Einführung in die Informatik
Ausarbeitung Übung 6

Julian Bertol

January 2, 2024

1 Aufgabe 1

Der Titel "'Aufgabe 1"' ist durch den konkreten Aufgabentitel zu ersetzen! Wie man mit LaTeX gut umgehen kann ist in [**?**] gut beschrieben. ... und auch in vielen Beispielen im Internet aufzufinden.

1.1 Problem

Das muss ich erst mal selbst formulieren, damit mir klar wurde was ich tun soll ...

1.2 Lösungskonzept

Installiern von Jupyter Notebook
Installieren von Python (Falls noch nicht geschehen)

```
sudo apt-get install python3
sudo apt-get install python3-pip
```

Installieren von Jupyter Notebook

```
sudo pip3 install jupyter
```

Installieren von Pytorch

```
sudo pip install torch torchvision
```

Code der Aufgabe:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import numpy as np
import torchvision

# Schritt 2: Datenbeschaffung
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

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

batch_size = 64

```

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

# Schritt 3: Erstellen des neuronalen Netzes
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 128)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = x.view(-1, 28 * 28)
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# Schritt 4: Training des Modells
model = SimpleNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum
    =0.9)

epochs = 5
train_losses = []

for epoch in range(epochs):
    running_loss = 0.0
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    train_loss = running_loss / len(train_loader)
    train_losses.append(train_loss)
    print(f'Epoch {epoch+1}/{epochs}, Training Loss: {
    train_loss}')

# Schritt 5: Evaluation des Modells
model.eval()
correct = 0
total = 0

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

```
71
72 accuracy = correct / total
73 print(f'Test Accuracy: {accuracy * 100:.2f}%')
74
75 # Optional: Visualisierung
76 def imshow(img):
77     img = img / 2 + 0.5  # Denormalisieren
78     npimg = img.numpy()
79     plt.imshow(np.transpose(npimg, (1, 2, 0)))
80     plt.show()
81
82 # Visualisierung einiger Vorhersagen
83 # Visualisierung einiger Vorhersagen
84 dataiter = iter(test_loader)
85 images, labels = next(dataiter)
86
87 imshow(torchvision.utils.make_grid(images))
88 print('GroundTruth:', ' '.join('%5s' % labels[j] for j in range
       (4)))
89
90 outputs = model(images)
91 _, predicted = torch.max(outputs, 1)
92
93 print('Predicted:', ' '.join('%5s' % predicted[j] for j in
       range(4)))
```

Beobachtet man die Relation von Loss, Epochen und Genaugkeit, so lässt sich ein absinken des Losses und der Genauigkeit ab ca. 3 Epochen beobachten. Mit 20 Epochen hat man jedoch noch ein sehr gutes Verhältnis von Dauer und Genauigkeit. Trainiert man weiter, läuft man Gefahr das Modell zu übertrainieren, was zu einer schlechteren Genauigkeit führen kann.