In [1]:

```
import numpy as np
```

In [285]:

```
# import pandas as pd
# # TODO: Set weight1, weight2, and bias
# weight1 = 1.0
# weight2 = 1.0
# bias = -0.501

# # DON'T CHANGE ANYTHING BELOW
# # Inputs and outputs
# test_inputs = [(0, 0), (0, 1), (1, 0), (1, 1)]
# correct_outputs = [False, True, True, True]
# outputs = []

# # Generate and check output
# for test_input, correct_output in zip(test_inputs, correct_outputs):
#     linear_combination = weight1 * test_input[0] + weight2 * test_input[1]
#     output = int(linear_combination >= 0)
#     is_correct_string = 'Yes' if output == correct_output else 'No'
#     outputs.append([test_input[0], test_input[1], linear_combination, outp

# # Print output
# num_wrong = len([output[4] for output in outputs if output[4] == 'No'])
# output_frame = pd.DataFrame(outputs, columns=['Input 1', '  Input 2', '  L
# if not num_wrong:
#     print('Nice!  You got it all correct.\n')
# else:
#     print('You got {} wrong.  Keep trying!\n'.format(num_wrong))
# print(output_frame.to_string(index=False))
```

Building a Simple Perceptron Class for AND implementation

In [139]:
```
class Perceptron(object):

    def __init__(self,inputs,iterations=10,learning_rate = 0.01):
        self.iterations = iterations
        self.learning_rate = learning_rate
        self.weights = np.zeros(inputs+1)

    def predict(self,inputs):
        summ = np.dot(inputs,self.weights[1:])+self.weights[0]
        if summ>0:
            activation =1
        else:
            activation = 0
        return activation


    def train(self,training_inputs,labels):
        for i in range(self.iterations):
            for inputs,label in zip(training_inputs,labels):
                prediction = self.predict(inputs)

                self.weights[1:] += self.learning_rate*(label-prediction)*in
                self.weights[0] += self.learning_rate*(label-prediction)

            print("weights after iteration {} :{},new bias: {} ".format(i,se
        print("Final Weights :{},new bias: {} ".format(self.weights[1:],self

```

In [135]:
```
training_inputs = []
training_inputs.append(np.array([1, 1]))
training_inputs.append(np.array([1, 0]))
training_inputs.append(np.array([0, 1]))
training_inputs.append(np.array([0, 0]))

#Adding Labels
labels = np.array([1, 0, 0, 0])
```

In [136]:
```
perceptron = Perceptron(2)
```

In [137]:
```
perceptron.train(training_inputs,labels)  ## training our perceptron
```

```
weights after iteration 0 :[0. 0.],new bias: -0.01
weights after iteration 1 :[0.   0.01],new bias: -0.01
weights after iteration 2 :[0.   0.01],new bias: -0.02
weights after iteration 3 :[0.01 0.01],new bias: -0.02
weights after iteration 4 :[0.01 0.02],new bias: -0.02
weights after iteration 5 :[0.01 0.02],new bias: -0.02
weights after iteration 6 :[0.01 0.02],new bias: -0.02
weights after iteration 7 :[0.01 0.02],new bias: -0.02
weights after iteration 8 :[0.01 0.02],new bias: -0.02
weights after iteration 9 :[0.01 0.02],new bias: -0.02
Final Weights :[0.01 0.02],new bias: -0.02
```

In [138]:
```python
inputs = np.array([1,1])  # testing our perceptron and it is working perfect
print(perceptron.predict(inputs))
```

1

In [92]:
```python
inputs = np.array([1,0])
print(perceptron.predict(inputs))
```

0

In [146]:
```python
perceptron.weights
```

Out[146]: array([-0.02, 0.01, 0.02])

In [274]:
```python

class Perceptron_OR(object):

    def __init__(self, no_of_inputs, threshold=10, learning_rate=0.01):
        self.threshold = threshold
        self.learning_rate = learning_rate
        self.weights = np.ones(no_of_inputs + 1)  ## including bias 2+1  = 3

    #Make prediction, here we are using a simple threshold of greater than z
    def predict(self, inputs):
        summ = np.dot(inputs, self.weights[1:]) + self.weights[0]  # self.we
        if summ >= 0:
            activation = 1
        else:
            activation = 0
        return activation

    def train(self, training_inputs, labels):
        for i in range(self.threshold):
            for inputs, label in zip(training_inputs, labels):
                prediction = self.predict(inputs)
                self.weights[1:] += self.learning_rate * (label - prediction

                self.weights[0] += self.learning_rate * (label - prediction)
            print("weights after iteration {} :{},new bias: {} ".format(i,se
        print("Final Weights :{},new bias: {} ".format(self.weights[1:],self
```

