22AIE304 Deep Learning Labsheet 5

Name: Aniketh Vijesh

Roll No: AM.EN.U4AIE22009

image.png

```
import torch
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

import torchvision.datasets as datasets
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

```
# Setting up the datasets and Dataloaders

train_set = datasets.MNIST(
    root="./data", train=True, download=True, transform=transforms.ToTensor()
)
test_set = datasets.MNIST(
    root="./data", train=False, download=True, transform=transforms.ToTensor()
)


train_loader = DataLoader(dataset=train_set, batch_size=64, shuffle=True, num_workers=2)
test_loader = DataLoader(dataset=test_set, batch_size=64, shuffle=False, num_workers=2)
```

```
class FeedforwardNeuralNet(nn.Module):
    def __init__(self, input_size=784, hidden_size=128, output_size=10, num_hidden_layers=2, activation=nn.ReLU()):
        super(FeedforwardNeuralNet, self).__init__()

        self.fc1 = nn.Linear(input_size, hidden_size)

        self.hidden_layers = nn.ModuleList([nn.Linear(hidden_size, hidden_size) for _ in range(num_hidden_layers - 1)])

        self.output = nn.Linear(hidden_size, output_size)


        self.activation = activation

    def forward(self, x):
        x = x.view(x.size(0), -1)

        x = self.activation(self.fc1(x))

        for layer in self.hidden_layers:
            x = self.activation(layer(x))

        x = self.output(x)
        return x
```

```
# Training function
def train_model(model, train_loader, optimizer, criterion, num_epochs):
    model.train()
    train_losses = []
    train_accuracies = []
    for epoch in range(num_epochs):
        correct = 0
        total = 0
        epoch_loss = 0
        for images, labels in train_loader:
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            epoch_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        epoch_accuracy = correct / total
        train_losses.append(epoch_loss / len(train_loader))
        train_accuracies.append(epoch_accuracy)
        print(f"Epoch {epoch+1}/{num_epochs}, Loss: {train_losses[-1]:.4f}, Accuracy: {epoch_accuracy:.4f}")

    return train_losses, train_accuracies
```

image.png

image-2.png

```
# Plot training loss and accuracy
def plot_training_curves(train_losses, train_accuracies):
    epochs = range(1, len(train_losses) + 1)
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.plot(epochs, train_losses, label="Training Loss")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.title("Training Loss Curve")
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs, train_accuracies, label="Training Accuracy", color='orange')
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.title("Training Accuracy Curve")
    plt.legend()

    plt.show()
```

```
# Evaluate the model on the test set
def evaluate_model(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    all_labels = []
    all_predictions = []

    with torch.no_grad():
        for images, labels in test_loader:
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            all_labels.extend(labels.cpu().numpy())
            all_predictions.extend(predicted.cpu().numpy())

    accuracy = correct / total
    print(f"Test Accuracy: {accuracy:.4f}")

    # Confusion matrix
    cm = confusion_matrix(all_labels, all_predictions)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[i for i in range(10)])
    disp.plot(cmap=plt.cm.Blues)
    plt.title("Confusion Matrix")
    plt.show()

