计算机视觉作业 6

张瀚文 2201212865

1. 作业要求

在 W6_MNIST_FC.ipynb 基础上，增加卷积层结构/增加 dropout 或者 BN 技术等，训练出尽可能高的 MNIST 分类效果。

2. 实验过程

2.1 导入数据，构建训练集和测试集

```
[3]
import torch
import torch.nn as nn
import torch.utils.data as Data
import torchvision
import torch.nn.functional as F
import numpy as np

# torch.manual_seed(1)

EPOCH = 20
LR = 0.001
DOWNLOAD_MNIST = True

train_data = torchvision.datasets.MNIST(root='./mnist/', train=True, transform=torchvision.transforms.ToTensor(),
                                        download=DOWNLOAD_MNIST, )
test_data = torchvision.datasets.MNIST(root='./mnist/', train=False)

print(train_data.train_data.shape)

train_x = torch.unsqueeze(train_data.train_data, dim=1).type(torch.FloatTensor) / 255.
train_y = train_data.train_labels
print(train_x.shape)

test_x = torch.unsqueeze(test_data.test_data, dim=1).type(torch.FloatTensor)[:2000] / 255.  # Tensor on GPU
test_y = test_data.test_labels[:2000]

...
torch.Size([60000, 28, 28])
torch.Size([60000, 1, 28, 28])
```

2.2 构建两层全连接神经网络模型（无卷积层）

data_size = 50000，采用 50000 条数据训练

batch_size = 10，每次使用 10 条数据更新梯度

神经网络模型由两个全连接层组成，激活函数使用 relu 函数，其中第一层的大小是 784*256，第二层的大小是 256*10，最后输出一个 10 维的向量作为预测结果

```
class FC(nn.Module):
    def __init__(self):
        super(FC, self).__init__()
        self.fc1 = nn.Linear(784, 256)
        self.fc2 = nn.Linear(256, 10)
#         self.fc3 = nn.Linear(10, 10)

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
#         x = F.relu(x)
#         x = self.fc3(x)

        output = x
        return output


fc = FC()

optimizer = torch.optim.Adam(fc.parameters(), lr=LR)
# loss_func = nn.MSELoss()
loss_func = nn.CrossEntropyLoss()

data_size = 50000
batch_size = 10
```

```
for epoch in range(EPOCH):
    random_indx = np.random.permutation(data_size)
    for batch_i in range(data_size // batch_size):
        indx = random_indx[batch_i * batch_size:(batch_i + 1) * batch_size]

        b_x = train_x[indx, :]
        b_y = train_y[indx]
#         print(b_x.shape)
#         print(b_y.shape)
#         pdb.set_trace()

        output = fc(b_x)

        loss = loss_func(output, b_y)

        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        if batch_i % 5000 == 0:
            test_output = fc(test_x)
            pred_y = torch.max(test_output, 1)[1].data.squeeze()
            # pred_y = torch.max(test_output, 1)[1].data.squeeze()
            accuracy = torch.sum(pred_y == test_y).type(torch.FloatTensor) / test_y.size(0)
            print('Epoch: ', epoch, '| train loss: %.4f' % loss.data.cpu().numpy(), '| test accuracy: %.3f' % accuracy)
```

[7]

2.3 构建两层卷积全连接神经网络模型

神经网络模型由两个卷积层和一个全连接层组成，激活函数使用 relu 函数，其中第一个卷积层接受 1 个通道的输入，进行窗口大小为 5 的卷积，并输出 16 个通道。第二层接受 16 个通道的输入，进行窗口大小为 5 的卷积，并输出 32 个通道。因此，在设计全连接层时应注意，其大小为（32*7*7，10），最终输出 10 维向量实现手写数字识别。

