Name: Krish Sukhani

UID: 2018140059

Batch: D

Branch: IT

In [1]:

```
import numpy as np
class Perceptron(object):
    def __init__(self, n, c=0.01):
        self.c = c
        self.weights = np.zeros(n+1)

    def unipolar_train(self, training_inputs, labels):
        iter=0

        while(True):
            iter+=1
            pred = []
            for inputs, label in zip(training_inputs, labels):
                prediction = self.unipolar_predict(inputs)
                pred.append(prediction)
                self.weights[1:] += self.c*(label-prediction)*inputs
                self.weights[0] += self.c*(label-prediction)

            print(self.weights)
            if(labels==pred).all():
                print("Epochs : {}".format(iter))
                break

    def unipolar_predict(self, inputs):
        summation = np.dot(inputs, self.weights[1:]) + self.weights[0]
        #Unipolar Activation
        if summation>0:
            activation =1
        else:
            activation=0
        return activation
```

In [2]:

```
print("Unipolar - AND Gate")
x_train = np.array([[1,1], [1,0], [0,1], [0,0]])
perceptron = Perceptron(x_train.shape[1])
y_train = np.array([1,0,0,0])
perceptron.unipolar_train(x_train,y_train)

AND_prediction = []
for x in range(len(x_train)):
    AND_prediction.append(perceptron.unipolar_predict(x_train[x]))
print("AND Input : ", x_train)
print("Unipolar AND - Result : ", AND_prediction)
```

```
Unipolar - AND Gate
[-0.01  0.    0.  ]
[-0.01  0.    0.01]
[-0.02  0.    0.01]
[-0.02  0.01  0.01]
[-0.02  0.01  0.02]
[-0.02  0.01  0.02]
Epochs : 6
AND Input :  [[1 1]
 [1 0]
 [0 1]
 [0 0]]
Unipolar AND - Result :  [1, 0, 0, 0]
```

In [3]:

```
print("Unipolar - OR Gate")
x_train = np.array([[1,1], [1,0], [0,1], [0,0]])
perceptron = Perceptron(x_train.shape[1])
y_train = np.array([1,1,1,0])
perceptron.unipolar_train(x_train,y_train)

OR_prediction = []
for x in range(len(x_train)):
    OR_prediction.append(perceptron.unipolar_predict(x_train[x]))
print("OR Input : ", x_train)
print("Unipolar OR - Result : ", OR_prediction)
```

```
Unipolar - OR Gate
[0.   0.01 0.01]
[0.   0.01 0.01]
Epochs : 2
OR Input :  [[1 1]
 [1 0]
 [0 1]
 [0 0]]
Unipolar OR - Result :  [1, 1, 1, 0]
```

In [4]:

```
print("Unipolar - NOT Gate")
x_train = np.array([[1], [0]])
y_train = np.array([0, 1])
perceptron = Perceptron(x_train.shape[1])
perceptron.unipolar_train(x_train, y_train)

NOT_prediction = []
for x in range(len(x_train)):
    NOT_prediction.append(perceptron.unipolar_predict(x_train[x]))
print("NOT Input : ", x_train)
print("Unipolar NOT - Result : ", NOT_prediction)
```

```
Unipolar - NOT Gate
[0.01 0.  ]
[ 0.01 -0.01]
[ 0.01 -0.01]
Epochs : 3
NOT Input :  [[1]
 [0]]
Unipolar NOT - Result :  [0, 1]
```

In [5]:

```
import numpy as np
class Perceptron(object):
    def __init__(self, n, c=0.01):
        self.c = c
        self.weights = np.zeros(n+1)

    def bipolar_train(self, training_inputs, labels):
        iter=0

        while(True):
            iter+=1
            pred = []
            for inputs, label in zip(training_inputs, labels):
                prediction = self.bipolar_predict(inputs)
                pred.append(prediction)
                self.weights[1:] += self.c*(label-prediction)*inputs
                self.weights[0] += self.c*(label-prediction)

            print(self.weights)
            if(labels==pred).all():
                print("Epochs : {}".format(iter))
                break

    def bipolar_predict(self, inputs):
        summation = np.dot(inputs, self.weights[1:]) + self.weights[0]
        #Biipolar Activation
        if summation>0:
            activation =1
        else:
            activation = -1
        return activation
```

In [6]:

```
print("Bipolar - AND Gate")

x_train = np.array([[1,1], [1,-1], [-1,1], [-1,-1]])
perceptron = Perceptron(x_train.shape[1])
y_train = np.array([1,-1,-1,-1])
perceptron.bipolar_train(x_train,y_train)

AND_prediction = []
for x in range(len(x_train)):
    AND_prediction.append(perceptron.bipolar_predict(x_train[x]))
print("AND Input : ", x_train)
print("Bipolar AND - Result : ", AND_prediction)
```

```
Bipolar - AND Gate
[-0.02  0.02  0.02]
[-0.02  0.02  0.02]
Epochs : 2
AND Input :  [[ 1  1]
 [ 1 -1]
 [-1  1]
 [-1 -1]]
Bipolar AND - Result :  [1, -1, -1, -1]
```

In [7]:

```
print("Bipolar - OR Gate")
x_train = np.array([[1,1], [1,-1], [-1,1], [-1,-1]])
perceptron = Perceptron(x_train.shape[1])
y_train = np.array([1,1,1,-1])
perceptron.bipolar_train(x_train,y_train)

OR_prediction = []
for x in range(len(x_train)):
    OR_prediction.append(perceptron.bipolar_predict(x_train[x]))
print("OR Input : ", x_train)
print("Bipolar OR - Result : ", OR_prediction)
```

```
Bipolar - OR Gate
[0.02 0.02 0.02]
[0.02 0.02 0.02]
Epochs : 2
OR Input :  [[ 1  1]
 [ 1 -1]
 [-1  1]
 [-1 -1]]
Bipolar OR - Result :  [1, 1, 1, -1]
```

In [8]:

```
print("Bipolar - NOT Gate")
x_train = np.array([[1], [-1]])
y_train = np.array([-1, 1])
perceptron = Perceptron(x_train.shape[1])
perceptron.bipolar_train(x_train, y_train)

NOT_prediction = []
for x in range(len(x_train)):
    NOT_prediction.append(perceptron.bipolar_predict(x_train[x]))
print("NOT Input : ", x_train)
print("Bipolar NOT - Result : ", NOT_prediction)
```

```
Bipolar - NOT Gate
[ 0.02 -0.02]
[ 0.02 -0.02]
Epochs : 2
NOT Input :  [[ 1]
 [-1]]
Bipolar NOT - Result :  [-1, 1]
```

Conclusion :

The difference between the two activation functions

-> The threshold value that we use

Unipolar -- its 1 and 0

Bipolar its 1 and -1

-> A single neuron can be used to train and implement the AND, OR and NOT logic gates successfully.

->More complex logical gates will require more neurons or more layers of single neurons.