    return accuracy
```

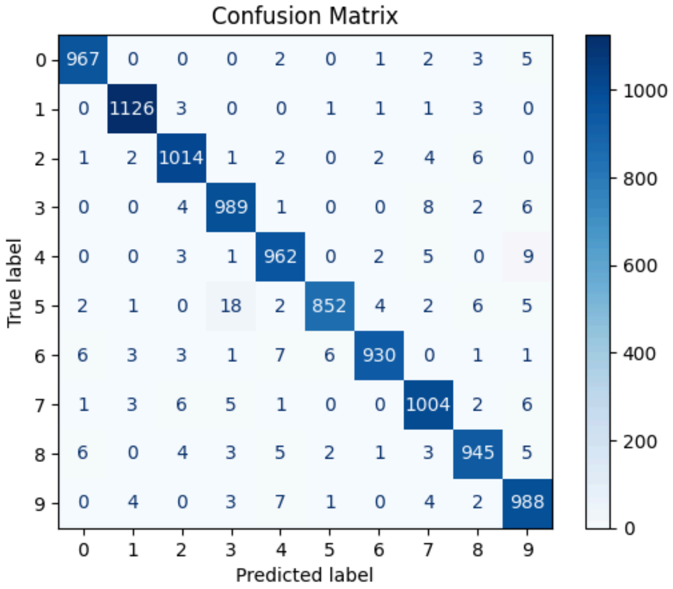
image.png

```
activation_functions = {"ReLU": nn.ReLU(), "Sigmoid": nn.Sigmoid(), "Tanh": nn.Tanh(), "LeakyReLU": nn.LeakyReLU()}
results = {}

for name, activation_function in activation_functions.items():
    print(f"Training with {name} activation function...")
    model = FeedforwardNeuralNet(activation=activation_function)
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    criterion = nn.CrossEntropyLoss()
    train_losses, train_accuracies = train_model(model, train_loader, optimizer, criterion, num_epochs=10)
    test_accuracy = evaluate_model(model, test_loader)
    results[name] = {"train_losses": train_losses, "train_accuracies": train_accuracies, "test_accuracy": test_accuracy}
```

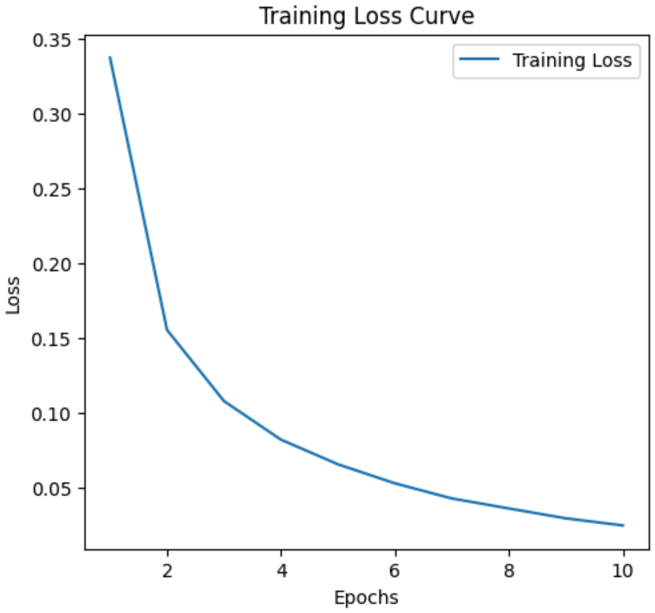
```
plot_training_curves(train_losses, train_accuracies)
```

