Perceptron From Scratch

ML practical 4

Dhruv Patel 17162121014

In [343]:

```
import numpy as np
```

In [344]:

```
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

In [345]:

```
def sigmoid_derivative(x):
    return x*(1-x)
```

In [346]:

```
# set the hyper-parameters
lr = 0.1
epochs = 10000
inputNeurons=2
hiddenNeurons = 2
outputNeurons = 1
```

In [347]:

```
# now we initialize the weights

# hidden_weights = np.random.uniform(size =(inputNeurons,hiddenNeurons))
# hidden_bias = np.random.uniform(size=(1,hiddenNeurons))

# output_weights = np.random.uniform(size =(hiddenNeurons,outputNeurons))
# output_bias = np.random.uniform(size=(1,outputNeurons))


```

In [348]:

```
## we will now see our initialized weights

print("Hidden Layer Weights:\n{} \n\n Hidden Layer Bias: {} ".format(hidden_
```

```
Hidden Layer Weights:
[[3.68067683 5.72557421]
 [3.68180422 5.73108652]]

 Hidden Layer Bias: [[-5.63454073 -2.37275633]]
```

In [349]:

```
def train(inputs,expected_output):

    hidden_weights = np.random.uniform(size =(inputNeurons,hiddenNeurons))
    hidden_bias = np.random.uniform(size=(1,hiddenNeurons))

    output_weights = np.random.uniform(size =(hiddenNeurons,outputNeurons))
    output_bias = np.random.uniform(size=(1,outputNeurons))

    for i in range(epochs):
        #Forward Propagation
        hidden_layer_activation = np.dot(inputs,hidden_weights)+hidden_bias
        hidden_layer_output = sigmoid(hidden_layer_activation)

        output_layer_activation = np.dot(hidden_layer_output,output_weights)
        predicted_output = sigmoid(output_layer_activation)

        #Backpropagation
        error = expected_output - predicted_output
        d_predicted_output = error * sigmoid_derivative(predicted_output)

        error_hidden_layer = d_predicted_output.dot(output_weights.T)
        d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_laye

        #Updating Weights and Biases
        output_weights += hidden_layer_output.T.dot(d_predicted_output) * lr
        output_bias += np.sum(d_predicted_output,axis=0,keepdims=True) * lr
        hidden_weights += inputs.T.dot(d_hidden_layer) * lr
        hidden_bias += np.sum(d_hidden_layer,axis=0,keepdims=True) * lr
    return predicted_output,hidden_weights,hidden_bias,output_weights,output
```

In [361]:

```
def convert_binary(expected_output,preds):
#     expected_output=expected_output.tolist()
#     preds = preds.tolist()
#     print(preds)
    count=1
    for i in range(len(preds)):
        if preds[i][0]>=0.5:
            curr_binary =1
        else:
            curr_binary=0
        print("For input {} Expected value was: {} , Predicted Value is: {}"
        count+=1
```

AND Training

In [362]:

```
inputs = np.array([[0,0],[0,1],[1,0],[1,1]])
expected_output = np.array([[0],[0],[0],[1]])

```

In [363]:

```
predicted_output,hidden_weights,hidden_bias,output_weights,output_bias=train
```

In [364]:

```
convert_binary(expected_output,predicted_output)
```

```
For input 1 Expected value was: 0 , Predicted Value is: 0
For input 2 Expected value was: 0 , Predicted Value is: 0
For input 3 Expected value was: 0 , Predicted Value is: 0
For input 4 Expected value was: 1 , Predicted Value is: 1
```

In [365]:

```
predicted_output
```

Out[365]:

```
array([[0.0106777 ],
       [0.03582489],
       [0.03543114],
       [0.95445481]])
```

OR Training

In [366]:

```
inputs = np.array([[0,0],[0,1],[1,0],[1,1]])
expected_output = np.array([[0],[1],[1],[1]])

```

In [367]:

```
predicted_output,hidden_weights,hidden_bias,output_weights,output_bias=train
```

In [369]:

```
convert_binary(expected_output,predicted_output)
```

```
For input 1 Expected value was: 0 , Predicted Value is: 0
For input 2 Expected value was: 1 , Predicted Value is: 1
For input 3 Expected value was: 1 , Predicted Value is: 1
For input 4 Expected value was: 1 , Predicted Value is: 1
```

In [380]:

```
predicted_output
```

Out[380]:

```
array([[0.04619242],
       [0.97450903],
       [0.97372888],
       [0.99163242]])
```

In [381]:

```
hidden_weights,hidden_bias,output_weights,output_bias
```

Out[381]:

```
(array([[4.53499075, 1.51040298],
        [4.47308069, 1.73650039]]),
 array([[-2.3857028 , -0.78058524]]),
 array([[7.48046162],
        [1.58443409]]),
 array([[-4.15590898]]))
```

For XNOR Training

In [382]:

```
inputs = np.array([[0,0],[0,1],[1,0],[1,1]])
expected_output = np.array([[1],[0],[0],[1]])

```

In [383]:

```
predicted_output,hidden_weights,hidden_bias,output_weights,output_bias=train
```

In [384]:

```
convert_binary(expected_output,predicted_output)
```

```
For input 1 Expected value was: 1 , Predicted Value is: 1
For input 2 Expected value was: 0 , Predicted Value is: 0
For input 3 Expected value was: 0 , Predicted Value is: 0
For input 4 Expected value was: 1 , Predicted Value is: 1
```

In [385]:

```
predicted_output
```

Out[385]:

```
array([[0.93566168],
       [0.06080888],
       [0.0619127 ],
       [0.93217631]])
```

In [386]:

```
hidden_weights,hidden_bias,output_weights,output_bias
```

Out[386]:

```
(array([[5.76633265, 3.46307802],
        [6.11418605, 3.51654622]]),
 array([[-2.40002401, -5.31892491]]),
 array([[-7.27467605],
        [ 7.91497096]]),
 array([[3.24368522]]))
```

In [ ]:

```

```