Assignment 4

# Part 1 – Original Results

## Output

Text

Description automatically generated

## Code

# DL8D.py CS5173/6073 cheng 2023  
# softmax repression on a random sample of MNIST  
# Usage: python DL8D.py  
  
import torch  
import torchvision  
import torch.utils.data as D  
import numpy as np  
import random  
import matplotlib.pyplot as plt  
  
mnist = torchvision.datasets.MNIST('/data/')  
num\_samples = 4096  
trainsample = D.RandomSampler(range(len(mnist)), num\_samples=num\_samples)  
sampleiter = iter(trainsample)  
  
x = []  
targets = []  
for i in range(num\_samples):  
 j = next(sampleiter)  
 x.append(list(mnist[j][0].getdata()))  
 targets.append(mnist[j][1])  
  
X = torch.tensor(x, dtype=torch.float32)  
y = torch.tensor(targets, dtype=torch.long)  
  
d = 28 \* 28  
q = 10  
class LinearRegression(torch.nn.Module):  
 def \_\_init\_\_(self):  
 super(LinearRegression, self).\_\_init\_\_()  
 self.linear = torch.nn.Linear(d, q)  
  
 def forward(self, x):  
 return self.linear(x)  
  
model = LinearRegression()  
loss\_fun = torch.nn.CrossEntropyLoss()  
optimizer = torch.optim.Adam(model.parameters())  
  
batch\_size = 256  
rounds = 1000  
indices = list(range(num\_samples))  
for i in range(rounds):  
 random.shuffle(indices)  
 batch\_indices = torch.tensor(indices[:batch\_size])  
 Xbatch = X[batch\_indices]  
 ybatch = y[batch\_indices]  
 o = model(Xbatch)  
 loss = loss\_fun(o, ybatch)  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
print(loss.item())  
o = model(X)  
ypred = torch.argmax(o, dim=1)  
misclassified = torch.sum((ypred != y))  
print('misclassified =', misclassified.item())

# Part 2 – Results after adding another linear and nonlinear function

## Output

Text

Description automatically generated

## Code

# DL8D.py CS5173/6073 cheng 2023  
# softmax repression on a random sample of MNIST  
# Usage: python DL8D.py  
  
import torch  
import torchvision  
import torch.utils.data as D  
import numpy as np  
import random  
import matplotlib.pyplot as plt  
  
mnist = torchvision.datasets.MNIST('/data/', download=True)  
num\_samples = 4096  
trainsample = D.RandomSampler(range(len(mnist)), num\_samples=num\_samples)  
sampleiter = iter(trainsample)  
  
x = []  
targets = []  
for i in range(num\_samples):  
 j = next(sampleiter)  
 x.append(list(mnist[j][0].getdata()))  
 targets.append(mnist[j][1])  
  
X = torch.tensor(x, dtype=torch.float32)  
y = torch.tensor(targets, dtype=torch.long)  
  
d = 28 \* 28  
q = 10  
  
  
class LinearRegression(torch.nn.Module):  
 def \_\_init\_\_(self):  
 super(LinearRegression, self).\_\_init\_\_()  
 self.linear\_1 = torch.nn.Linear(d, 256)  
 self.linear\_2 = torch.nn.Linear(256, q)  
 self.activation = torch.nn.ReLU()  
  
 def forward(self, x):  
 h = self.activation(self.linear\_1(x))  
 return self.linear\_2(h)  
  
  
model = LinearRegression()  
loss\_fun = torch.nn.CrossEntropyLoss()  
optimizer = torch.optim.Adam(model.parameters())  
  
batch\_size = 256  
rounds = 1000  
indices = list(range(num\_samples))  
for i in range(rounds):  
 random.shuffle(indices)  
 batch\_indices = torch.tensor(indices[:batch\_size])  
 Xbatch = X[batch\_indices]  
 ybatch = y[batch\_indices]  
 o = model(Xbatch)  
 loss = loss\_fun(o, ybatch)  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
print(loss.item())  
o = model(X)  
ypred = torch.argmax(o, dim=1)  
misclassified = torch.sum((ypred != y))  
print('misclassified =', misclassified.item())