```
Training with ReLU activation function...
Epoch 1/10, Loss: 0.3371, Accuracy: 0.9079
Epoch 2/10, Loss: 0.1553, Accuracy: 0.9548
Epoch 3/10, Loss: 0.1077, Accuracy: 0.9685
Epoch 4/10, Loss: 0.0820, Accuracy: 0.9760
Epoch 5/10, Loss: 0.0655, Accuracy: 0.9802
Epoch 6/10, Loss: 0.0528, Accuracy: 0.9842
Epoch 7/10, Loss: 0.0427, Accuracy: 0.9872
Epoch 8/10, Loss: 0.0360, Accuracy: 0.9893
Epoch 9/10, Loss: 0.0294, Accuracy: 0.9916
Epoch 10/10, Loss: 0.0247, Accuracy: 0.9932
Test Accuracy: 0.9777
```

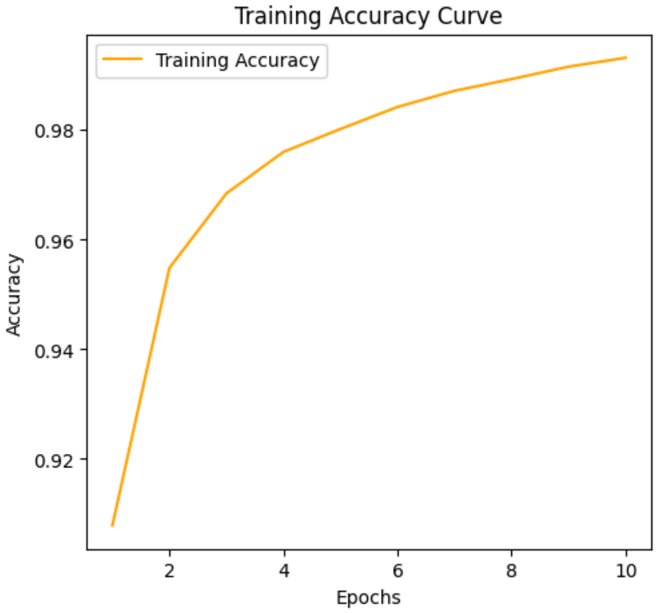


Confusion Matrix

	0	1	2	3	4	5	6	7	8	9
0	967	0	0	0	2	0	1	2	3	5
1	0	1126	3	0	0	1	1	1	3	0
2	1	2	1014	1	2	0	2	4	6	0
3	0	0	4	989	1	0	0	8	2	6
4	0	0	3	1	962	0	2	5	0	9
5	2	1	0	18	2	852	4	2	6	5
6	6	3	3	1	7	6	930	0	1	1
7	1	3	6	5	1	0	0	1004	2	6
8	6	0	4	3	5	2	1	3	945	5
9	0	4	0	3	7	1	0	4	2	988

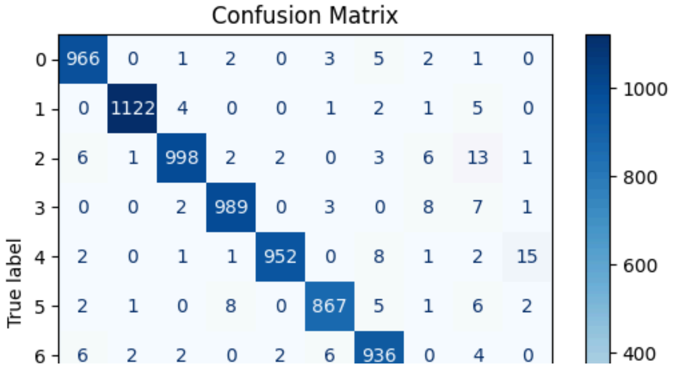


Training Loss Curve



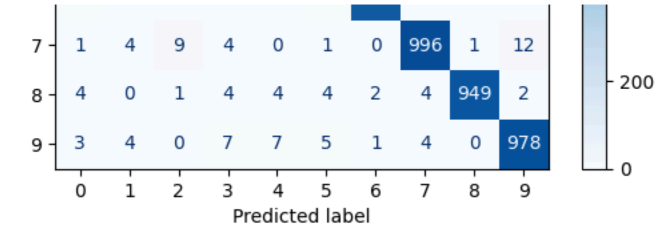
Training Accuracy Curve

