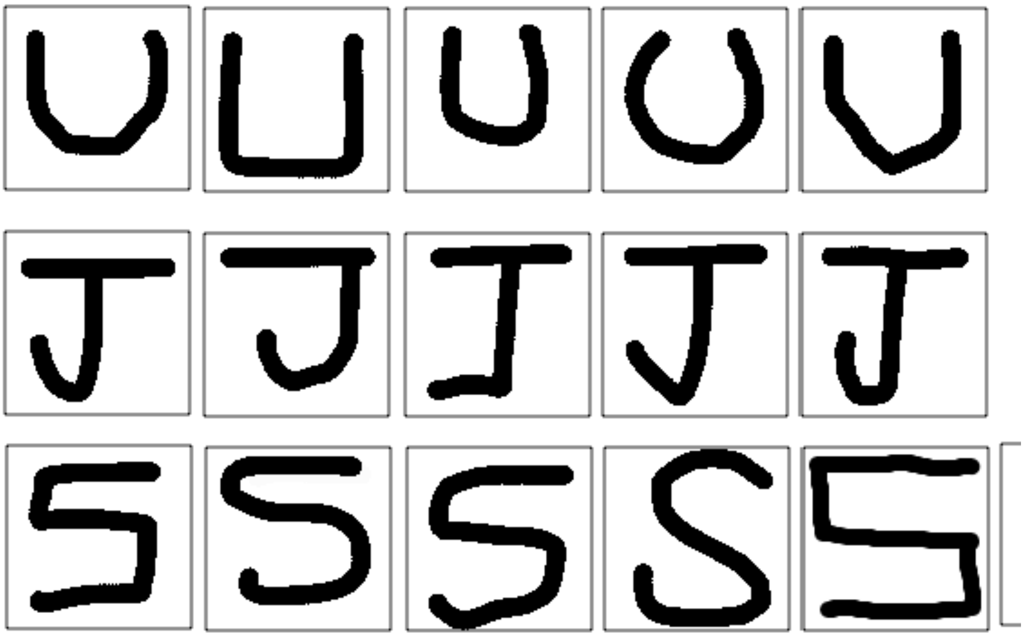
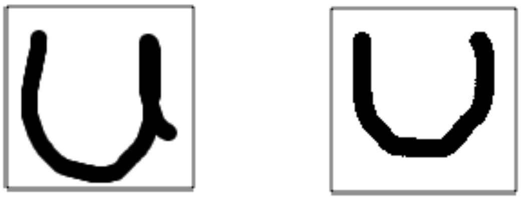
ASSIGNMENT 1 OF N.F.T

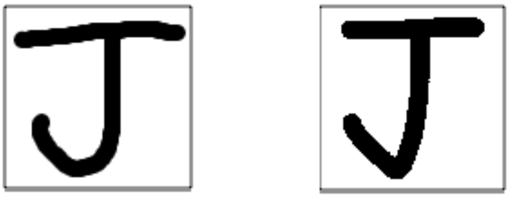
Activation Function Used - Sigmoidal Function = 1/(1+exp(-x))

