DAT565/DIT407 Assignment 6

Henning Anevik Joar Forsberg

2024-10-16

Problem 1: The dataset

Start by importing the necessary libraries A. Then the using the Torchvision the data set was loaded and converted to images to Tensors B. Running the for loop containing X, y in test_loader it outputs "Shape of X [N, C, H, W]: torch.Size([1, 1, 28, 28])" & Shape of y: torch.Size([1]) torch.int64.

Problem 2: Single hidden layer

The only parameters that was chosen were the input/output features. I chose 512 output features from the inputlayer and 512 output features again, from the hidden layer to the output layer. The only real reason I chose this value was because it was the one that they used in the PyTorch tutorial. Since they were using the Fashion-MNIST dataset, in which the pictures have the same dimension and channels as the ones in regular MNIST, I figured that these values would be appropriate. The output of the epochs can be seen in Table 1. The single hidden layer is a part of a class found in C.

Epoch	**Accuracy (%)**	**Avg Loss**
1	92.0	0.268380
2	94.3	0.185549
3	95.5	0.143503
4	96.4	0.118555
5	97.0	0.102227
6	97.2	0.091949
7	97.4	0.084933
8	97.5	0.080052
9	97.7	0.076061
10	97.8	0.073988

Table 1: Test Accuracy and Average Loss over Epochs

The code the yielded the result and calling for the single_hidden_layer case can be seen in D

Problem 3: Two hidden layer

The output of the epochs can be seen in Table 2. The two hidden layer is a part of a class found in C.

Epoch	Accuracy (%)	Avg Loss
1	91.4	0.278967
2	94.7	0.173336
3	96.0	0.129509
4	96.6	0.105004
5	97.1	0.093644
6	97.2	0.087248
7	97.3	0.085166
8	97.4	0.082628
9	97.3	0.083641
10	97.4	0.082626
11	97.5	0.080177
12	97.5	0.078681
13	97.6	0.079203
14	97.7	0.075715
15	97.8	0.075144
16	97.8	0.073845
17	97.8	0.073529
18	97.8	0.073852
19	97.8	0.073472
20	97.7	0.073709
21	97.7	0.074024
22	97.7	0.073715
23	97.7	0.074576
24	97.7	0.073206
25	97.8	0.070811
26	97.8	0.070414
27	97.8	0.069112
28	97.9	0.067155
29	98.0	0.065977
30	98.0	0.065002
31	98.0	0.066218
32	98.1	0.064178
33	98.0	0.065483
34	98.0	0.064749
35	98.0	0.064682
36	98.1	0.063898
37	98.0	0.064737
38	98.0	0.064277
39	98.0	0.064417
40	98.1	0.063805

Table 2: Test Accuracy and Average Loss over 40 Epochs

The code the yielded the result and calling for the two_layers case can be seen in E

Problem 4: Convolutional neural network

The cnn function is a part of a class found in C. As this code was developed a bit earlier then the rest so it doesn't yield the same outputs as the other two cases. The output is only the last epoch, epoch 40. Which landed on an accuracy of 99.1% and a average loss of 0.033193. The network structure consists of two convolutional layers: Conv Layer 1: 32 filters, 3x3 kernel, followed by MaxPooling (2x2) & Conv Layer 2: 64 filters, 3x3 kernel, followed by MaxPooling (2x2). After each convolutional and the fully connected layer ReLU where applied. SGD where used for traning with the l2 regularization was used to prevent overfitting. Some inspiration was taken from similar cnn projects seen in [1]. So in summary over the 40 epochs the model had a accuracy of 99.1%, meeting the requirement.

The code the yielded the result and calling for the cnn case can be seen in F

References

[1] geeksforgeeks. *Applying Convolutional Neural Network on mnist dataset*. Retrieved 2024-10-16. 2024. URL: https://www.geeksforgeeks.org/applying-convolutional-neural-network-on-mnist-dataset/.