```
Training with Sigmoid activation function...
Epoch 1/10, Loss: 0.5352, Accuracy: 0.8706
Epoch 2/10, Loss: 0.2272, Accuracy: 0.9351
Epoch 3/10, Loss: 0.1745, Accuracy: 0.9495
Epoch 4/10, Loss: 0.1419, Accuracy: 0.9585
Epoch 5/10, Loss: 0.1179, Accuracy: 0.9658
Epoch 6/10, Loss: 0.1001, Accuracy: 0.9710
Epoch 7/10, Loss: 0.0855, Accuracy: 0.9761
Epoch 8/10, Loss: 0.0735, Accuracy: 0.9794
Epoch 9/10, Loss: 0.0642, Accuracy: 0.9825
Epoch 10/10, Loss: 0.0563, Accuracy: 0.9844
Test Accuracy: 0.9753
```

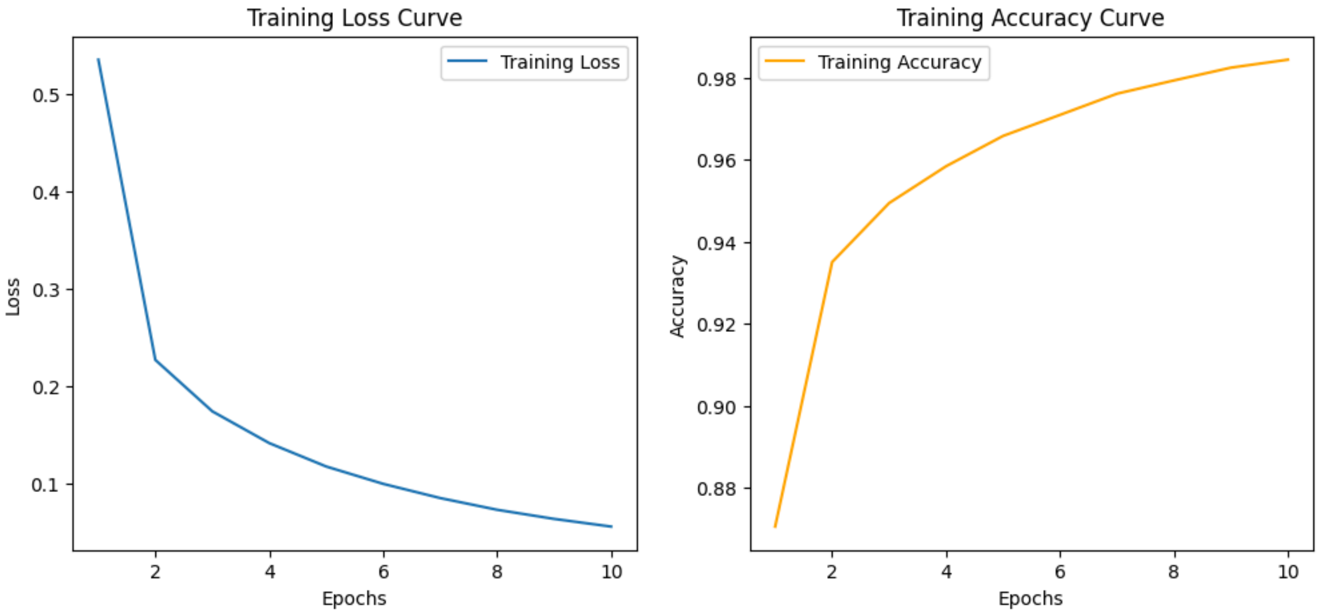


Confusion Matrix

0	966	0	1	2	0	3	5	2	1	0
1	0	1122	4	0	0	1	2	1	5	0
2	6	1	998	2	2	0	3	6	13	1
3	0	0	2	989	0	3	0	8	7	1
4	2	0	1	1	952	0	8	1	2	15
5	2	1	0	8	0	867	5	1	6	2
6	6	2	2	0	2	6	936	0	4	0

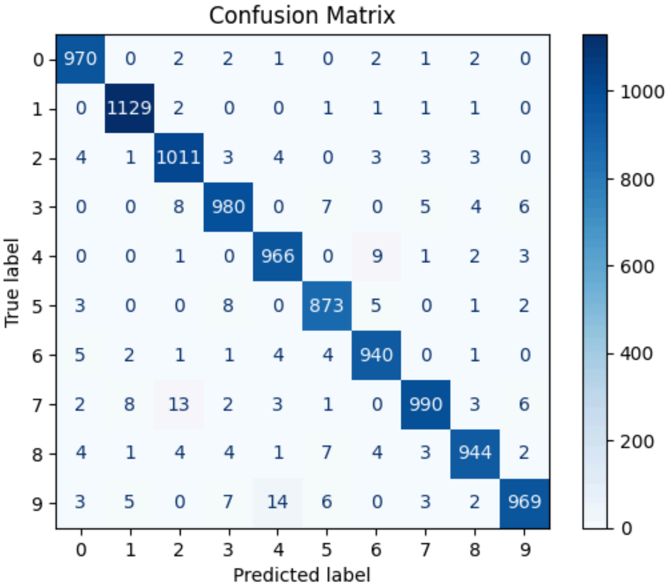


	0	1	2	3	4	5	6	7	8	9
7	1	4	9	4	0	1	0	996	1	12
8	4	0	1	4	4	4	2	4	949	2
9	3	4	0	7	7	5	1	4	0	978

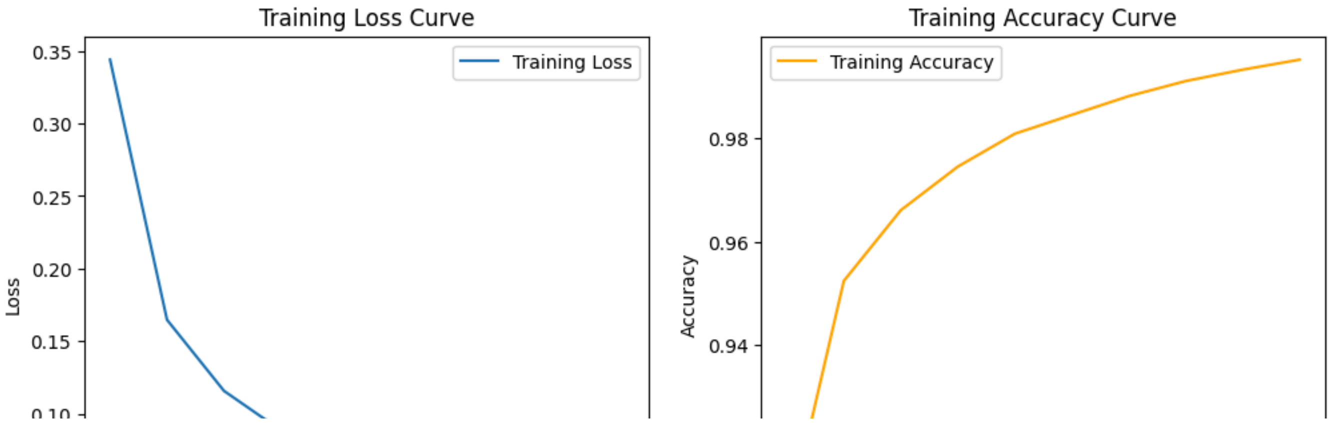

Training Loss Curve

Training Accuracy Curve