Images Of inputs

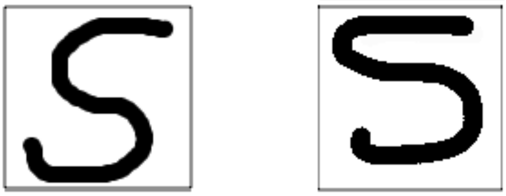




u1



j4



s2

Corresponding image vectors

```
# input u

u1 = [[0,0,0,0,0,0,0,0,0,0],
      [0,1,0,0,0,0,0,1,1,0],
      [0,1,0,0,0,0,0,1,1,0],
      [0,1,0,0,0,0,0,0,1,0],
      [0,1,0,0,0,0,0,0,1,0],
      [0,1,1,0,0,0,0,1,1,0],
      [0,1,1,0,0,0,0,1,1,0],
      [0,0,1,1,1,1,1,1,0,0],
      [0,0,0,0,0,0,0,0,0,0],
      [0,0,0,0,0,0,0,0,0,0]]

u2 = [[0,0,0,0,0,0,0,0,0,0],
      [0,1,0,0,0,0,0,1,1,0],
      [0,1,0,0,0,0,0,1,1,0],
      [1,1,0,0,0,0,0,1,1,0],
      [1,1,0,0,0,0,0,1,1,0],
      [1,1,0,0,0,0,0,1,1,0],
      [1,1,0,0,0,0,0,1,1,0],
      [1,1,0,0,0,0,0,1,1,0],
      [0,1,1,1,1,1,1,1,1,0],
      [0,0,1,1,1,1,1,1,0,0]]

u3 = [[0,0,0,0,0,0,0,0,0,0],
      [0,0,1,0,0,0,1,1,0,0],
      [0,0,1,0,0,0,1,1,0,0],
      [0,0,1,0,0,0,0,1,0,0],
      [0,0,1,0,0,0,0,1,0,0],
      [0,0,1,0,0,0,0,1,0,0],
	[0,0,1,1,1,1,1,1,0,0],
	[0,0,0,1,1,1,1,0,0,0],
	[0,0,0,0,0,0,0,0,0,0],
	[0,0,0,0,0,0,0,0,0,0]
]

u4 = [[0,0,0,0,0,0,0,0,0,0],
	[0,0,1,1,0,0,1,1,0,0],
	[0,1,1,1,0,0,1,1,1,0],
	[0,1,1,0,0,0,0,1,1,0],
	[0,1,0,0,0,0,0,0,1,0],
	[0,1,1,0,0,0,0,1,1,0],
	[0,1,1,0,0,0,0,1,1,0],
	[0,0,1,1,1,1,1,1,0,0],
	[0,0,0,1,1,1,1,0,0,0],
	[0,0,0,0,0,0,0,0,0,0]]

u5 = [[0,0,0,0,0,0,0,0,0,0],
	[0,1,0,0,0,0,0,1,1,0],
	[0,1,0,0,0,0,0,1,1,0],
	[0,1,0,0,0,0,0,1,1,0],
	[0,1,0,0,0,0,0,1,1,0],
	[0,1,1,0,0,0,0,1,1,0],
	[0,0,1,1,0,0,0,1,1,0],
	[0,0,1,1,1,0,1,1,1,0],
	[0,0,0,1,1,1,1,1,0,0],
	[0,0,0,0,0,0,0,0,0,0]
]



# input j
```

```
j1 = [[0,0,0,0,0,0,0,0,0,0],
      [0,0,0,0,0,0,0,0,0,0],
      [0,1,1,1,1,1,1,1,1,0],
      [0,0,0,0,1,1,0,0,0,0],
      [0,0,0,0,1,1,0,0,0,0],
      [0,0,0,0,1,1,0,0,0,0],
      [0,1,0,0,1,1,0,0,0,0],
      [0,1,1,0,1,1,0,0,0,0],
      [0,1,1,1,1,1,0,0,0,0],
      [0,0,1,1,1,0,0,0,0,0],
]

j2 = [[0,0,0,0,0,0,0,0,0,0],
      [0,1,1,1,1,1,1,1,1,0],
      [0,1,1,1,1,1,1,1,1,0],
      [0,0,0,0,0,0,0,1,1,0],
      [0,0,0,0,0,0,0,1,1,0],
      [0,0,0,1,0,0,0,1,1,0],
      [0,0,0,1,0,0,0,1,1,0],
      [0,0,0,1,1,0,0,1,1,0],
      [0,0,0,1,1,1,1,1,0,0],
      [0,0,0,0,0,0,0,0,0,0]
]

j3 = [[0,0,0,0,0,0,0,0,0,0],
      [0,1,1,1,1,1,1,1,1,0],
      [0,1,0,0,0,1,1,0,0,0],
      [0,0,0,0,0,1,1,0,0,0],
      [0,0,0,0,0,1,1,0,0,0],
      [0,0,0,0,0,1,1,0,0,0],
      [0,0,0,0,0,1,1,0,0,0],
      [0,0,0,0,0,1,1,0,0,0],
      [0,1,1,1,1,1,0,0,0,0],
      [0,1,1,1,1,1,0,0,0,0]
  ]

  j4 = [[0,0,0,0,0,0,0,0,0,0],
      [0,1,1,1,1,1,1,1,1,0],
      [0,0,0,0,0,1,0,0,0,0],
      [0,0,0,0,0,1,0,0,0,0],
      [0,0,0,0,0,1,0,0,0,0],
      [0,0,0,0,0,1,0,0,0,0],
      [0,1,0,0,0,1,0,0,0,0],
      [0,1,1,0,0,1,0,0,0,0],
      [0,0,1,1,1,1,0,0,0,0],
      [0,0,0,1,1,0,0,0,0,0],
  ]

  j5 = [[0,0,0,0,0,0,0,0,0,0],
      [0,1,1,1,1,1,1,1,1,0],
