

DAT565/DIT407 Assignment 6

Connell Hagen
connellh@student.chalmers.se

Måns Redin
mansre@student.chalmers.se

2024-10-16

1 Problem 1: Pre-processing the Dataset

The dataset used in this project was a MNIST dataset. This pre-defined PyTorch dataset consists of 70 000 images of handwritten digits, with their respective labels. The dataset was split into two parts: a training set of size 60 000, and a test set of the remainder. Different neural network models will be discussed and were implemented with their test accuracy showing which were most effective. The images size was verified to be 28x28, with all pixels in the normalized greyscale.

2 Problem 2: Single Hidden Layer

We constructed a fully-connected, one layer neural network using ReLU as the activation function, Stochastic Gradient Descent as the optimization function, and Cross Entropy Loss as the loss function. We did not use any special parameters for this model beyond the default. Our single layer was 200 neurons large. The total accuracy came out to be 87.17%. The accuracy after each epoch can be viewed in table 1.

Table 1: The Single-Layer Model's Validation Accuracy with Respect to the Test Set for 10 Epochs

Epoch	Accuracy
1	36.12
2	39.82
3	43.81
4	44.45
5	45.17
6	45.99
7	47.54
8	49.32
9	50.76
10	51.88

3 Problem 3: Two Hidden Layers

We expanded our network by using a second hidden layer, next. For these layers, the first was 500 neurons large, and the second was 300 neurons large. We used ReLU, SGD, and Cross Entropy Loss, like the previous model. We also enabled L2 regularization by using the weight decay parameter with a value of 0.0001. We also set the learning rate to 0.01. The total The results can be viewed in table 2.

Table 2: The Dual-Layer Model's Validation Accuracy with Respect to the Test Set for 60 Epochs

Epoch	Accuracy	Epoch	Accuracy	Epoch	Accuracy
1	71.89	21	96.31	41	97.49
2	88.39	22	96.30	42	97.53
3	90.02	23	96.37	43	97.62
4	91.14	24	96.45	44	97.65
5	92.09	25	96.61	45	97.62
6	92.38	26	96.57	46	97.70
7	93.16	27	96.66	47	97.72
8	93.50	28	96.89	48	97.74
9	93.79	29	97.03	49	97.72
10	97.04	30	97.05	50	97.81
11	94.26	31	97.09	51	97.71
12	94.55	32	97.25	52	97.74
13	94.87	33	97.31	53	97.79
14	95.05	34	97.34	54	97.75
15	95.24	35	97.30	55	97.72
16	95.26	36	97.38	56	97.90
17	95.65	37	97.37	57	97.87
18	95.85	38	97.33	58	97.94
19	96.06	39	97.49	59	97.93
20	96.18	40	97.57	60	97.82

As can be seen in table 2, the accuracy increased vastly from the neural network with one hidden layer.

4 Problem 4: Convolutional Neural Network

In this part of the project, a Convolutional Neural Network (CNN) was implemented. CNNs are designed to detect essential features in early layers, and analyze each feature further in the deeper layers. The architecture was made of two convolutional layers, a pooling layer, and 2 fully-connected layers, making it fairly computationally heavy, certainly heavier than the recent trained models. It comes, however, with an increased accuracy for image recognition. We used a weight decay value of 0.00001, and a learning rate of 0.01 for this model. We reached an accuracy of 98.9% at best, but for the last 40 epochs the accuracy oscillated around 98.5-99%.

Table 3: The Convolutional Model’s Validation Accuracy with Respect to the Test Set for 60 Epochs

Epoch	Accuracy	Epoch	Accuracy	Epoch	Accuracy
1	85.68	21	98.35	41	98.67
2	91.70	22	98.48	42	98.87
3	94.88	23	98.02	43	98.63
4	94.80	24	98.47	44	98.64
5	96.45	25	98.12	45	98.78
6	97.32	26	98.04	46	98.30
7	97.21	27	98.61	47	98.77
8	97.82	28	98.66	48	98.55
9	97.77	29	98.63	49	98.76
10	97.85	30	98.63	50	98.78
11	98.17	31	98.70	51	98.75
12	98.11	32	98.53	52	98.88
13	97.85	33	98.66	53	98.69
14	98.27	34	98.64	54	98.85
15	98.27	35	98.51	55	98.85
16	98.24	36	98.75	56	98.82
17	98.40	37	98.75	57	98.89
18	98.39	38	98.84	58	98.89
19	97.92	39	98.55	59	98.80
20	98.38	40	98.64	60	98.85

