import torch  
import torch.nn as nn  
  
X = torch.tensor(([2, 9], [1, 5], [3, 6]), dtype=torch.float) # 3 X 2 tensor  
y = torch.tensor(([92], [100], [89]), dtype=torch.float) # 3 X 1 tensor  
xPredicted = torch.tensor(([4, 8]), dtype=torch.float) # 1 X 2 tensor  
  
print(X.size())  
print(y.size())  
  
# scale units  
X\_max, \_ = torch.max(X, 0)  
xPredicted\_max, \_ = torch.max(xPredicted, 0)  
  
X = torch.div(X, X\_max)  
xPredicted = torch.div(xPredicted, xPredicted\_max)  
y = y / 100 # max test score is 100  
  
  
class Neural\_Network(nn.Module):  
 def \_\_init\_\_(self, ):  
 super(Neural\_Network, self).\_\_init\_\_()  
 # parameters  
 # *TODO: parameters can be parameterized instead of declaring them here* self.inputSize = 2  
 self.outputSize = 1  
 self.hiddenSize = 3  
  
 # weights  
 self.W1 = torch.randn(self.inputSize, self.hiddenSize) # 2 X 3 tensor  
 self.W2 = torch.randn(self.hiddenSize, self.outputSize) # 3 X 1 tensor  
  
 def forward(self, X):  
 self.z = torch.matmul(X, self.W1) # 3 X 3 ".dot" does not broadcast in PyTorch  
 self.z2 = self.sigmoid(self.z) # activation function  
 self.z3 = torch.matmul(self.z2, self.W2)  
 o = self.sigmoid(self.z3) # final activation function  
 return o  
  
 def sigmoid(self, s):  
 return 1 / (1 + torch.exp(-s))  
  
 def sigmoidPrime(self, s):  
 # derivative of sigmoid  
 return s \* (1 - s)  
  
 def backward(self, X, y, o):  
 self.o\_error = y - o # error in output  
 self.o\_delta = self.o\_error \* self.sigmoidPrime(o) # derivative of sig to error  
 self.z2\_error = torch.matmul(self.o\_delta, torch.t(self.W2))  
 self.z2\_delta = self.z2\_error \* self.sigmoidPrime(self.z2)  
 self.W1 += torch.matmul(torch.t(X), self.z2\_delta)  
 self.W2 += torch.matmul(torch.t(self.z2), self.o\_delta)  
  
 def train(self, X, y):  
 # forward + backward pass for training  
 o = self.forward(X)  
 self.backward(X, y, o)  
  
 def saveWeights(self, model):  
 # we will use the PyTorch internal storage functions  
 torch.save(model, "NN")  
 # you can reload model with all the weights and so forth with:  
 # torch.load("NN")  
  
 def predict(self):  
 print("Predicted data based on trained weights: ")  
 print("Input (scaled): \n" + str(xPredicted))  
 print("Output: \n" + str(self.forward(xPredicted)))  
  
  
NN = Neural\_Network()  
for i in range(100): # trains the NN 1,000 times  
 print("#" + str(i) + " Loss: " + str(torch.mean((y - NN(X))\*\*2).detach().item())) # mean sum squared loss  
 NN.train(X, y)  
NN.saveWeights(NN)  
NN.predict()