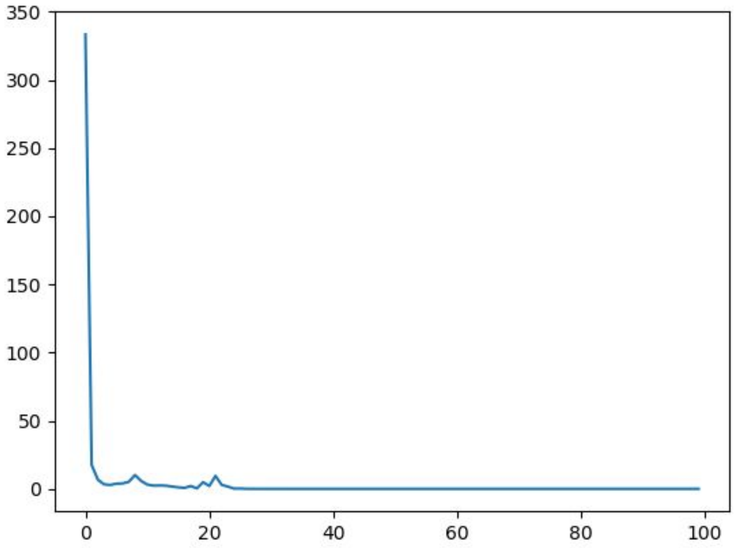
```
(COMP9517) C:\Repositories\03-University\COMP9517\labs\lab05>python
lab5_pytorch_template.py

Epoch:  10 Accuracy:  96.0

Epoch:  20 Accuracy:  97.0


Epoch:  30 Accuracy:  96.8

Epoch:  40 Accuracy:  96.6

Epoch:  50 Accuracy:  96.8

Epoch:  60 Accuracy:  97.0

Epoch:  70 Accuracy:  97.0

Epoch:  80 Accuracy:  97.0

Epoch:  90 Accuracy:  97.0

Epoch:  100 Accuracy:  97.0
```



NB Learning occurs almost fully after one epoch

```
import torch
import pandas as pd
import numpy as np
import torch.nn.functional as F
import torch.utils.data as utils

# Load data(do not change)
data = pd.read_csv("src/mnist_train.csv")
train_data = data[:2000]
test_data = data[2000:2500]


# ----- Prepare Data ----- #
# step one: preparing your data including data normalization
def create_tensor(row):
    label = row["label"]
    image = row.drop("label")
    image = np.array(image, dtype=np.uint8)
    image = np.reshape(image, (28,28))
    image = np.array([image])
    return label, image


test_data = test_data.apply(create_tensor, axis=1)
test_labels, test_data = zip(*test_data.to_list())
test_data = list(test_data)

train_data = train_data.apply(create_tensor, axis=1)
train_labels, train_data = zip(*train_data.to_list())
train_data = list(train_data)

# step two: transform np array to pytorch tensor
train_data = torch.stack([torch.Tensor(i) for i in train_data])
train_labels = list(train_labels)
train_labels = np.eye(10)[train_labels]
train_labels = torch.stack([torch.Tensor(i) for i in train_labels])

test_data = torch.stack([torch.Tensor(i) for i in test_data])
test_labels = list(test_labels)
test_labels = np.eye(10)[test_labels]
test_labels = torch.stack([torch.Tensor(i) for i in test_labels])


train_dataset = utils.TensorDataset(train_data, train_labels)
train_dataloader = utils.DataLoader(train_dataset, batch_size=20)

test_dataset = utils.TensorDataset(test_data, test_labels)
test_dataloader = utils.DataLoader(test_dataset, batch_size=20)
```

```
# ----- Build CNN Network ----- #
# Define your model here
class mymodel(torch.nn.Module):
    def __init__(self):
        super(mymodel, self).__init__()
        self.conv1 = torch.nn.Conv2d(1, 10, kernel_size=3, stride=1)
        self.conv2 = torch.nn.Conv2d(10, 50, kernel_size=3, stride=1)

        self.pool1 = torch.nn.MaxPool2d(kernel_size=2)
        self.pool2 = torch.nn.MaxPool2d(kernel_size=2)

        self.fc1 = torch.nn.Linear(1250, 256)
        self.fc2 = torch.nn.Linear(256, 10)

    def forward(self, x):
        x = self.pool1(F.relu(self.conv1(x)))
        x = self.pool2(F.relu(self.conv2(x)))
        x = x.view(x.shape[0], -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# Define our model
model = mymodel()
# Define your learning rate
learning_rate = 0.002
# Define your optimizer
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
# Define your loss function
criterion = F.cross_entropy

# ----- Complete PlotLearningCurve function ----- #
def PlotLearningCurve(epoch, trainingloss, testingloss):
    import matplotlib.pyplot as plt
    plt.plot(trainingloss)
    plt.show()
    pass

# ----- Main Function ----- #
trainingloss = []
testingloss = []
# Define number of iterations
epochs = 100
for epoch in range(1, epochs + 1):
    print("Epoch:", epoch)
    epoch_loss = 0
    # step one : fit your model by using training data and get predict label
    model.train()
    i = 0
    for images, labels in train_dataloader:
        output = model(images)
        labels = torch.Tensor(list(map(lambda x: np.argmax(x), labels)))
        loss = criterion(output, labels.long())

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()
        i+=1

    trainingloss.append(epoch_loss)

    model.eval()
    epoch_correct = 0
    epoch_total = 0
    with torch.no_grad():
        for images, labels in test_dataloader:
            ps = model(images)
            labels = list(map(lambda x: np.argmax(x).item(), labels))
            predicted = list(map(lambda x: np.argmax(x).item(), ps))
            for x in range(len(labels)):
                if labels[x] == predicted[x]:
                    epoch_correct += 1

            epoch_total += len(labels)

    accuracy = epoch_correct/epoch_total * 100
    if epoch % 10 == 0:
        print("Epoch: ", epoch, "Accuracy: ", accuracy)

PlotLearningCurve(None, trainingloss, None)
```