Name: Krish Sukhani

UID: 2018140059

Batch: D

Branch: IT

In [1]:

```
import numpy as np
```

In [2]:

```
def unipolar_activation(input):
    if input > 0:
        return 1
    else:
      return 0
```

In [3]:

```
alpha=0.05
```

In [4]:

```
def perceptron_activation(input, output_y, input_size):
    weights = np.zeros(input_size + 1)
    for iteration in range(0,50):
        for num, i in enumerate(input):
            unipolar_activation_output = unipolar_activation(np.sum(i * w
eights[1:]) + weights[0])
            difference = output_y[num] - unipolar_activation_output
            weights[1:] += difference*alpha*i
            weights[0] += difference*alpha
    output = np.array([])
    print('The weights are : \n \n', weights)
    for i in input:
        unipolar_activation_output = unipolar_activation(np.sum(i * weigh
ts[1:]) + weights[0])
        output = np.append(output, [unipolar_activation_output])
    difference = output_y - unipolar_activation_output
    return output
```

In [5]:

```
input = [[0,0], [0,1], [1,0], [1,1]]
```

In [6]:

```
output = [0, 0, 0, 1]
```

In [7]:

```
print("-----AND-----")
print(perceptron_activation(np.array(input), np.array(output), 2))
```

```
-----AND-----
The weights are :

 [-0.1   0.1   0.05]
[0. 0. 0. 1.]
```

In [8]:

```
input = [[0,0], [0,1], [1,0], [1,1]]
```

In [9]:

```
output = [0, 1, 1, 1]
```

In [10]:

```
print("-----OR-----")
print(perceptron_activation(np.array(input), np.array(output), 2))
```

```
-----OR-----
The weights are :

 [0.   0.05 0.05]
[0. 1. 1. 1.]
```

In [11]:

```
input = [[0], [1]]
```

In [12]:

```
output = [1, 0]
```

In [13]:

```
print("-----NOT-----")
print(perceptron_activation(np.array(input), np.array(output), 1))
```

```
-----NOT-----
The weights are :

 [ 0.05 -0.05]
[1. 0.]
```

In [14]:

```
def bipolar_activation(input):
    if input > 0:
        return 1
    else:
      return 1
```

In [15]:

```
alpha=0.05
```

In [16]:

```
def perceptron_activation_bipolar(input, output_y, input_size):
    weights = np.zeros(input_size + 1)
    for iteration in range(0,50):
        for num, i in enumerate(input):
            op = bipolar_activation(np.sum(i * weights[1:]) + weights[0])
            difference = output_y[num] - op
            weights[1:] += difference*alpha*i
            weights[0] += difference*alpha
    output = np.array([])
    print('The weights are : \n \n', weights)
    for i in input:
        op = bipolar_activation(np.sum(i * weights[1:]) + weights[0])
        output = np.append(output, [op])
    difference = output_y - op
    return output
```

In [17]:

```
input = [[-1, 1], [-1,1], [1,-1], [1,1]]
```

In [18]:

```
output = [-1, -1, -1, 1]
```

In [19]:

```
print("-----AND-----")
print(perceptron_activation_bipolar(np.array(input), np.array(output), 2
))
```

```
-----AND-----
The weights are :

 [-15.   5.  -5.]
[1. 1. 1. 1.]
```

In [20]:

```
input = [[-1,-1], [-1,1], [1,-1], [1,1]]
```

In [21]:

```
output = [-1, 1, 1, 1]
```

In [22]:

```
print("-----OR-----")
print(perceptron_activation_bipolar(np.array(input), np.array(output), 2
))
```

```
-----OR-----
The weights are :

 [-5.  5.  5.]
[1. 1. 1. 1.]
```

In [23]:

```
input = [[-1], [1]]
```

In [24]:

```
output = [1, -1]
```

In [25]:

```
print("-----NOT-----")
print(perceptron_activation_bipolar(np.array(input), np.array(output), 1
))
```

```
-----NOT-----
The weights are :

 [-5. -5.]
[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.