5_MNIST_classification

吴清柳

July 31, 2023

1 MNIST 手写数字识别 CNN

[1]:
```python
import torch
from torch import nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader, random_split
```

[2]:
```python
# Define a vanilla CNN model
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()

        self.conv_layers = nn.Sequential(
            # (n, 1, 28, 28) -> (n,32,28,28)
            nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            # (n,32,28,28)->(n,32,14,14)
            nn.MaxPool2d(kernel_size=2, stride=2),
            # (n,32,14,14)->(n,64,14,14)
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            # (n,64,14,14)->(n,64,7,7)
            nn.MaxPool2d(kernel_size=2, stride=2),
        )

        self.fc_layers = nn.Sequential(
            # (n,64*7*7)->(n,128)
            nn.Linear(64 * 7 * 7, 128),
            nn.ReLU(),
            # (n,128)->(n,10)
            nn.Linear(128, 10),
        )

    def forward(self, x):
        x = self.conv_layers(x)
        x = x.view(x.size(0), -1)  # flatten the tensor
```

```
        x = self.fc_layers(x)
        return x
```

1.1 train the model

[3]:
```
# Get mean and std of dataset for normalizing the data
trainset = datasets.MNIST(
    root="../dataset",
    train=True,
    download=True,
    transform=transforms.ToTensor(),
)

# Prepare the data
# Normalize data based on its mean and std
print(list(trainset.data.size()))
dataloader = DataLoader(trainset, batch_size=len(trainset), shuffle=True)
# iterate through the dataloader
for images, _ in dataloader:
    # calculate mean and std
    mean = torch.mean(images)
    std = torch.std(images)
    break  # just once, for batch_size=len(trainset)
print("mean: ", mean, "std: ", std)

transform = transforms.Compose(
    [transforms.ToTensor(), transforms.Normalize(mean, std)]
)
```

```
[60000, 28, 28]
mean:  tensor(0.1307) std:  tensor(0.3081)
```

[4]:
```
# load traning data, split as training and validation data
train_data = datasets.MNIST(
    root="../dataset", train=True, download=True, transform=transform
)
num_train = len(train_data)
valid_size = int(0.1 * num_train)
train_size = num_train - valid_size
train_dataset, valid_dataset = random_split(
    train_data, [train_size, valid_size]
)

trainloader = DataLoader(train_dataset, batch_size=64, shuffle=True)
validloader = DataLoader(valid_dataset, batch_size=64, shuffle=False)
```

[5]:
```python
# load the test data
test_data = datasets.MNIST(
    root="../dataset", train=False, download=True, transform=transform
)
testloader = DataLoader(test_data, batch_size=64, shuffle=False)
```

[6]:
```python
device = torch.device("cuda" if torch.cuda.is_available else "cpu")
# initialize the model
model = CNN().to(device)

# set up the loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

[9]:
```python
# training loop
# use running_loss to record the accumulated loss over one epoch, and use
# average loss to indicate the performance of model.
num_epochs = 10
for epoch in range(num_epochs):  # loop over the dataset multiple times
    running_loss = 0.0
    model.train()  # set model to train mode
    for inputs, labels in trainloader:
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = inputs.to(device), labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward, backward, optimize
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
    print(f"Epoch {epoch+1},loss: {running_loss/len(trainloader)}")

    # validation phase
    model.eval()  # set the model to evaluation mode
    with torch.no_grad():
        running_loss = 0.0
        correct_predictions = 0
        total_predictions = 0
        for inputs, labels in validloader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
```

```
            running_loss += loss.item()

            _, predicted = torch.max(outputs.data, 1)
            total_predictions += labels.size(0)
            correct_predictions += (predicted == labels).sum().item()

        print(
            f"Validation loss: {running_loss/len(validloader)}, Validation␣
↪accuracy: {correct_predictions/total_predictions*100}%"
        )

print("Finished Training")
```

```
Epoch 1,loss: 0.028304429684096456
Validation loss: 0.04271628986208135, Validation accuracy: 98.81666666666666%
Epoch 2,loss: 0.021274503798435967
Validation loss: 0.03697977416115773, Validation accuracy: 99.15%
Epoch 3,loss: 0.01537711199552354
Validation loss: 0.049520803215118384, Validation accuracy: 98.75%
Epoch 4,loss: 0.01282685359298321
Validation loss: 0.0530124326333533, Validation accuracy: 98.68333333333334%
Epoch 5,loss: 0.011363748343382605
Validation loss: 0.0417140398139383, Validation accuracy: 99.1%
Epoch 6,loss: 0.0090890347068663
Validation loss: 0.04158499751673012, Validation accuracy: 99.06666666666666%
Epoch 7,loss: 0.007120275658035777
Validation loss: 0.044122362386646116, Validation accuracy: 99.16666666666667%
Epoch 8,loss: 0.007252769251623118
Validation loss: 0.04512742588576778, Validation accuracy: 99.16666666666667%
Epoch 9,loss: 0.00626154085346377
Validation loss: 0.051429426284004605, Validation accuracy: 99.0%
Epoch 10,loss: 0.005433234465427955
Validation loss: 0.050215302291842415, Validation accuracy: 99.13333333333333%
Finished Training
```

[10]:

```
# Testing phase
model.eval()
with torch.no_grad():
    correct_predictions = 0
    total_predictions = 0
    for inputs, labels in testloader:
        inputs, labels = inputs.to(device), labels.to(device)

        outputs = model(inputs)

        _, predicted = torch.max(outputs.data, 1)
```

```
        total_predictions += labels.size(0)
        correct_predictions += (predicted == labels).sum().item()

    print(f"Test Accuracy: {correct_predictions/total_predictions*100}%")
```

```
Test Accuracy: 99.16%
```

[ ]:

```

```