```
Training with Tanh activation function...
Epoch 1/10, Loss: 0.3439, Accuracy: 0.9050
Epoch 2/10, Loss: 0.1648, Accuracy: 0.9525
Epoch 3/10, Loss: 0.1158, Accuracy: 0.9661
Epoch 4/10, Loss: 0.0886, Accuracy: 0.9745
Epoch 5/10, Loss: 0.0685, Accuracy: 0.9809
Epoch 6/10, Loss: 0.0545, Accuracy: 0.9845
Epoch 7/10, Loss: 0.0437, Accuracy: 0.9881
Epoch 8/10, Loss: 0.0351, Accuracy: 0.9910
Epoch 9/10, Loss: 0.0278, Accuracy: 0.9932
Epoch 10/10, Loss: 0.0224, Accuracy: 0.9952
Test Accuracy: 0.9772
```


Confusion Matrix

	0	1	2	3	4	5	6	7	8	9
0	970	0	2	2	1	0	2	1	2	0
1	0	1129	2	0	0	1	1	1	1	0
2	4	1	1011	3	4	0	3	3	3	0
3	0	0	8	980	0	7	0	5	4	6
4	0	0	1	0	966	0	9	1	2	3
5	3	0	0	8	0	873	5	0	1	2
6	5	2	1	1	4	4	940	0	1	0
7	2	8	13	2	3	1	0	990	3	6
8	4	1	4	4	1	7	4	3	944	2
9	3	5	0	7	14	6	0	3	2	969


Training Loss Curve

Training Accuracy Curve