      [0,1,1,1,1,1,1,1,1,0],
      [0,0,0,0,1,1,0,0,0,0],
      [0,0,0,0,1,1,0,0,0,0],
      [0,0,0,0,1,1,0,0,0,0],
      [0,0,1,0,1,1,0,0,0,0],
      [0,0,1,0,1,1,0,0,0,0],
      [0,0,1,0,1,1,0,0,0,0],
      [0,0,1,1,1,0,0,0,0,0]
  ]

  # input s

  s1 = [[0,0,1,1,1,1,1,1,1,0],
      [0,1,1,1,1,1,1,1,1,0],
      [1,1,0,0,0,0,0,0,0,0],
      [0,1,1,1,1,1,1,0,0,0],
      [0,0,0,1,1,1,1,1,1,0],
      [0,0,0,0,0,0,0,0,1,0],
      [0,0,0,0,0,0,0,0,1,0],
      [0,0,1,0,0,0,0,1,1,0],
      [0,0,1,1,1,1,1,1,0,0],
      [0,0,0,0,0,0,0,0,0,0],
]

s2 = [[0,0,0,0,0,0,0,0,0,0],
      [0,0,1,1,1,1,1,1,1,0],
      [0,1,1,1,1,1,1,1,1,0],
      [0,1,1,0,0,0,0,0,0,0],
      [0,1,1,1,1,1,1,1,0,0],
      [0,0,1,1,1,1,1,1,1,0],
      [0,0,0,0,0,0,0,0,1,0],
      [0,0,0,0,0,0,0,0,1,0],
      [0,1,0,0,0,1,1,1,1,0],
      [0,1,1,1,1,1,1,1,0,0],


]

s3 = [[0,0,0,0,1,1,1,1,0,0],
      [0,0,1,1,1,1,1,1,1,0],
      [0,0,1,1,0,0,0,0,1,0],
      [0,0,1,1,0,0,0,0,0,0],
      [0,0,0,1,1,0,0,0,0,0],
      [0,0,0,0,1,1,1,0,0,0],
      [0,0,1,0,0,1,1,1,1,0],
      [0,1,1,0,0,0,0,1,1,0],
      [0,1,1,0,0,0,0,0,1,0],
      [0,0,1,1,1,1,1,1,1,0]
]
```

```
s4 = [[0,0,0,0,1,1,1,1,0,0],
      [0,0,0,1,1,1,1,1,1,0],
      [0,0,1,1,0,0,0,0,1,1],
      [0,0,1,1,0,0,0,0,0,0],
      [0,0,0,1,1,0,0,0,0,0],
      [0,0,0,0,1,1,1,1,0,0],
      [0,1,1,0,0,1,1,1,1,0],
      [0,1,1,0,0,0,0,1,1,0],
      [0,1,1,0,0,0,0,0,1,0],
      [0,0,1,1,1,1,1,1,1,0]
     ]

s5 = [[1,1,1,1,1,1,1,1,1,1],
      [1,1,1,1,1,1,1,1,1,1],
      [1,1,0,0,0,0,0,0,0,0],
      [1,1,0,0,0,0,0,0,0,0],
      [1,1,1,1,0,0,0,0,0,0],
      [1,1,1,1,1,1,1,1,1,1],
      [0,0,0,0,0,0,0,0,1,1],
      [0,0,0,0,0,0,0,0,1,1],
      [0,0,0,0,0,0,0,0,0,1],
      [1,1,1,1,1,1,1,1,1,1]
]

Utest = [[0,0,0,0,0,0,0,0,0,0],
         [0,1,0,0,0,0,0,1,0,0],
         [0,1,0,0,0,0,0,1,1,0],
         [0,1,0,0,0,0,0,1,1,0],
         [1,1,0,0,0,0,0,1,1,0],
         [1,1,0,0,0,0,0,1,1,0],
         [1,1,0,0,0,0,0,1,1,0],
         [0,1,1,0,0,0,0,1,1,0],
         [0,1,1,1,1,1,1,1,0,0],
         [0,0,1,1,1,1,1,0,0,0]]
Jtest = [[0,0,0,0,0,0,0,0,0,0],
         [1,1,1,1,1,1,1,1,1,1],
         [1,1,0,0,0,1,1,0,0,0],
         [0,0,0,0,0,1,1,0,0,0],
         [0,0,0,0,0,1,1,0,0,0],
         [0,0,0,0,0,1,1,0,0,0],
         [0,1,1,0,0,1,1,0,0,0],
         [0,1,1,0,0,1,1,0,0,0],
         [0,0,1,1,1,1,0,0,0,0],
         [0,0,0,1,1,0,0,0,0,0]]
sTest = [[0,0,0,0,1,1,1,1,1,1],
         [0,0,0,1,1,1,1,1,1,1],
         [0,0,1,1,0,0,0,0,0,0],
         [0,0,1,1,0,0,0,0,0,0],
         [0,0,1,1,1,1,0,0,0,0],
         [0,0,0,1,1,1,1,1,0,0],
         [0,0,0,0,0,0,0,1,0,0],
         [0,1,0,0,0,0,1,1,0,0],
         [0,1,1,1,1,1,1,1,0,0],
         [0,0,1,1,1,1,1,0,0,0]]
 input_test = [[utest,u1],[jtest,j4],[stest,s2]]
```

