Ai-Assignment03-Q1

June 23, 2021

[184]:
```
# Xor function return binary from depend on the input
def Xor(x1,x2):

    if x1==x2:
        return 1
    else:
        return 0
```

[185]:
```
# Xor Trunth Table :
# dictionary format sample
# dict= {
#     (x1,x2) :  [("Actual output",),("Perceptron output",)]
# }

perceptron= {

}

# Truth table
def XorTable(x1_vector,x2_vector):

    for i in range(len(x1_vector)):

#         print(Xor(x1_vector[i],x2_vector[i]))
        perceptron[(x1_vector[i]),x2_vector[i]]=[("Actual␣
 ↪output",Xor(x1_vector[i],x2_vector[i]))]

    return perceptron
```

[220]:
```
# Equation = w_old + learning_Rate * (actual_ouput - perceptron_output) * x[i]

def Gradient_decent(old_weight_vector ,␣
 ↪actual_output,perceptron_output,x_values,  learning_rate ):
```

```
#     print("Old Weight Vector : ",old_weight_vector )
#     print("Actual output : ",actual_output)
#     print("perceptron output : ",perceptron_output)
#     print("X Values : ",x_values[0])
#     print("Learning Rate : ",learning_rate)

    for i in range(len(x_values)):
#         print("X values " ,x_values[i])
        actual_perceptron = actual_output - perceptron_output
        actual_perceptron_into_xValues = actual_perceptron * x_values[i]
        temp= old_weight_vector[i]+ learning_rate *␣
 ↪actual_perceptron_into_xValues
        old_weight_vector[i] = temp

    return old_weight_vector
```

[228]:
```
import numpy as np
np.random.seed(1293)
import pprint as pp
def main():
    x0 = 1 # baise input
    x1 = [0,0,1,1]
    x2 = [0,1,0,1]
    perceptron = XorTable(x1,x2)

    # weight vector randomly as W1 -> w1,w2,w3
    w_i = [round(np.random.random() , 1),round(np.random.random() , 1),round(np.
 ↪random.random() , 1)]
#     print("Weight Vector : ",w_i)


    for i in range(len(x1)):
#         print((x1[i],x2[i]))
#
#         print("Acuumulation :",(x0*w_i[0])+(x1[i]*w_i[1])+(x2[i]*w_i[2]))


        if (x0*w_i[0])+(x1[i]*w_i[1])+(x2[i]*w_i[2]) > 0:
```

```
#             print(perceptron[(x1[i],x2[i])])
            perceptron[(x1[i],x2[i])].append(("Perception output" ,1))
        else:
            perceptron[(x1[i],x2[i])].append(("Perception output" ,0))
#             pp.pprint("P - >",perceptron)


    W_delta = [0 , 0 , 0]  ## initilze with zero
#     print("Perceptron : ",perceptron)

    x_values = [(1,0,0),(1,0,1),(1,1,0),(1,1,1)]

    for i in range(4):

        perceptron_output =  perceptron[(x1[i],x2[i])][1][1]
        actual_output = perceptron[(x1[i],x2[i])][0][1]

        W_delta = Gradient_decent(W_delta␣
 ↪,actual_output,perceptron_output,x_values[i],learning_rate=0.1)
#         print("return vector  ",W_delta)

    # update the weight vector with the old vector

    print("return Vector : ",W_delta)
    print("Random vector : ",w_i)

#     vector  = old_Vector[i] + random_Vector[i]
    return [W_delta[0]+w_i[0],W_delta[1]+w_i[1],W_delta[2]+w_i[2]]




main()
```

```
return Vector :  [-0.1, -0.1, -0.1]
Random vector :  [0.0, 0.5, 0.2]
```

[228]: [-0.1, 0.4, 0.1]

[ ]:

[ ]:

[ ]:

[ ]: