```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms


batch_size = 64
learning_rate = 0.001
num_epochs = 10

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

train_dataset = torchvision.datasets.MNIST(root='./data', train=True,
transform=transform, download=True)
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
batch_size=batch_size, shuffle=True)


Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-
ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-
ubyte.gz to ./data/MNIST/raw/train-images-idx3-ubyte.gz

100%|██████████| 9912422/9912422 [00:00<00:00, 93293132.32it/s]

Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to
./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-
ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-
ubyte.gz to ./data/MNIST/raw/train-labels-idx1-ubyte.gz

100%|██████████| 28881/28881 [00:00<00:00, 18567702.92it/s]

Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to
./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
to ./data/MNIST/raw/t10k-images-idx3-ubyte.gz

100%|██████████| 1648877/1648877 [00:00<00:00, 27080999.13it/s]

Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to
./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
```

```
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
to ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz

100%|██████████| 4542/4542 [00:00<00:00, 11239250.01it/s]

Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to
./data/MNIST/raw
```

```python
class ANN(nn.Module):
    def __init__(self):
        super(ANN, self).__init__()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(28 * 28, 128)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x

model = ANN()

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

for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backpropagation and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if (i+1) % 100 == 0:
            print(f'Epoch [{epoch+1}/{num_epochs}], Step
[{i+1}/{len(train_loader)}], Loss: {loss.item():.4f}')

print('Training Finished')
```

```
Epoch [1/10], Step [100/938], Loss: 0.6009
Epoch [1/10], Step [200/938], Loss: 0.3787
Epoch [1/10], Step [300/938], Loss: 0.4398
Epoch [1/10], Step [400/938], Loss: 0.4403
Epoch [1/10], Step [500/938], Loss: 0.2239
```

```
Epoch [1/10], Step [600/938], Loss: 0.3549
Epoch [1/10], Step [700/938], Loss: 0.1925
Epoch [1/10], Step [800/938], Loss: 0.1621
Epoch [1/10], Step [900/938], Loss: 0.3188
Epoch [2/10], Step [100/938], Loss: 0.0912
Epoch [2/10], Step [200/938], Loss: 0.1825
Epoch [2/10], Step [300/938], Loss: 0.2192
Epoch [2/10], Step [400/938], Loss: 0.2396
Epoch [2/10], Step [500/938], Loss: 0.0756
Epoch [2/10], Step [600/938], Loss: 0.1828
Epoch [2/10], Step [700/938], Loss: 0.2874
Epoch [2/10], Step [800/938], Loss: 0.1717
Epoch [2/10], Step [900/938], Loss: 0.2425
Epoch [3/10], Step [100/938], Loss: 0.2002
Epoch [3/10], Step [200/938], Loss: 0.2204
Epoch [3/10], Step [300/938], Loss: 0.1484
Epoch [3/10], Step [400/938], Loss: 0.0933
Epoch [3/10], Step [500/938], Loss: 0.0628
Epoch [3/10], Step [600/938], Loss: 0.0749
Epoch [3/10], Step [700/938], Loss: 0.2822
Epoch [3/10], Step [800/938], Loss: 0.0531
Epoch [3/10], Step [900/938], Loss: 0.0485
Epoch [4/10], Step [100/938], Loss: 0.1120
Epoch [4/10], Step [200/938], Loss: 0.1745
Epoch [4/10], Step [300/938], Loss: 0.0613
Epoch [4/10], Step [400/938], Loss: 0.0788
Epoch [4/10], Step [500/938], Loss: 0.1219
Epoch [4/10], Step [600/938], Loss: 0.0515
Epoch [4/10], Step [700/938], Loss: 0.0524
Epoch [4/10], Step [800/938], Loss: 0.1940
Epoch [4/10], Step [900/938], Loss: 0.0826
Epoch [5/10], Step [100/938], Loss: 0.1465
Epoch [5/10], Step [200/938], Loss: 0.0870
Epoch [5/10], Step [300/938], Loss: 0.0440
Epoch [5/10], Step [400/938], Loss: 0.0211
Epoch [5/10], Step [500/938], Loss: 0.1637
Epoch [5/10], Step [600/938], Loss: 0.0640
Epoch [5/10], Step [700/938], Loss: 0.0734
Epoch [5/10], Step [800/938], Loss: 0.0264
Epoch [5/10], Step [900/938], Loss: 0.0720
Epoch [6/10], Step [100/938], Loss: 0.0087
Epoch [6/10], Step [200/938], Loss: 0.0405
Epoch [6/10], Step [300/938], Loss: 0.0502
Epoch [6/10], Step [400/938], Loss: 0.1903
Epoch [6/10], Step [500/938], Loss: 0.1153
Epoch [6/10], Step [600/938], Loss: 0.1051
Epoch [6/10], Step [700/938], Loss: 0.0588
Epoch [6/10], Step [800/938], Loss: 0.0961
Epoch [6/10], Step [900/938], Loss: 0.0412
```

```
Epoch [7/10], Step [100/938], Loss: 0.1116
Epoch [7/10], Step [200/938], Loss: 0.0742
Epoch [7/10], Step [300/938], Loss: 0.0561
Epoch [7/10], Step [400/938], Loss: 0.0674
Epoch [7/10], Step [500/938], Loss: 0.1323
Epoch [7/10], Step [600/938], Loss: 0.0499
Epoch [7/10], Step [700/938], Loss: 0.0924
Epoch [7/10], Step [800/938], Loss: 0.0420
Epoch [7/10], Step [900/938], Loss: 0.0096
Epoch [8/10], Step [100/938], Loss: 0.0180
Epoch [8/10], Step [200/938], Loss: 0.0766
Epoch [8/10], Step [300/938], Loss: 0.0297
Epoch [8/10], Step [400/938], Loss: 0.0245
Epoch [8/10], Step [500/938], Loss: 0.0553
Epoch [8/10], Step [600/938], Loss: 0.0698
Epoch [8/10], Step [700/938], Loss: 0.0541
Epoch [8/10], Step [800/938], Loss: 0.0245
Epoch [8/10], Step [900/938], Loss: 0.0263
Epoch [9/10], Step [100/938], Loss: 0.0848
Epoch [9/10], Step [200/938], Loss: 0.0063
Epoch [9/10], Step [300/938], Loss: 0.0820
Epoch [9/10], Step [400/938], Loss: 0.0400
Epoch [9/10], Step [500/938], Loss: 0.0579
Epoch [9/10], Step [600/938], Loss: 0.0408
Epoch [9/10], Step [700/938], Loss: 0.0723
Epoch [9/10], Step [800/938], Loss: 0.0036
Epoch [9/10], Step [900/938], Loss: 0.0888
Epoch [10/10], Step [100/938], Loss: 0.0285
Epoch [10/10], Step [200/938], Loss: 0.0456
Epoch [10/10], Step [300/938], Loss: 0.1433
Epoch [10/10], Step [400/938], Loss: 0.0572
Epoch [10/10], Step [500/938], Loss: 0.0726
Epoch [10/10], Step [600/938], Loss: 0.0113
Epoch [10/10], Step [700/938], Loss: 0.0486
Epoch [10/10], Step [800/938], Loss: 0.0122
Epoch [10/10], Step [900/938], Loss: 0.0884
Training Finished
```

```
test_dataset = torchvision.datasets.MNIST(root='./data', train=False,
transform=transform, download=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
batch_size=batch_size, shuffle=False)

# Switch model to evaluation mode
model.eval()

correct = 0
total = 0

# Disable gradient computation during evaluation
```

```python
with torch.no_grad():
    for images, labels in test_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)  # Get the predicted
class for each image
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: {:.2f}
%'.format(100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 97.28%
```