In [1]:

```
import numpy as np
```

In [2]:

```
input_value = np.array([[0,0],[0,1],[1,1],[1,0]])
print(input_value.shape)
input_value
```

```
(4, 2)
```

Out[2]:

```
array([[0, 0],
       [0, 1],
       [1, 1],
       [1, 0]])
```

In [3]:

```
output = np.array([0,1,1,0])
output = output.reshape(4,1)
output.shape
```

Out[3]:

```
(4, 1)
```

In [4]:

```
weights = np.array([[0.1],[0.2]])
weights
```

Out[4]:

```
array([[0.1],
       [0.2]])
```

In [5]:

```
bias = 0.3
```

In [6]:

```
# activation function
```

In [7]:

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

In [8]:

```
def der(x):
    return sigmoid_func(x)*(1-sigmoid_func(x))
```

In [9]:

```
#udating the weights
```

In [10]:

```
for epochs in range(10000):
    input_arr = input_value

    weighted_sum = np.dot(input_arr,weights)+bias
    first_output = sigmoid_func(weighted_sum)

    error = first_output - output # error in the prediction
    total_error = np.square(np.subtract(first_output,output).mean())# ((sum of all error)/n
    #print(total_error)
    first_der = error
    second_der = der(first_output)
    derivative = first_der*second_der

    t_input = input_value.T # transpose
    final_derivative = np.dot(t_input,derivative)

# update weights
weights= weights - 0.05*final_derivative

#update bias
for i in derivative:
    bias = bias - 0.05 * i
print(weights)
print(bias)
```

```
[[0.09714683]
 [0.20828807]]
[0.29481765]
```

In [11]:

```
#predictions
pred = np.array([0,1])
result = np.dot(pred,weights)+bias
res = sigmoid_func(result)
print(res)
```

```
[0.62318891]
```