In [275]:
```python
inputs_or_gate= []
```

In [276]:
```python
training_inputs = []
training_inputs.append(np.array([0,0]))
training_inputs.append(np.array([0,1]))
training_inputs.append(np.array([1,0]))
training_inputs.append(np.array([1,1]))

#Adding Labels
labels_or = np.array([0,1,1,1])
```

In [277]:
```python
perceptron_or = Perceptron_OR(2)
```

In [278]:

```
perceptron_or.train(training_inputs,labels_or)
```

```
weights after iteration 0 :[1. 1.],new bias: 0.99
weights after iteration 1 :[1. 1.],new bias: 0.98
weights after iteration 2 :[1. 1.],new bias: 0.97
weights after iteration 3 :[1. 1.],new bias: 0.96
weights after iteration 4 :[1. 1.],new bias: 0.95
weights after iteration 5 :[1. 1.],new bias: 0.94
weights after iteration 6 :[1. 1.],new bias: 0.9299999999999999
weights after iteration 7 :[1. 1.],new bias: 0.9199999999999999
weights after iteration 8 :[1. 1.],new bias: 0.9099999999999999
weights after iteration 9 :[1. 1.],new bias: 0.8999999999999999
Final Weights :[1. 1.],new bias: 0.8999999999999999
```

In [279]:

```
inputs = np.array([1,0])
print(perceptron.predict(inputs))
```

```
0
```

In [283]:

```
import numpy as np
#np.random.seed(0)

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

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

#Input datasets
inputs = np.array([[0,0],[0,1],[1,0],[1,1]])
expected_output = np.array([[0],[1],[1],[1]])

epochs = 10000
lr = 0.1
inputLayerNeurons, hiddenLayerNeurons, outputLayerNeurons = 2,2,1

#Random weights and bias initialization
hidden_weights = np.random.uniform(size=(inputLayerNeurons,hiddenLayerNeuron
hidden_bias =np.random.uniform(size=(1,hiddenLayerNeurons))
output_weights = np.random.uniform(size=(hiddenLayerNeurons,outputLayerNeuro
output_bias = np.random.uniform(size=(1,outputLayerNeurons))

print("Initial hidden weights: ",end='')
print(*hidden_weights)
print("Initial hidden biases: ",end='')
print(*hidden_bias)
print("Initial output weights: ",end='')
print(*output_weights)
print("Initial output biases: ",end='')
print(*output_bias)


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

    output_layer_activation = np.dot(hidden_layer_output,output_weights)
    output_layer_activation += output_bias
    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_layer_ou

    #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
```

```python

print("Final hidden weights: ",end='')
print(*hidden_weights)
print("Final hidden bias: ",end='')
print(*hidden_bias)
print("Final output weights: ",end='')
print(*output_weights)
print("Final output bias: ",end='')
print(*output_bias)

print("\nOutput from neural network after 10,000 epochs: ",end='')
print(*predicted_output)

```

```
Initial hidden weights: [0.61147471 0.54163914] [0.86336028 0.05129989]
Initial hidden biases: [0.18525641 0.34895599]
Initial output weights: [0.02654494] [0.99419315]
Initial output biases: [0.95781594]
Final hidden weights: [2.73100448 4.06500869] [2.84785377 3.982446  ]
Final hidden bias: [-1.58970364 -2.17452362]
Final output weights: [3.57458172] [6.18610858]
Final output bias: [-4.35501231]

Output from neural network after 10,000 epochs: [0.04237271] [0.97686554] [0.97
654002] [0.99514686]
```

In [284]:

```python
np.random.uniform(size=(2,2))
```

Out[284]:

```
array([[0.85683785, 0.4696105 ],
       [0.28039111, 0.61509522]])
```

In [ ]:

```python

```