DAT565/DIT407 Assignment 6

Ola Bratt
ola.bratt@gmail.com

Patrick Attimont
patrickattimont@gmail.com

2024-02-xx

This paper is addressing the assignment 6 study queries within the *Introduction to Data Science & AI* course, DIT407 at the University of Gothenburg and DAT565 at Chalmers. The main source of information for this project is derived from the lectures and Skiena [1]. Assignment 6 is about neural networks.

Problem 1: The dataset

The dataset consists of 60,000 training images and 10,000 test images. Each image is a 28x28 pixel grayscale image. The images are labeled with the corresponding digit. A random set of the images are shown in figure 1.

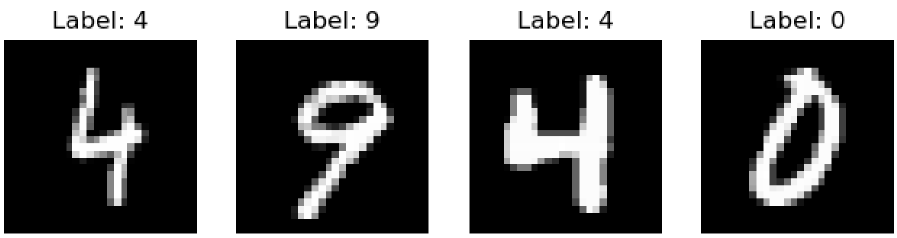

Figure 1: MNIST images

Problem 2: Single hidden layer

The neural network with a single hidden layer has 784 (28*28) input nodes, 300 hidden nodes, and 10 output nodes. The activation function is ReLU for the hidden layer with a batch normalization and logarithmic softmax for the output layer.

```
Epoch [1/10], Training Loss: 0.2200, Test Loss: 0.1065, Test Accuracy: 0.9695
Epoch [2/10], Training Loss: 0.1057, Test Loss: 0.0921, Test Accuracy: 0.9725
Epoch [3/10], Training Loss: 0.0768, Test Loss: 0.0728, Test Accuracy: 0.9765
Epoch [4/10], Training Loss: 0.0613, Test Loss: 0.0666, Test Accuracy: 0.9793
Epoch [5/10], Training Loss: 0.0497, Test Loss: 0.0643, Test Accuracy: 0.9791
Epoch [6/10], Training Loss: 0.0421, Test Loss: 0.0639, Test Accuracy: 0.9798
```

```
Epoch [7/10], Training Loss: 0.0355, Test Loss: 0.0628, Test Accuracy: 0.9811
Epoch [8/10], Training Loss: 0.0309, Test Loss: 0.0569, Test Accuracy: 0.9823
Epoch [9/10], Training Loss: 0.0279, Test Loss: 0.0616, Test Accuracy: 0.9796
Epoch [10/10], Training Loss: 0.0225, Test Loss: 0.0590, Test Accuracy: 0.9816
```

