```
import torch
import torch.nn as nn
import torchvision.datasets as dsets
import torchvision.transforms as transforms
from torch.autograd import Variable



# MNIST Dataset (Images and Labels)
train_dataset = dsets.MNIST(root ='./data',
              train = True,
              transform = transforms.ToTensor(),
              download = True)
batch_size = 50
test_dataset = dsets.MNIST(root ='./data',
            train = False,
            transform = transforms.ToTensor())

# Dataset Loader (Input Pipeline)
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)



# Hyper Parameters
input_size = 784
num_classes = 10
num_epochs = 5
batch_size = 100
learning_rate = 0.001



class LogisticRegression(nn.Module):
    def __init__(self, input_size, num_classes):
        super(LogisticRegression, self).__init__()
        self.linear = nn.Linear(input_size, num_classes)

    def forward(self, x):
        out = self.linear(x)
        return out



model = LogisticRegression(input_size, num_classes)



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



# Training the Model
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = Variable(images.view(-1, 28 * 28))
        labels = Variable(labels)

        # Forward + Backward + Optimize
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        if (i + 1) % 100 == 0:
            print('Epoch: [% d/% d], Step: [% d/% d], Loss: %.4f'
                % (epoch + 1, num_epochs, i + 1,
                    len(train_dataset) // batch_size, loss.data))
```

```
Epoch: [ 1/ 5], Step: [ 300/ 600], Loss: 1.9775
Epoch: [ 1/ 5], Step: [ 400/ 600], Loss: 1.8696
Epoch: [ 1/ 5], Step: [ 500/ 600], Loss: 1.7381
Epoch: [ 1/ 5], Step: [ 600/ 600], Loss: 1.7125
Epoch: [ 1/ 5], Step: [ 700/ 600], Loss: 1.6234
Epoch: [ 1/ 5], Step: [ 800/ 600], Loss: 1.6167
Epoch: [ 1/ 5], Step: [ 900/ 600], Loss: 1.4918
Epoch: [ 1/ 5], Step: [ 1000/ 600], Loss: 1.4142
Epoch: [ 1/ 5], Step: [ 1100/ 600], Loss: 1.4807
Epoch: [ 1/ 5], Step: [ 1200/ 600], Loss: 1.4734
Epoch: [ 2/ 5], Step: [ 100/ 600], Loss: 1.4078
Epoch: [ 2/ 5], Step: [ 200/ 600], Loss: 1.3097
Epoch: [ 2/ 5], Step: [ 300/ 600], Loss: 1.4305
Epoch: [ 2/ 5], Step: [ 400/ 600], Loss: 1.1940
Epoch: [ 2/ 5], Step: [ 500/ 600], Loss: 1.1231
Epoch: [ 2/ 5], Step: [ 600/ 600], Loss: 1.1887
Epoch: [ 2/ 5], Step: [ 700/ 600], Loss: 1.2000
Epoch: [ 2/ 5], Step: [ 800/ 600], Loss: 1.2827
Epoch: [ 2/ 5], Step: [ 900/ 600], Loss: 1.0901
Epoch: [ 2/ 5], Step: [ 1000/ 600], Loss: 1.2144
Epoch: [ 2/ 5], Step: [ 1100/ 600], Loss: 1.1319
Epoch: [ 2/ 5], Step: [ 1200/ 600], Loss: 0.9549
Epoch: [ 3/ 5], Step: [ 100/ 600], Loss: 1.0599
Epoch: [ 3/ 5], Step: [ 200/ 600], Loss: 1.0000
Epoch: [ 3/ 5], Step: [ 300/ 600], Loss: 1.1094
Epoch: [ 3/ 5], Step: [ 400/ 600], Loss: 0.9460
Epoch: [ 3/ 5], Step: [ 500/ 600], Loss: 0.9049
Epoch: [ 3/ 5], Step: [ 600/ 600], Loss: 1.0086
Epoch: [ 3/ 5], Step: [ 700/ 600], Loss: 1.0806
Epoch: [ 3/ 5], Step: [ 800/ 600], Loss: 0.9107
Epoch: [ 3/ 5], Step: [ 900/ 600], Loss: 0.8483
Epoch: [ 3/ 5], Step: [ 1000/ 600], Loss: 0.8997
Epoch: [ 3/ 5], Step: [ 1100/ 600], Loss: 1.0020
Epoch: [ 3/ 5], Step: [ 1200/ 600], Loss: 0.9539
Epoch: [ 4/ 5], Step: [ 100/ 600], Loss: 1.0002
Epoch: [ 4/ 5], Step: [ 200/ 600], Loss: 1.0690
Epoch: [ 4/ 5], Step: [ 300/ 600], Loss: 0.7204
Epoch: [ 4/ 5], Step: [ 400/ 600], Loss: 0.8605
Epoch: [ 4/ 5], Step: [ 500/ 600], Loss: 0.9070
Epoch: [ 4/ 5], Step: [ 600/ 600], Loss: 0.9304
Epoch: [ 4/ 5], Step: [ 700/ 600], Loss: 0.8130
Epoch: [ 4/ 5], Step: [ 800/ 600], Loss: 0.7922
Epoch: [ 4/ 5], Step: [ 900/ 600], Loss: 0.8024
Epoch: [ 4/ 5], Step: [ 1000/ 600], Loss: 0.8400
Epoch: [ 4/ 5], Step: [ 1100/ 600], Loss: 0.8124
Epoch: [ 4/ 5], Step: [ 1200/ 600], Loss: 0.8905
Epoch: [ 5/ 5], Step: [ 100/ 600], Loss: 0.7574
Epoch: [ 5/ 5], Step: [ 200/ 600], Loss: 0.8195
Epoch: [ 5/ 5], Step: [ 300/ 600], Loss: 0.7503
Epoch: [ 5/ 5], Step: [ 400/ 600], Loss: 0.8072
Epoch: [ 5/ 5], Step: [ 500/ 600], Loss: 0.6906
Epoch: [ 5/ 5], Step: [ 600/ 600], Loss: 0.6157
Epoch: [ 5/ 5], Step: [ 700/ 600], Loss: 0.6567
