```
import torch
import torch.nn as nn
import torchvision.transforms as transforms
import torchvision.datasets as dsets
import random as rand
import math

def act(x):
    return x*x*400

if torch.cuda.is_available():
    avDev = torch.device("cuda")
else:
    avDev = torch.device("cpu")

print(avDev)

train_dataset = dsets.MNIST(root='./data',
                            train=True,
                            transform=transforms.ToTensor(),
                            download=True)

test_dataset = dsets.MNIST(root='./data',
                           train=False,
                           transform=transforms.ToTensor())


batch_size = 100
epochs=5

train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)

test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False)

class LogisticRegressionModel(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(LogisticRegressionModel, self).__init__()
        self.linear1 = nn.Linear(input_dim, 300)
        self.linear2 = nn.Linear(300, output_dim)

    def forward(self, x):
        out = act(self.linear1(x))
        out = self.linear2(out)
        return out

input_dim = 28*28
output_dim = 10

model = LogisticRegressionModel(input_dim, output_dim)
```

cuda

```
model.to(avDev)

criterion = nn.CrossEntropyLoss().to(avDev)



learning_rate = 0.001

optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

iter = 0
for epoch in range(epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.view(-1, 28*28).to(avDev)
        labels = labels.to(avDev)

        optimizer.zero_grad()

        outputs = model(images)

        loss = criterion(outputs, labels)#

        loss.backward()

        optimizer.step()

    if iter % 1 == 0:
        correct = 0
        total = 0
        for images, labels in test_loader:
            images = images.view(-1, 28*28).to(avDev)

            outputs = model(images)

            _, predicted = torch.max(outputs.data, 1)

            total += labels.size(0)
            correct += (predicted.cpu() == labels.cpu()).sum().float()

        accuracy = 100. * correct / total

        # Print Loss
        print('Epochs: {}. Loss: {}. Accuracy: {}'.format(epoch, loss.item(), ac

```

```
Epochs: 0. Loss: 0.46888694167137146. Accuracy: 94.37999725341797
Epochs: 1. Loss: 0.060228001326322556. Accuracy: 95.54000091552734
Epochs: 2. Loss: 0.2160300314426422. Accuracy: 95.80999755859375
Epochs: 3. Loss: 0.056213587522506714. Accuracy: 96.12000274658203
Epochs: 4. Loss: 0.017406903207302094. Accuracy: 96.38999938964844
```

```
1
```