DAT565/DIT407 Assignment 6

Sebastian Miles
miless@chalmers.se

Olle Lapidus
ollelap@chalmers.se

2024-10-11

Problem 1

We verify that the images are 28x28 pixels grayscales and plot some images from the train dataset and some from the test dataset. See figure 1 and 2.

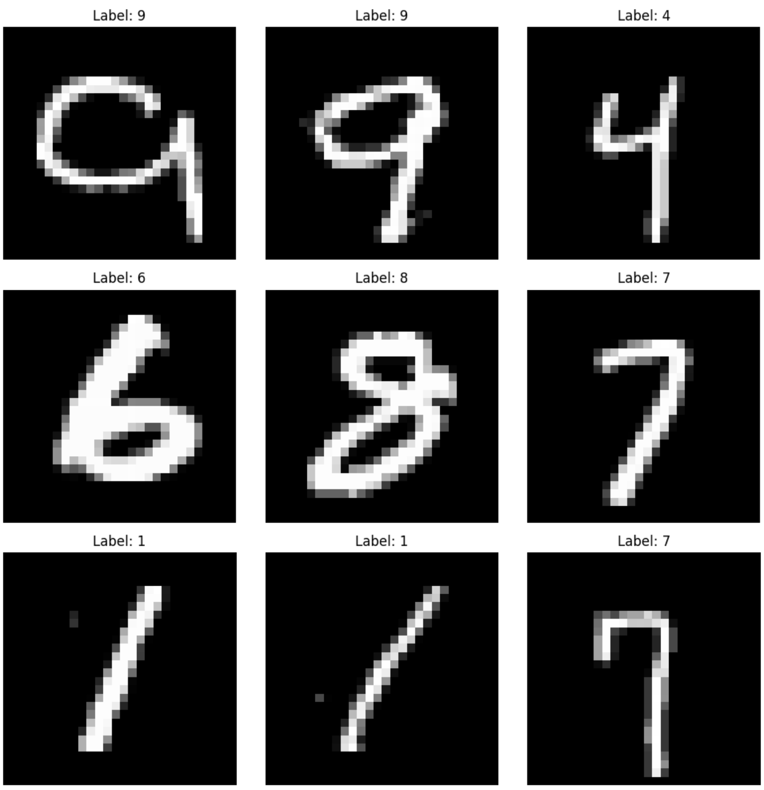

Figure 1: Various images from the train dataset

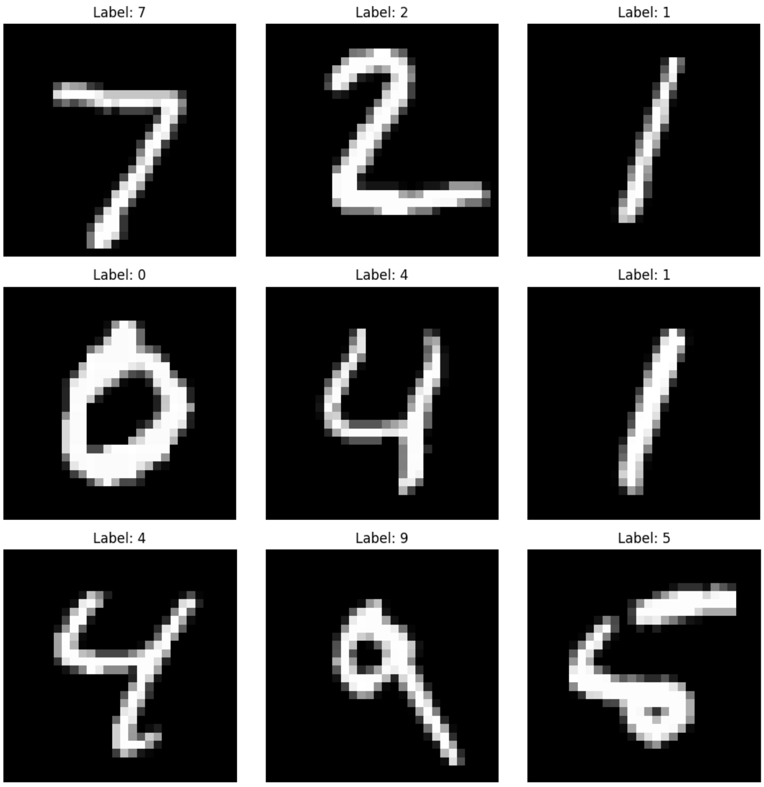

Figure 2: Various images from the test dataset

Problem 2

We train the data with a single hidden layer. We put the hidden layer size as 128 and the learning rate for the SGD as 0.01. The accuracy over 10 epochs is shown in table 1.