Accuracy for single hidden layer: 0.9816

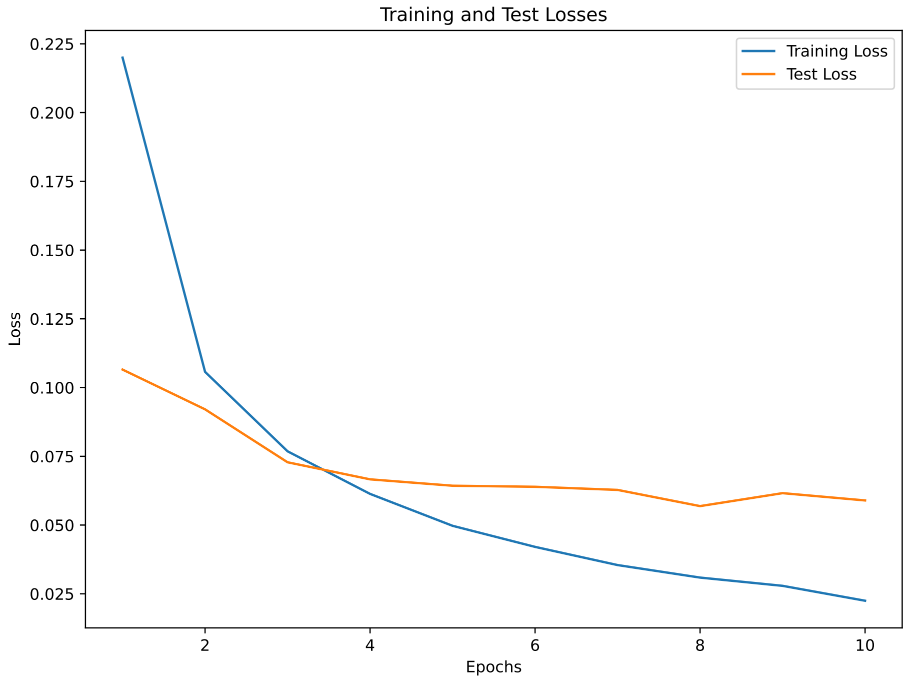

Figure 2: Single hidden layer

Problem 3: Two hidden layers

Accuracy for two hidden layers: 0.9855

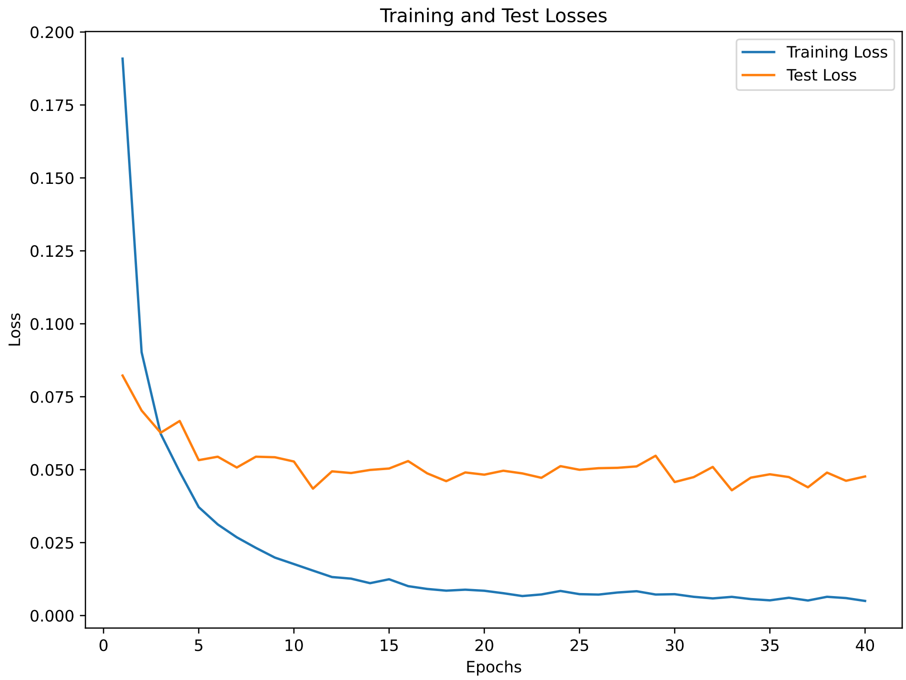

Figure 3: Two hidden layers

Problem 4: Convolutional neural network

Accuracy for convolutional neural network: 0.9929

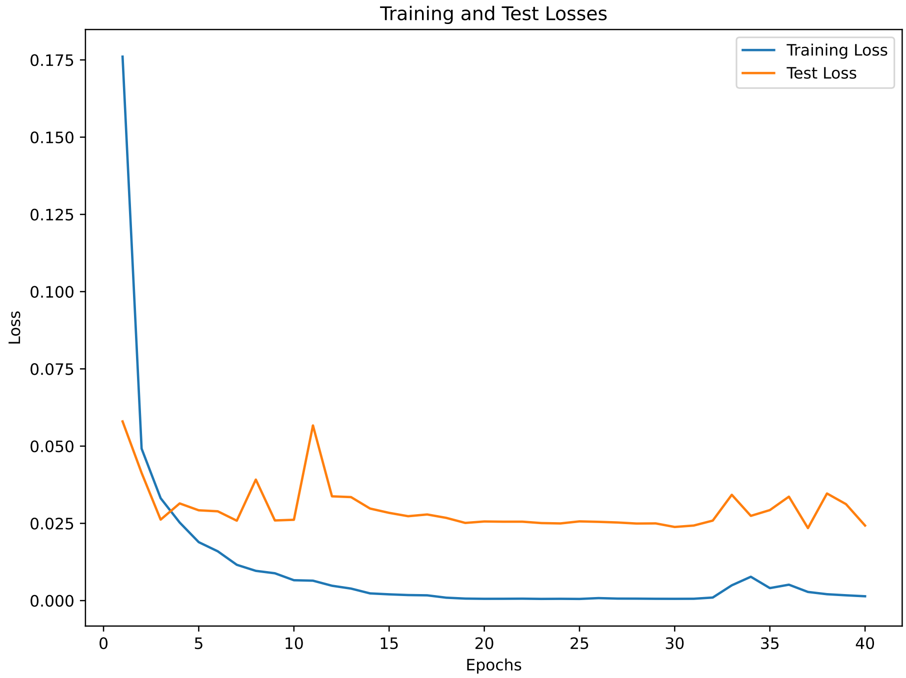

Figure 4: Convolutional neural network

References

[1] Steven S Skiena. *The Data Science Design Manual*. Retrieved 2024-01-20. 2024. URL: https://ebookcentral.proquest.com/lib/gu/detail.action?docID=6312797.