CODE

Basic Code For All

```
import numpy as np
from numpy import exp
np.random.seed(0)
```

```
class Linearlayer():
    def __init__(self,n_inp,n_out):
        self.weights = np.random.randn(n_inp,n_out)
        self.bias = np.zeros((1,n_out))
    def forward(self,inputs):
        self.output = np.dot(inputs,self.weights) + self.bias
```

For 1 hidden Layer

Code (in Python)

```
class NeuralNet():
    def __init__(self,n_inp,n_out,alpha):
        self.inp = n_inp
        self.out = n_out
        self.hidd_no = 50
        self.alpha = alpha
        self.error = 1
        self.layer1 = Linearlayer(n_inp,self.hidd_no)
        self.layer2 = Linearlayer(self.hidd_no,n_out)

    def act_fun(self,x):
        return 1/(1+exp(-x))
    def der_act_fun(self,x):
        x = self.act_fun(x)
        return x*(1-x)

    def forward(self,input_set):
        self.input_set = input_set
        self.layer1.forward(input_set)
        self.inp_hidden = self.act_fun(self.layer1.output)
        self.layer2.forward(self.inp_hidden)
        self.fout = self.act_fun(self.layer2.output)
        return self.fout
    def learn(self,input_set,output_set):
        nnout = self.forward(input_set)
#          print("output is - ",nnout)
        self.error = 0
        for i in range(len(output_set)):
            self.error+=(output_set[i]-nnout[0][i])**2
        self.error/=2
#          print("error - ",error)
        self.backpropgatel1(output_set)
        self.backpropgatel2()
    def backpropgatel1(self,output):
        self.errorhid = []
        xins = self.inp_hidden[0]
        yout = self.fout[0]
        yins = self.layer2.output[0]
        for i in range(len(xins)):
            for j in range(len(yout)):
                diff = -1*(output[j]-yout[j])*self.der_act_fun(yins[j])
                chw = diff*xins[i]
#                  print("chw at ",i,j," is",chw)
                self.errorhid.append(diff)
                self.layer2.weights[i][j]-= self.alpha*chw
        for j in range(len(yout)):
            self.layer2.bias[0][j]-=self.alpha*self.errorhid[j]
    def backpropgatel2(self):
        for i in range(self.inp):
            for j in range(self.hidd_no):
                cng = 0
                for k in range(len(self.layer2.weights[j])):
                    cng += self.errorhid[k]*self.layer2.weights[j][k]
                cng *= self.der_act_fun(self.layer1.output[0][j])*self.input_set[i]
                self.layer1.weights[i][j]-=self.alpha*cng
        for j in range(self.hidd_no):
            cng = 0
            for k in range(len(self.layer2.weights[j])):
                cng += self.errorhid[k]*self.layer2.weights[j][k]
            cng *= self.der_act_fun(self.layer1.output[0][j])
            self.layer1.bias[0][j]-=self.alpha*cng
    def which_class(self):
        lst = list(self.fout[0])
        index = lst.index(max(lst))
        print(index+1)
```

