Prac5

```
import numpy as np
from matplotlib.pylab import randn
from math import exp

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

def sigmoid_derivative(x):
    return x * (1-x)

class NeuralNetwork:
   def __init__(self,layerSizes):
      self.weights=[]
      self.layerSizes=layerSizes
      for i in range(1,len(layerSizes)):
         self.weights.append(np.random.randn(layerSizes[i-1],layerSizes[i]))


   def forward_propogation(self,inputData):
            self.activations=[inputData]
            self.zValues=[]
            for i in range(len(self.layerSizes)-1):
               z=np.dot(self.activations[i],self.weights[i])
               self.zValues.append(z)
               activation=sigmoid(z)
               self.activations.append(activation)
            return self.activations[-1]


   def backward_propogation(self,inputData,targetOutput,learningRate):
      output=self.forward_propogation(inputData)
      error=targetOutput-output
      delta=error*sigmoid_derivative(output)

      for i in range(len(layerSizes)-2,-1,-1):
         gradient=np.dot(self.activations[i].T,delta)
         self.weights[i]+=learningRate*gradient
         error=np.dot(delta,self.weights[i].T)
         delta=error*sigmoid_derivative(self.activations[i])


   def train(self,inputData,targetOutput,epochs,learningRate):
      for _ in range(epochs):
         self.backward_propogation(inputData,targetOutput,learningRate)
      return self.forward_propogation(inputData)



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

layerSizes=[2,4,1]

nn=NeuralNetwork(layerSizes)
```

```
output=nn.train(X,y,10000,0.1)
print("Output after training\n")
print(output)
```