In [11]:
```
import numpy as np

X = np.array([0,1])
y = 1

alpha1 = np.array(([0.1,0.3]), dtype=float)
alpha2 = np.array(([0.3,0.4]), dtype=float)
beta = np.array(([0.4,0.6]), dtype=float)
learning_rate = 0.01
def sigmoid(x):
    return 1/(1+np.exp(-x))
def sigmoidPrime(z):
    return np.exp(-z)/((1+np.exp(-z))**2)
def costFunction(y, yHat):
    J = (y-yHat)**2
    return J
def costFunction_prime(y, yHat):
    #derivative corresponds to y_hat
    return -2*(y-yHat)

def compute_y_hat(alpha1, alpha2, beta, X, y):
    input_for_z1 = np.dot(alpha1, X)
    z1= sigmoid(input_for_z1)
    input_for_z2 = np.dot(alpha2, X)
    z2= sigmoid(input_for_z2)
    z = np.array([z1,z2])
    output_for_z = np.dot(z,beta)
    y_hat = sigmoid(output_for_z)
    return y_hat

def compute_y_hat_prime_beta(alpha1, alpha2, beta, X, y):
    y_hat = compute_y_hat(alpha1, alpha2, beta, X, y)
    y_hat_prime = costFunction_prime(y, yHat)

    input_for_z1 = np.dot(alpha1, X)
    z1= sigmoid(input_for_z1)
    input_for_z2 = np.dot(alpha2, X)
    z2= sigmoid(input_for_z2)
    z = np.array([z1,z2])
    betaT_z_prime = sigmoidPrime(z)

    beta_prime1 = y_hat_prime*betaT_z_prime*z[0]
    beta_prime2 = y_hat_prime*betaT_z_prime*z[1]
    return beta_prime1, beta_prime2

def compute_y_hat_prime_beta(alpha1, alpha2, beta, X, y):
    y_hat = compute_y_hat(alpha1, alpha2, beta, X, y)
    y_hat_prime = costFunction_prime(y, y_hat)
    input_for_z1 = np.dot(alpha1, X)
    z1= sigmoid(input_for_z1)
    input_for_z2 = np.dot(alpha2, X)
    z2= sigmoid(input_for_z2)
    z = np.array([z1,z2])

    betaT_z_prime = sigmoidPrime(np.dot(z,beta))
    beta_prime1 = y_hat_prime*betaT_z_prime*z[0]
```

```
    beta_prime2 = y_hat_prime*betaT_z_prime*z[1]
    return beta_prime1, beta_prime2

def compute_y_hat_prime_alpha(alpha1, alpha2, beta, X, y):
    y_hat = compute_y_hat(alpha1, alpha2, beta, X, y)
    y_hat_prime = costFunction_prime(y, y_hat)

    input_for_z1 = np.dot(alpha1, X)
    z1= sigmoid(input_for_z1)
    input_for_z2 = np.dot(alpha2, X)
    z2= sigmoid(input_for_z2)
    z = np.array([z1,z2])

    betaT_z_prime = sigmoidPrime(np.dot(z,beta))

    alpha_prime11 = y_hat_prime*betaT_z_prime*beta[0]*sigmoidPrime(input_fo
    alpha_prime12 = y_hat_prime*betaT_z_prime*beta[0]*sigmoidPrime(input_fo
    alpha_prime21 = y_hat_prime*betaT_z_prime*beta[1]*sigmoidPrime(input_fo
    alpha_prime22 = y_hat_prime*betaT_z_prime*beta[1]*sigmoidPrime(input_fo

    return alpha_prime11,alpha_prime12,alpha_prime21,alpha_prime22
def update_beta(alpha1, alpha2, beta, X, y, learning_rate):
    beta_prime1, beta_prime2 = compute_y_hat_prime_beta(alpha1, alpha2, bet
    beta_prime = np.array([beta_prime1, beta_prime2])
    beta_update = beta-learning_rate*beta_prime
    return beta_update
def update_alpha1(alpha1, alpha2, beta, X, y, learning_rate):
    alpha_prime11,alpha_prime12,alpha_prime21,alpha_prime22 = compute_y_hat
    alpha1_prime = np.array([alpha_prime11,alpha_prime12])
    alpha_update = alpha1-learning_rate*alpha1_prime
    return alpha_update

def update_alpha2(alpha1, alpha2, beta, X, y, learning_rate):
    alpha_prime11,alpha_prime12,alpha_prime21,alpha_prime22 = compute_y_hat
    alpha2_prime = np.array([alpha_prime21,alpha_prime22])
    alpha_update = alpha2-learning_rate*alpha2_prime
    return alpha_update

def main_function(alpha1, alpha2, beta, X, y, learning_rate):
    while y-compute_y_hat(alpha1, alpha2, beta, X, y)>0.01:
        alpha1 = update_alpha1(alpha1, alpha2, beta, X, y, learning_rate)
        alpha2 = update_alpha2(alpha1, alpha2, beta, X, y, learning_rate)
        beta = update_beta(alpha1, alpha2, beta, X, y, learning_rate)
    return compute_y_hat(alpha1, alpha2, beta, X, y)
```

In [12]:
```
main_function(alpha1, alpha2, beta, X, y, learning_rate)
```

Out[12]:
```
0.9900000229217649
```

In [ ]:
```

```