For hidden 2 Layer

```
class NeuralNet2():
    def __init__(self,n_inp,n_out,alpha):
        self.inp = n_inp
        self.out = n_out
        self.hidd_no1 = 50
        self.hidd_no2 = 10
        self.alpha = alpha
        self.error = 1
        self.layer1 = Linearlayer(n_inp,self.hidd_no1)
        self.layer2 = Linearlayer(self.hidd_no1,self.hidd_no2)
        self.layer3 = Linearlayer(self.hidd_no2,n_out)

    def act_fun(self,x):
        return 1/(1+exp(-x))
    def der_act_fun(self,x):
        x = self.act_fun(x)
        return x*(1-x)

    def forward(self,input_set):
        self.input_set = input_set
        self.layer1.forward(input_set)
        self.inp_hidden1 = self.act_fun(self.layer1.output)
        self.layer2.forward(self.inp_hidden1)
        self.inp_hidden2 = self.act_fun(self.layer2.output)
        self.layer3.forward(self.inp_hidden2)
        self.fout = self.act_fun(self.layer3.output)
        return self.fout
    def learn(self,input_set,output_set):
        nnout = self.forward(input_set)
#         print("output is - ",nnout)
        self.error = 0
        for i in range(len(output_set)):
            self.error+=(output_set[i]-nnout[0][i])**2
        self.error/=2
#         print("error - ",error)
        self.backpropgatel1(output_set)
        self.backpropgatel2()
        self.backpropgatel3()
    def backpropgatel1(self,output):
        self.errorhid1 = []
        xins = self.inp_hidden2[0]
        yout = self.fout[0]
        yins = self.layer3.output[0]
        for i in range(len(xins)):
            for j in range(len(yout)):
                diff = -1*(output[j]-yout[j])*self.der_act_fun(yins[j])
                chw = diff*xins[i]
#                  print("chw at ",i,j," is",chw)
                self.errorhid1.append(diff)
                self.layer3.weights[i][j]-= self.alpha*chw
        for j in range(len(yout)):
            self.layer3.bias[0][j]-=self.alpha*self.errorhid1[j]
    def backpropgatel2(self):
        self.errorhid2 = []
        for i in range(self.hidd_no1):
            for j in range(self.hidd_no2):
                cng = 0
                for k in range(len(self.layer3.weights[j])):
                    cng += self.errorhid1[k]*self.layer3.weights[j][k]
                diff = cng*self.der_act_fun(self.layer2.output[0][j])
                self.errorhid2.append(diff)
                cng=diff*self.inp_hidden1[0][i]
                self.layer2.weights[i][j]-=self.alpha*cng

        for j in range(self.hidd_no2):
            self.layer2.bias[0][j]-=self.alpha*self.errorhid2[j]

    def backpropgatel3(self):
        self.errorhid3 = []
        for i in range(self.inp):
            for j in range(self.hidd_no1):
                cng = 0
                for k in range(len(self.layer2.weights[j])):
                    cng+=self.errorhid2[k]*self.layer2.weights[j][k]
                diff = cng*self.der_act_fun(self.layer1.output[0][j])
                self.errorhid3.append(diff)
                cng = diff*self.input_set[i]
                self.layer1.weights[i][j]-=self.alpha*cng
            for j in range(self.hidd_no1):
                self.layer1.bias[0][j]-=self.alpha*self.errorhid3[j]
    def which_class(self):
        lst = list(self.fout[0])
        index = lst.index(max(lst))
        print(index+1)
```