Epoch	Accuracy
1	79.45%
2	82.06%
3	84.35%
4	83.96%
5	84.61%
6	85.36%
7	84.10%
8	85.57%
9	84.92%
10	86.75%

Table 1: Accuracy of the test data over 10 epochs.

Problem 3

For this problem we used weight decay = 0.0001 and learning rate = 0.04. The accuracies are displayed in table 2. The accuracy seems to plateau at around 98%, which is also what we were supposed to reach.

Epoch	Accuracy
1	90.61%
2	94.65%
3	94.68%
4	95.81%
5	96.90%
6	94.19%
7	97.10%
8	97.29%
9	97.29%
10	97.19%
11	97.85%
12	97.76%
13	98.05%
14	97.92%
15	97.90%
16	97.21%
17	98.04%
18	98.12%
19	98.10%
20	98.07%
21	98.13%
22	98.02%
23	98.11%
24	97.80%
25	98.20%
26	98.23%
27	98.17%
28	98.21%
29	98.22%
30	98.24%
31	98.29%
32	98.22%
33	98.22%
34	98.28%
35	98.30%
36	98.30%
37	98.30%
38	98.36%
39	98.28%
40	98.34%

Table 2: Accuracy of the test data over 40 epochs with two hidden layers.

Problem 4

In this model we first have a convolution layer with 32 filters and 3x3 kernel, and max pooling with a 2x2 window. In the second layer we have 64 filters with a 3x3 kernel and a 2x2 max pooling. Finally we have a hidden layer with 128 neurons. In between every layer we use ReLU activation which adds non-linearity to the model. We also used the same weight decay and learning rate as in problem 3. The accuracy for 40 epochs is shown in table 3. We note that the accuracies plateau at around 99.1% meaning that we have likely reached the best accuracy for the model, and running it for more epochs would be a waste.

The $\vdots$ implies that the data continued roughly in the range 99% to 99.2%.

Epoch	Accuracy
1	97.34%
2	98.16%
3	98.50%
4	98.29%
5	98.80%
6	98.61%
7	98.88%
8	97.84%
9	98.76%
10	99.08%
11	99.17%
12	99.12%
13	99.04%
$\vdots$	$\vdots$
37	99.10%
38	99.11%
39	99.11%
40	99.15%

Table 3: Accuracy of the test data over 40 epochs with a convolution NN.

Code

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

input_size = 28 * 28
hidden_size = 128
output_size = 10
weight_decay = 0.0001
```

```
learning_rate = 0.04

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))  # Normalize to [-1, 1]
])

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

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)


def train_model(model, train_loader, test_loader, num_epochs):
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=learning_rate, weight_decay=weight_decay)

    for epoch in range(1, num_epochs + 1):
        model.train()
        for batch_idx, (data, target) in enumerate(train_loader):
            optimizer.zero_grad()
            #output = model(data.view(-1, 28*28))  # Cant do this with model3
            output = model(data)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()

        test_loss, accuracy = validate(model, test_loader, criterion)
        print(f'{epoch} & {accuracy:.2f}\\%')

    return model

def validate(model, test_loader, criterion):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad(): # disable gradient calculation for validation
        for data, target in test_loader:
```

```
            #output = model(data.view(-1, 28*28)) # Cant do this with model3
            output = model(data)
            test_loss += criterion(output, target).item()
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)
    accuracy = 100.0 * correct / len(test_loader.dataset)
    return test_loss, accuracy


model1 = nn.Sequential(
    nn.Linear(28*28, hidden_size),
    nn.ReLU(),
    nn.Linear(hidden_size, output_size)
)


#train_model(model1, train_loader, test_loader, num_epochs=10)

model2 = nn.Sequential(
    nn.Linear(28*28, 500),
    nn.ReLU(),
    nn.Linear(500, 300),
    nn.ReLU(),
    nn.Linear(300, 10),
)

#train_model(model2, train_loader, test_loader, num_epochs=40)

model3 = nn.Sequential(
    nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3, stride=1, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),

    nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),

    nn.Flatten(),
    nn.Linear(64 * 7 * 7, 128),
    nn.ReLU(),
```

```
        nn.Linear(128, 10)
)

train_model(model3, train_loader, test_loader,
    ↪ num_epochs=40)
```