A Code

```
1
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5 import torch
6 import torch.nn.functional as F
7 from PIL import Image
8 from torch import nn
9 from torch.utils.data import DataLoader
10 from torch.nn import CrossEntropyLoss
11 from torch.optim import SGD
12 from torchvision import datasets
13 from torchvision.transforms import ToTensor
14 from tqdm import tqdm
15
16 training_data = datasets.MNIST(
17     root="./data",
18     train=True,
19     download=True,
20     transform=ToTensor()
21 )
22
23 test_data = datasets.MNIST(
24     root="./data",
25     train=False,
26     download=True,
27     transform=ToTensor()
28 )
29
30 train_dataloader = DataLoader(training_data,
31                               batch_size=64, shuffle=True)
32 test_dataloader = DataLoader(test_data, batch_size=64,
33                              shuffle=True)
34
35 train_features, train_labels = next(iter(
36     train_dataloader))
37 img = train_features[0].squeeze()
38 label = train_labels[0]
39 plt.imshow(img, cmap="gray")
40 plt.show()
41
42 test_features, test_labels = next(iter(test_dataloader
43 ))
44 img = test_features[0].squeeze()
45 label = test_labels[0]
46 plt.imshow(img, cmap="gray")
```

```

44 plt.show()
45
46
47 L1_NEURONS = 200
48
49 class OneLayer(nn.Module):
50     def __init__(self):
51         super(OneLayer, self).__init__()
52
53         self.fc1 = nn.Linear(28 * 28, L1_NEURONS)
54         self.fc2 = nn.Linear(L1_NEURONS, 10)
55
56     def forward(self, x):
57         x = F.relu(self.fc1(x));
58         x = F.relu(self.fc2(x))
59
60         return x
61
62 model = OneLayer()
63 loss_fn = CrossEntropyLoss()
64 optimizer = SGD(model.parameters())
65
66
67 epochs = 10
68 running_loss = 0
69
70 for e in range(epochs):
71     for i, data in tqdm(enumerate(train_dataloader)):
72         model.train()
73
74         inputs, labels = data
75
76         optimizer.zero_grad()
77
78         flattened_inputs = inputs.view(inputs.size(0),
79                                         -1)
80         outputs = model(flattened_inputs)
81
82         loss = loss_fn(outputs, labels)
83         loss.backward()
84
85         optimizer.step()
86
87         running_loss += loss.item()
88
89     print(f"Epoch_{e}: Average loss of {running_loss /
90           64} per batch")
91     running_loss = 0
92
93     # test benchmark

```

```

92     model.eval()
93
94     total_correct = 0
95     total_samples = 0
96
97     for i, (inputs, labels) in tqdm(enumerate(
98         test_dataloader)):
99         flattened_inputs = inputs.view(inputs.size(0),
100             -1)
101
102         outputs = model(flattened_inputs)
103         _, predicted = torch.max(outputs, 1)
104
105         total_correct += (predicted == labels).sum().
106             item()
107         total_samples += labels.size(0)
108
109     accuracy = 100 * total_correct / total_samples
110     print(f"Epoch_{e}, Accuracy:_{accuracy}")
111
112 torch.save(model.state_dict(), "model/one_layer_model.
113    .pth")
114
115 L1_NEURONS = 500
116 L2_NEURONS = 300
117
118 class TwoLayers(nn.Module):
119     def __init__(self):
120         super(TwoLayers, self).__init__()
121
122         self.fc1 = nn.Linear(28 * 28, L1_NEURONS)
123         self.fc2 = nn.Linear(L1_NEURONS, L2_NEURONS)
124         self.fc3 = nn.Linear(L2_NEURONS, 10)
125
126     def forward(self, x):
127         x = F.relu(self.fc1(x))
128         x = F.relu(self.fc2(x))
129         x = F.relu(self.fc3(x))
130
131         return x
132
133 model = TwoLayers()
134 loss_fn = CrossEntropyLoss()
135 optimizer = SGD(model.parameters(), weight_decay=1e-4,
136     lr=0.01) # weight decay for L2 regularization