For Single hidden layer Neural network

Input = 100
Output = 3
hidden_neurons = 50

Note : All are trained for 100 epochs

- Alpha = 0.01
  - Error after 100 epoch = error is - 0.1992105937590337
  - Predicted correct for j and s and predicted u as j and predicted correct for all 3 trained inputs (83.33% (5/6) efficiency)
- Alpha = 0.05
  - Error after 100 epoch = error is - 0.002153407458248942
  - Predicted correct output for all 3 test inputs and 3 trained inputs (100% efficiency)
- Alpha = 0.1
  - Error after 100 epoch = error is - 0.002019482160076674
  - Predicted correct output for all 3 test inputs and 3 trained inputs (100% efficiency)
- Alpha = 0.2
  - Error after 100 epoch =error is - 0.0010556727701782128
  - Predicted correct output for all 3 test inputs and 3 trained inputs (100% efficiency)
- Alpha = 0.4
  - Error after 100 epoch = error is - 0.00026034047380031747 (converged quickly)
  - Predicted correct output for all 3 test inputs and 3 trained inputs (100% efficiency)
- Alpha = 0.8
  - Error after 100 epoch =error is - 0.00015053401174070942 (converged very quickly)
  - Predicted correct output for all 3 test inputs and 3 trained inputs (100% efficiency)

Outputs

ALPHA = 0.01

U1 = array([[0.01772885, 0.13619307, 0.09615003]])

uTest = array([[0.0007731 , 0.06295517, 0.05880396]])

J4 = array([[0.0784517 , 0.92336794, 0.23808348]])

jtest = array([[0.43880883, 0.89569099, 0.03346722]])

S2 = array([[0.02714864, 0.11810099, 0.1553307]])

stest = array([[0.1830266 , 0.01438707, 0.97828183]])

ALPHA = 0.05

U1 = array([[0.9621225 , 0.03167795, 0.0257187]])

uTest = array([[0.8474327 , 0.09403217, 0.04622646]])

J4 = array([[0.02091829, 0.98307875, 0.02984112]])

jtest = array([[0.09110888, 0.56132816, 0.05741785]])

```
S2 = array([[0.03163939, 0.1133088 , 0.89158344]])

stest = array([[0.01558436, 0.19135153, 0.99346636]])
```

ALPHA = 0.1

```
U1 = array([[0.96580879, 0.02113358, 0.0017923 ]])

uTest = array([[0.97261992, 0.00423807, 0.48793959]])

J4 = array([[0.03669785, 0.96575974, 0.04366001]])

jtest = array([[0.00971823, 0.91681462, 0.01639753]])

S2 = array([[0.06804528, 0.04928206, 0.94049755]])

stest = array([[0.42553386, 0.01481461, 0.97068732]])
```

ALPHA = 0.2

```
U1 = array([[0.97517681, 0.02024575, 0.02171927]])

uTest = array([[0.28482741, 0.14000225, 0.03947411]])

J4 = array([[0.00940287, 0.96468571, 0.03081901]])

jtest = array([[0.02617459, 0.98487596, 0.05299596]])

S2 = array([[0.04576887, 0.00476103, 0.95665998]])

stest = array([[0.01091105, 0.16071301, 0.75531609]])
```

ALPHA = 0.4

```
U1 = array([[0.99668737, 0.01210557, 0.01838829]])

uTest = array([[0.83027372, 0.00667054, 0.02497717]])

J4 = array([[3.46298529e-04, 9.99234806e-01, 2.03777527e-02]])

jtest = array([[7.48432699e-04, 9.03936565e-01, 1.81905874e-01]])
```

```
S2 = array([[0.01622731, 0.02203935, 0.98218355]])

stest = array([[0.12111521, 0.93368645, 0.23008061]])
ALPHA = 0.8
U1 = array([[0.99796456, 0.01530609, 0.00472475]])

uTest = array([[0.97795404, 0.00406444, 0.01031407]])

J4 = array([[7.66155666e-04, 9.89956217e-01, 4.11356541e-03]])

jtest = array([[0.02631317, 0.97005967, 0.07347251]])

S2 = array([[0.00272027, 0.01354684, 0.97415128]])

stest = array([[3.72717535e-03, 2.72993753e-04, 9.06605493e-01]])
```