A Libraries

```
plot_samples = False
import torchvision.datasets.mnist
import matplotlib.pyplot as plt
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import torch.nn as nn
import torch.optim
```

B The dataset

```
transform = transforms.ToTensor()

# load the mnist dataset, training and testsplit
mnist_train =
    ↪ torchvision.datasets.MNIST(root='/content/data',
    ↪ train=True, download=True, transform=transform)
mnist_test =
    ↪ torchvision.datasets.MNIST(root='/content/data',
    ↪ train=False, download=True, transform=transform)

```

```
# plot an image from the training and testing set
train_sample = mnist_train[0]
test_sample = mnist_test[0]
if plot_samples:
    plt.imshow(train_sample[0].squeeze(), cmap="grey")
    plt.savefig("train_sample")
    plt.imshow(test_sample[0].squeeze(), cmap="grey")
    plt.savefig("test_sample")

train_dataloader = DataLoader(mnist_train)
test_dataloader = DataLoader(mnist_test)

for X, y in test_dataloader:
    print(f"Shape of X [N, C, H, W]: {X.shape}")
    print(f"Shape of y: {y.shape} {y.dtype}")
    break
```

C class NeuralNetwork

```
class NeuralNetwork(nn.Module):
    def __init__(self, topology):
        super().__init__()
        self.flatten = nn.Flatten()
        match topology:
            case "single_hidden_layer":
                self.linear_relu_stack =
                    ↪ nn.Sequential(
                nn.Linear(28*28, 512),
                nn.ReLU(),
                nn.Linear(512, 512),
                nn.ReLU(),
                nn.Linear(512, 10)
            )
            case "two_layers":
                self.linear_relu_stack =
                    ↪ nn.Sequential(
                nn.Linear(28*28, 500),
                nn.ReLU(),
                nn.Linear(500, 300),
                nn.ReLU(),
                nn.Linear(300, 512),
                nn.ReLU(),
                nn.Linear(512, 10)
            )
            case "cnn":
                self.conv_layers = nn.Sequential(
                nn.Conv2d(1, 32, kernel_size=3,
                    ↪ padding=1),
```

```
27                  nn.ReLU(),
28                  nn.MaxPool2d(kernel_size=2, stride=2),
29
30                  nn.Conv2d(32, 64, kernel_size=3,
                        ↪ padding=1),
31                  nn.ReLU(),
32                  nn.MaxPool2d(kernel_size=2, stride=2)
33              )
34              self.fc_layers = nn.Sequential(
35                  nn.Flatten(),
36                  nn.Linear(64 * 7 * 7, 512),
37                  nn.ReLU(),
38                  nn.Linear(512, 10)
39              )
40
41      def forward(self, x):
42          if self.topology == "cnn":
43              x = self.conv_layers(x)
44              x = self.fc_layers(x)
45              return x
46          else():
47              x = self.flatten(x)
48              logits = self.linear_relu_stack(x)
49              return logits
50
51  def train(dataloader, model, loss_fn, optimizer):
52      size = len(dataloader.dataset)
53      model.train()
54      for batch, (X, y) in enumerate(dataloader):
55          pred = model(X)
56          loss = loss_fn(pred, y)
57
58          loss.backward()
59          optimizer.step()
60          optimizer.zero_grad()
61
62          if batch % 100 == 0:
63              loss, current = loss.item(), (batch + 1)
                    ↪ * len(X)
64              #print(f"loss: {loss:>7f}
                    ↪ [{current:>5d}/{size:>5d}]")
65
66  def test(dataloader, model, loss_fn):
67      size = len(dataloader.dataset)
68      num_batches = len(dataloader)
69      model.eval()
70      test_loss, correct = 0, 0
71      with torch.no_grad():
72          for X, y in dataloader:
73              pred = model(X)
```

```
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) ==
               ↪ y).type(torch.float).sum().item()
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy:
       ↪ {(100*correct):>0.1f}%, Avg loss:
       ↪ {test_loss:>8f} \n")
```

D Single Hidden Layer

```
model = NeuralNetwork(topology="single_hidden_layer")
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(),
   ↪ lr=1e-3)

epochs = 10
for t in range(epochs):
    print(f"Epoch
       ↪ {t+1}\n-------------------------------")
    train(train_dataloader, model, loss_fn, optimizer)
print("Done!")
```

E Two Hidden Layer

```
model = NeuralNetwork(topology="two_layers")
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(),
   ↪ lr=1e-3, weight_decay=0.0005)

epochs = 40
for t in range(epochs):
    print(f"Epoch
       ↪ {t+1}\n-------------------------------")
    train(train_dataloader, model, loss_fn, optimizer)
print("Done!")
```

F Convolutional neural network

```
import torch.optim as optim
epochs = 40

model = NeuralNetwork(topology="cnn")
loss_fn = nn.CrossEntropyLoss()
```

```
optimizer = optim.SGD(model.parameters(), lr=1e-3,
    ↪ weight_decay=0.0005)

for epoch in range(epochs):
    print(f"Epoch␣
        ↪ {epoch+1}\n-------------------------------")
    train(train_dataloader, model, loss_fn, optimizer)
    test(test_dataloader, model, loss_fn)
```