Appendix: Source Code

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from torchvision import datasets
from torch.utils.data import DataLoader


class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_sizes, output_size):
        super(NeuralNet, self).__init__()
        layer_sizes = [input_size] + hidden_sizes + [output_size]

        layers = []
        for i in range(len(layer_sizes) - 1):
            layers.append(nn.Linear(layer_sizes[i], layer_sizes[i+1]))
            if i < len(layer_sizes) - 2:  # Add ReLU and batch normalization except for the last layer
                layers.append(nn.BatchNorm1d(layer_sizes[i+1]))
                layers.append(nn.ReLU())
            else:
                layers.append(nn.LogSoftmax(dim=1)) #layers.append(nn.Softmax(dim=1))


        self.model = nn.Sequential(*layers)

    def forward(self, x):
        x = x.view(-1, 28 * 28)
        x = self.model(x)
        return x

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=1)
        self.fc1 = nn.Linear(32 * 7 * 7, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, kernel_size=2, stride=2)
```

```python
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, kernel_size=2, stride=2)
        x = x.view(-1, 32 * 7 * 7)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        x = F.log_softmax(x, dim=1)
        return x

def plot_images(dataloader, classes):

    for images, labels in train_loader:
        print("Image shape:", images.size())
        print("Label shape:", labels.size())

        fig = plt.figure(figsize=(10, 10))
        for i in range(4):
            plt.subplot(5, 5, i + 1)
            plt.imshow(images[i].squeeze(), cmap='gray')
            plt.title(f'Label: {labels[i]}')
            plt.axis('off')
        plt.show()
        fig.savefig('mnist_images.png', bbox_inches='tight')
        break

def calculate_accuracy(model, dataloader):
    correct = 0
    total = 0
    model.eval()

    with torch.no_grad():
        for inputs, labels in dataloader:
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum().item()
            total += labels.size(0)

    accuracy = correct / total

    return accuracy


def train(model, criterion, optimizer, train_loader, test_loader, num_epochs, name):
    train_losses = []
    test_losses = []

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0

        for images, labels in train_loader:
            outputs = model(images)
            loss = criterion(outputs, labels)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            running_loss += loss.item()


        epoch_loss = running_loss / len(train_loader)
```

```
        train_losses.append(epoch_loss)

        model.eval()
        correct = 0
        total = 0
        test_loss = 0.0
        with torch.no_grad():
            for images, labels in test_loader:
                outputs = model(images)
                _, predicted = torch.max(outputs, 1)
                correct += (predicted == labels).sum().item()
                total += labels.size(0)
                loss = criterion(outputs, labels)
                test_loss += loss.item()
        accuracy = correct / total
        test_loss /= len(test_loader)
        test_losses.append(test_loss)

        print(f"Epoch [{epoch+1}/{num_epochs}], Training Loss: {epoch_loss:.4f}, Test Loss: {test_loss:.4f}, Test Accuracy: {accuracy:.4f}")


    fig, ax = plt.subplots(figsize=(8, 6), layout='constrained')
    ax.plot(range(1, num_epochs + 1), train_losses, label='Training Loss')
    ax.plot(range(1, num_epochs + 1), test_losses, label='Test Loss ')
    ax.set_xlabel('Epochs')
    ax.set_ylabel('Loss')
    ax.set_title('Training and Test Losses')
    ax.legend()
    plt.show()
    fig.savefig(name + ".pdf", bbox_inches='tight')




# Importing the dataset
batch_size = 32
transform = transforms.Compose([
    transforms.Resize((28, 28)),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

train_dataset = datasets.MNIST(root='Assignment6/', train=True, download=True, transform=transform)
test_dataset = datasets.MNIST(root='Assignment6/', train=False, download=True, transform=transform)

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=2)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False, num_workers=2)

print("train dataset: ", len(train_dataset))
print("test dataset: ", len(test_dataset))


# Single hidden layer
input_size = 28 * 28
```

```python
hidden_sizes = [300]
output_size = 10

modelSHL = NeuralNet(input_size, hidden_sizes, output_size)
learning_rate = 0.1
optimizer = optim.SGD(modelSHL.parameters(), lr=learning_rate)
num_epochs = 10
criterion = nn.CrossEntropyLoss()

train(modelSHL, criterion, optimizer, train_loader, test_loader,
    num_epochs, "single_hidden_layer")
accuracy = calculate_accuracy(modelSHL, test_loader)
print(f"Accuracy for single hidden layer: {accuracy:.4f}")


# Two hidden layers
hidden_sizes = [500, 300]
weight_decay = 0.0001
modelTHL = NeuralNet(input_size, hidden_sizes, output_size)
optimizer = optim.SGD(modelTHL.parameters(), lr=learning_rate,
    weight_decay=weight_decay)
num_epochs = 40

train(modelTHL, criterion, optimizer, train_loader, test_loader,
    num_epochs, "two_hidden_layer")
accuracy = calculate_accuracy(modelTHL, test_loader)
print(f"Accuracy for two hidden layers: {accuracy:.4f}")


# Convolutional neural network
modelCNN = CNN()
weight_decay = 0.0001
optimizer = optim.SGD(modelCNN.parameters(), lr=learning_rate,
    weight_decay=weight_decay)
num_epochs = 40

train(modelCNN, criterion, optimizer, train_loader, test_loader,
    num_epochs, "cnn")
accuracy = calculate_accuracy(modelCNN, test_loader)
print(f"Accuracy for convolutional neural network: {accuracy:.4f}")
```