```

```

137 for e in range(epochs):
138     for i, data in tqdm(enumerate(train_dataloader)):
139         model.train()
140
141         inputs, labels = data
142
143         optimizer.zero_grad()
144
145         flattened_inputs = inputs.view(inputs.size(0),
146                                         -1)
147         outputs = model(flattened_inputs)
148
149         loss = loss_fn(outputs, labels)
150         loss.backward()
151
152         optimizer.step()
153
154         running_loss += loss.item()
155
156     print(f"Epoch_{e}: Average loss of {running_loss /
157           64} per batch")
158     running_loss = 0
159
160     # test benchmark
161     model.eval()
162
163     total_correct = 0
164     total_samples = 0
165
166     for i, (inputs, labels) in tqdm(enumerate(
167         test_dataloader)):
168         flattened_inputs = inputs.view(inputs.size(0),
169                                         -1)
170
171         outputs = model(flattened_inputs)
172         _, predicted = torch.max(outputs, 1)
173
174         total_correct += (predicted == labels).sum().
175             item()
176         total_samples += labels.size(0)
177
178     accuracy = 100 * total_correct / total_samples
179     print(f"Epoch_{e}, Accuracy: {accuracy}")
180
181 torch.save(model.state_dict(), "model/2-layer-0-01-
182         learning-rate.pth")
183
184 L1_OUTPUT_CHANNELS = 16
185 L2_OUTPUT_CHANNELS = 32

```

```

181
182 class ConvNetwork(nn.Module):
183     def __init__(self):
184         super(ConvNetwork, self).__init__()
185
186         # convolutional layers
187         self.conv1 = nn.Conv2d(in_channels=1,
188                                 out_channels=L1_OUTPUT_CHANNELS,
189                                 kernel_size=3, padding=1)
188         self.conv2 = nn.Conv2d(in_channels=
189                                 L1_OUTPUT_CHANNELS, out_channels=
190                                 L2_OUTPUT_CHANNELS, kernel_size=3, padding
191                                 =1)
192
193         # pooling layer
194         self.pool = nn.MaxPool2d(kernel_size=2, stride
195                                 =2)
196
197         # fully-connected layers
198         self.fc1 = nn.Linear(L2_OUTPUT_CHANNELS * 7 *
199                               7, 128)
200         self.fc2 = nn.Linear(128, 10)
201
202     def forward(self, x):
203         x = self.pool(F.relu(self.conv1(x)))
204         x = self.pool(F.relu(self.conv2(x)))
205         x = x.view(-1, L2_OUTPUT_CHANNELS * 7 * 7) #
206             flatten images
207         x = F.relu(self.fc1(x))
208         x = self.fc2(x)
209
210         return x
211
212 model = ConvNetwork()
213 loss_fn = CrossEntropyLoss()
214 optimizer = SGD(model.parameters(), weight_decay=1e-5,
215                 lr=0.01) # weight decay for L2 regularization
216
217 epochs = 60
218 running_loss = 0
219
220 for e in range(epochs):
221     for i, data in tqdm(enumerate(train_dataloader)):
222         model.train()
223
224         inputs, labels = data
225
226         optimizer.zero_grad()

```



```

222
223         outputs = model(inputs)
224
225         loss = loss_fn(outputs, labels)
226         loss.backward()
227
228         optimizer.step()
229
230         running_loss += loss.item()
231
232     print(f"Epoch_{e}: Average loss of {running_loss /
233           64} per batch")
234     running_loss = 0
235
236     # test benchmark
237     model.eval()
238
239     total_correct = 0
240     total_samples = 0
241
242     for i, (inputs, labels) in tqdm(enumerate(
243         test_dataloader)):
244         outputs = model(inputs)
245         _, predicted = torch.max(outputs, 1)
246
247         total_correct += (predicted == labels).sum().
248             item()
249         total_samples += labels.size(0)
250
251     accuracy = 100 * total_correct / total_samples
252     print(f"Epoch_{e}, Accuracy: {accuracy}")
253
254 torch.save(model.state_dict(), "model/conv-network.pth")

```