Epoch: [ 5/ 5], Step: [ 800/ 600], Loss: 0.7662
Epoch: [ 5/ 5], Step: [ 900/ 600], Loss: 0.7824
Epoch: [ 5/ 5], Step: [ 1000/ 600], Loss: 0.8043
Epoch: [ 5/ 5], Step: [ 1100/ 600], Loss: 0.6856
Epoch: [ 5/ 5], Step: [ 1200/ 600], Loss: 0.6119
```

```
# Test the Model
correct = 0
total = 0
for images, labels in test_loader:
    images = Variable(images.view(-1, 28 * 28))
    outputs = model(images)
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum()

print('Accuracy of the model on the 10000 test images: % d %%' % (
            100 * correct / total))
```

```
Accuracy of the model on the 10000 test images:  85 %
```

```
import torch
import torch.nn as nn
import torchvision.datasets as dsets
import torchvision.transforms as transforms
from torch.autograd import Variable
```

```
# MNIST Dataset (Images and Labels)
train_dataset = dsets.MNIST(root ='./data',
              train = True,
              transform = transforms.ToTensor(),
              download = True)

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

# Dataset Loader (Input Pipeline)
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)

# Hyper Parameters
input_size = 784
num_classes = 10
num_epochs = 5
batch_size = 100
learning_rate = 0.001

# Model
class LogisticRegression(nn.Module):
  def __init__(self, input_size, num_classes):
    super(LogisticRegression, self).__init__()
    self.linear = nn.Linear(input_size, num_classes)

  def forward(self, x):
    out = self.linear(x)
    return out


model = LogisticRegression(input_size, num_classes)

# Loss and Optimizer
# Softmax is internally computed.
# Set parameters to be updated.
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr = learning_rate)

# Training the Model
for epoch in range(num_epochs):
  for i, (images, labels) in enumerate(train_loader):
    images = Variable(images.view(-1, 28 * 28))
    labels = Variable(labels)

    # Forward + Backward + Optimize
    optimizer.zero_grad()
    outputs = model(images)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()

    if (i + 1) % 100 == 0:
      print('Epoch: [% d/% d], Step: [% d/% d], Loss: %.4f'
        % (epoch + 1, num_epochs, i + 1,
          len(train_dataset) // batch_size, loss.data))

# Test the Model
correct = 0
total = 0
for images, labels in test_loader:
  images = Variable(images.view(-1, 28 * 28))
  outputs = model(images)
  _, predicted = torch.max(outputs.data, 1)
  total += labels.size(0)
  correct += (predicted == labels).sum()

print('Accuracy of the model on the 10000 test images: % d %%' % (
      100 * correct / total))
```

```
Epoch: [ 1/ 5], Step: [ 100/ 600], Loss: 2.2391
Epoch: [ 1/ 5], Step: [ 200/ 600], Loss: 2.1111
Epoch: [ 1/ 5], Step: [ 300/ 600], Loss: 2.0845
Epoch: [ 1/ 5], Step: [ 400/ 600], Loss: 1.9413
Epoch: [ 1/ 5], Step: [ 500/ 600], Loss: 1.8801
Epoch: [ 1/ 5], Step: [ 600/ 600], Loss: 1.8433
Epoch: [ 2/ 5], Step: [ 100/ 600], Loss: 1.7449
Epoch: [ 2/ 5], Step: [ 200/ 600], Loss: 1.6715
Epoch: [ 2/ 5], Step: [ 300/ 600], Loss: 1.6423
Epoch: [ 2/ 5], Step: [ 400/ 600], Loss: 1.5381
Epoch: [ 2/ 5], Step: [ 500/ 600], Loss: 1.5653
Epoch: [ 2/ 5], Step: [ 600/ 600], Loss: 1.4361
Epoch: [ 3/ 5], Step: [ 100/ 600], Loss: 1.5050
Epoch: [ 3/ 5], Step: [ 200/ 600], Loss: 1.3620
Epoch: [ 3/ 5], Step: [ 300/ 600], Loss: 1.3850
Epoch: [ 3/ 5], Step: [ 400/ 600], Loss: 1.3414
Epoch: [ 3/ 5], Step: [ 500/ 600], Loss: 1.3932
Epoch: [ 3/ 5], Step: [ 600/ 600], Loss: 1.2719
Epoch: [ 4/ 5], Step: [ 100/ 600], Loss: 1.2336
Epoch: [ 4/ 5], Step: [ 200/ 600], Loss: 1.2778
Epoch: [ 4/ 5], Step: [ 300/ 600], Loss: 1.1559
Epoch: [ 4/ 5], Step: [ 400/ 600], Loss: 1.1081
Epoch: [ 4/ 5], Step: [ 500/ 600], Loss: 1.1411
Epoch: [ 4/ 5], Step: [ 600/ 600], Loss: 1.0666
Epoch: [ 5/ 5], Step: [ 100/ 600], Loss: 1.1531
Epoch: [ 5/ 5], Step: [ 200/ 600], Loss: 1.0794
Epoch: [ 5/ 5], Step: [ 300/ 600], Loss: 0.9705
Epoch: [ 5/ 5], Step: [ 400/ 600], Loss: 1.0563
Epoch: [ 5/ 5], Step: [ 500/ 600], Loss: 1.0063
Epoch: [ 5/ 5], Step: [ 600/ 600], Loss: 0.9746
Accuracy of the model on the 10000 test images:  82 %
```