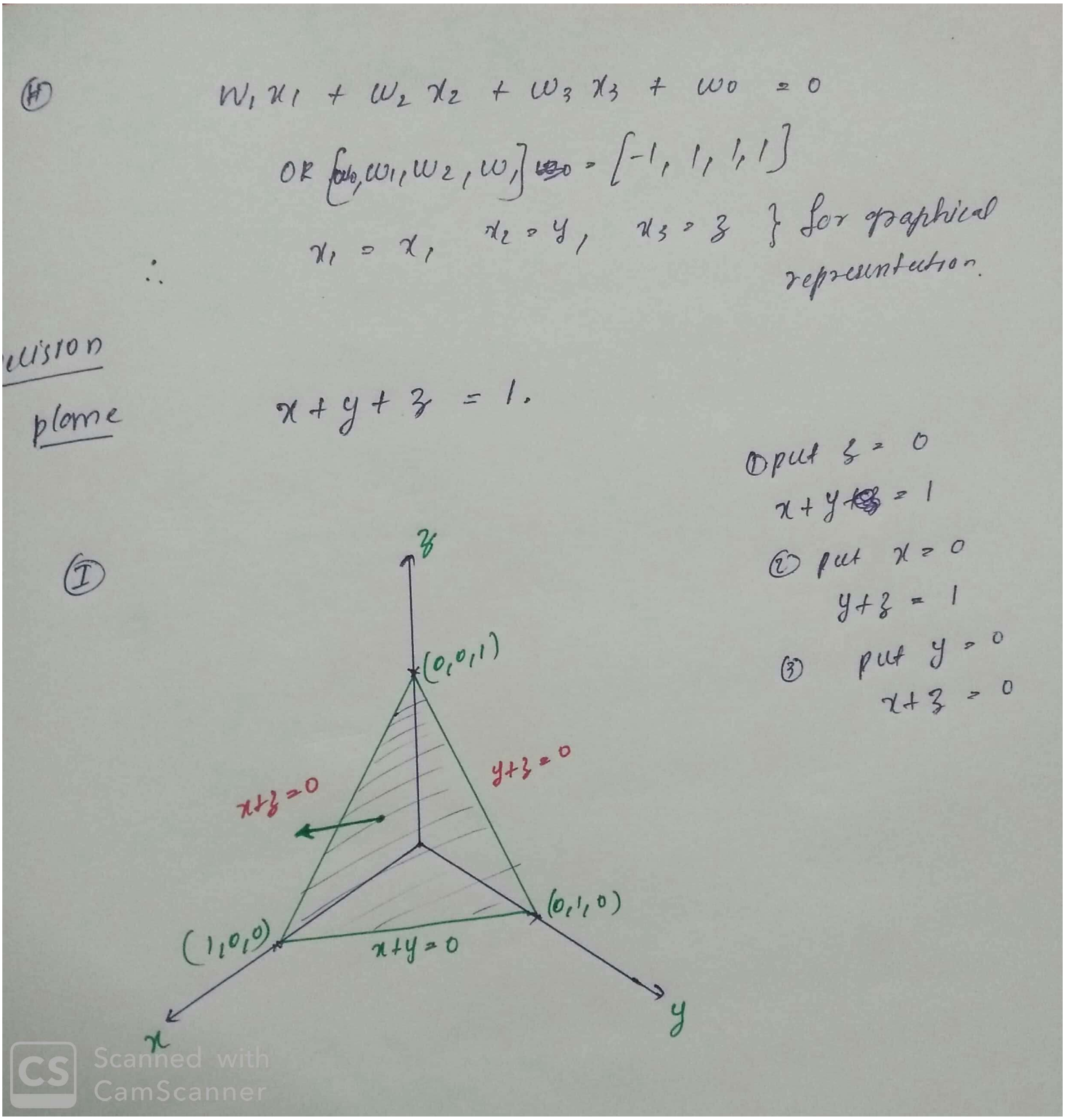


(a)

So from the above training we got the Weights

weights

> $$[W_0, W_1, W_2, W_3] = [-1, 1, 1, 1]$$



(H) $w_1 x_1 + w_2 x_2 + w_3 x_3 + w_0 = 0$

OR $[w_0, w_1, w_2, w] = [-1, 1, 1, 1]$

$\therefore$ $x_1 = x,\quad x_2 = y,\quad x_3 = z$ } for graphical representation.

ecision plane $x + y + z = 1.$

(1) put $z = 0$
$x + y = 1$

(2) put $x = 0$
$y + z = 1$

(3) put $y = 0$
$x + z = 0$

```
# rough work area

s= np.array([2,3,3])
np.insert(s, 0 ,1)
```

```
array([1, 2, 3, 3])
```