For 2 Hidden Layers

Input = 100
Output = 3
hidden_neurons in layer 1 = 50
Hidden neutrons in layer 2 = 10

Note : All are trained for 100 epochs

- Alpha = 0.01
  - Error after 100 epoch =error is - 0.0010043579114966902
  - Predicted correct output for all 3 test inputs and 3 trained inputs (100% efficiency)
- Alpha = 0.05
  - Error after 100 epoch =error is - 0.04587529206998059
  - Predicted correct output for all 3 test inputs and 3 trained inputs (100% efficiency)
- Alpha = 0.1
  - Error after 100 epoch = error is - 0.020211443065898664
  - Predicted correct output for all 3 test inputs and 3 trained inputs (100% efficiency)
- Alpha = 0.2
  - Error after 100 epoch =error is - 0.008004867423238407 (converged quickly)
  - Predicted correct output for all 3 test inputs and 3 trained inputs (100% efficiency)
- Alpha = 0.4
  - Error after 100 epoch =error is - 0.003612699556835002 (converged quickly)
  - Predicted correct output for all 3 test inputs and 3 trained inputs (100% efficiency)
- Alpha = 0.8
  - Error after 100 epoch =error is - 0.0008186011433955137 (again converged very quickly)
  - Predicted correct output for all 3 test inputs and 3 trained inputs (100% efficiency)

ALPHA = 0.01

```
U1 =[[0.99896702 0.00619659 0.00274966]]

uTest =[[0.99302617 0.0016544  0.00367683]]

J4 =[[3.59476441e-04 9.95706442e-01 1.92523475e-03]]

jtest =[[0.01591234 0.98575458 0.03556903]]

S2 =[[0.00123883 0.00753557 0.98955168]]

stest =[[3.94803073e-03 1.76200151e-04 9.29357319e-01]]
```

ALPHA = 0.05

```
U1 =[[0.99896702 0.00619659 0.00274966]]

uTest =[[0.99302617 0.0016544  0.00367683]]

J4 =[[3.59476441e-04 9.95706442e-01 1.92523475e-03]]

jtest =[[0.01591234 0.98575458 0.03556903]]

S2 =[[0.00123883 0.00753557 0.98955168]]
```

```
stest =[[3.94803073e-03 1.76200151e-04 9.29357319e-01]]

ALPHA = 0.1
U1 =[[0.88276722 0.10501336 0.05018982]]
uTest =[[0.90939902 0.1593211  0.09706815]]
J4 =[[0.05755287 0.82782483 0.05679039]]
jtest =[[0.06107836 0.78237743 0.05764118]]
S2 =[[0.0826416  0.08274429 0.96560414]]
stest =[[0.27847572 0.11558183 0.2920963 ]]

ALPHA = 0.2
U1 =[[0.87366494 0.0855708  0.01815406]]
uTest =[[0.8842864  0.14832468 0.01091981]]
J4 =[[0.14087747 0.87487268 0.06902822]]
jtest =[[0.12581729 0.88110337 0.07548734]]
S2 =[[0.0452707  0.08114755 0.91918825]]
stest =[[0.03981503 0.12466144 0.91824328]]

ALPHA = 0.4
U1 =[[0.96307876 0.04316516 0.03537255]]
uTest =[[0.92341607 0.05154423 0.12429239]]
J4 =[[0.01069722 0.96615217 0.04228547]]
jtest =[[0.00447994 0.92501428 0.10443163]]
S2 =[[0.03496    0.01274715 0.95689745]]
stest =[[0.10493992 0.01623718 0.86813372]]

ALPHA = 0.8
U1 =[[0.96736698 0.0306476  0.01274288]]
uTest =[[0.96988046 0.02797436 0.01674648]]
J4 =[[0.03335234 0.96806431 0.01445435]]
```

```
jtest =[[0.04931626 0.79817694 0.01474549]]

S2 =[[0.01827854 0.01706512 0.97736233]]

stest =[[0.04075552 0.22152476 0.88947028]]
```