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(nn.Conv2d(in_channels=1, out_channels=16, kernel_size=5, stride=1, padding=2,),
                                   nn.ReLU(), nn.MaxPool2d(kernel_size=2),)
        self.conv2 = nn.Sequential(nn.Conv2d(16, 32, 5, 1, 2), nn.ReLU(), nn.MaxPool2d(2),)
        self.out = nn.Linear(32 * 7 * 7, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.size(0), -1)

        output = self.out(x)
        return output


cnn = CNN()

optimizer = torch.optim.Adam(cnn.parameters(), lr=LR)
# loss_func = nn.MSELoss()
loss_func = nn.CrossEntropyLoss()

data_size = 50000
batch_size = 10
```

```
for epoch in range(EPOCH):
    random_indx = np.random.permutation(data_size)
    for batch_i in range(data_size // batch_size):
        indx = random_indx[batch_i * batch_size:(batch_i + 1) * batch_size]

        b_x = train_x[indx, :]
        b_y = train_y[indx]
#         print(b_x.shape)
#         print(b_y.shape)
#         pdb.set_trace()

        output = cnn(b_x)

        loss = loss_func(output, b_y)

        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        if batch_i % 5000 == 0:
            test_output = cnn(test_x)
            pred_y = torch.max(test_output, 1)[1].data.squeeze()
            # pred_y = torch.max(test_output, 1)[1].data.squeeze()
            accuracy = torch.sum(pred_y == test_y).type(torch.FloatTensor) / test_y.size(0)
            print('Epoch: ', epoch, '| train loss: %.4f' % loss.data.cpu().numpy(), '| test accuracy: %.3f' % accuracy)
```

[8]

2.4 构建卷积全连接神经网络模型，加入 BN 和 dropout 层

在 2.3 构建的神经网络模型的基础上，加入 Batch Normalization 和 Dropout 技术，其中 Dropout 的概率设置为 0.25，最终由全连接层输出 10 维向量实现手写数字识别。

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(nn.Conv2d(in_channels=1, out_channels=16, kernel_size=5, stride=1, padding=2,),
                                   nn.ReLU(), nn.MaxPool2d(kernel_size=2),)
        self.conv2 = nn.Sequential(nn.Conv2d(16, 32, 5, 1, 2), nn.ReLU(), nn.MaxPool2d(2),)
        self.fc = nn.Linear(32 * 7 * 7, 200)
        self.bn = nn.BatchNorm1d(200)
        self.dropout = nn.Dropout(p=0.25)
        self.out = nn.Linear(200, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        x = self.bn(x)
        x = F.relu(x)
        x = self.dropout(x)
        output = self.out(x)
        return output


cnn = CNN()

optimizer = torch.optim.Adam(cnn.parameters(), lr=LR)
# loss_func = nn.MSELoss()
loss_func = nn.CrossEntropyLoss()

data_size = 50000
batch_size = 10
```

```
for epoch in range(EPOCH):
    random_indx = np.random.permutation(data_size)
    for batch_i in range(data_size // batch_size):
        indx = random_indx[batch_i * batch_size:(batch_i + 1) * batch_size]

        b_x = train_x[indx, :]
        b_y = train_y[indx]
#         print(b_x.shape)
#         print(b_y.shape)
#         pdb.set_trace()

        output = cnn(b_x)

        loss = loss_func(output, b_y)

        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        if batch_i % 5000 == 0:
            test_output = cnn(test_x)
            pred_y = torch.max(test_output, 1)[1].data.squeeze()
            # pred_y = torch.max(test_output, 1)[1].data.squeeze()
            accuracy = torch.sum(pred_y == test_y).type(torch.FloatTensor) / test_y.size(0)
            print('Epoch: ', epoch, '| train loss: %.4f' % loss.data.cpu().numpy(), '| test accuracy: %.3f' % accuracy)
```

[14]

3. 实验结果

实验结果表明，在模型中加入卷积层对预测精度的提升是非常显著的。然而，加入 Batch Normalization 和 Dropout 技术对预测精度却没有很大提升，原因可能是加入的位置不恰当或参数设置不当。下面展示 2.4 模型的运行结果，可以看出，经过 20 个 epoch 的训练，测试集上预测准确率可达 99.0%左右。

```
Epoch:  0 | train loss: 2.4955 | test accuracy: 0.295
Epoch:  1 | train loss: 0.0040 | test accuracy: 0.984
Epoch:  2 | train loss: 0.0003 | test accuracy: 0.985
Epoch:  3 | train loss: 0.0007 | test accuracy: 0.987
Epoch:  4 | train loss: 0.0030 | test accuracy: 0.984
Epoch:  5 | train loss: 0.0013 | test accuracy: 0.987
Epoch:  6 | train loss: 0.0311 | test accuracy: 0.989
Epoch:  7 | train loss: 0.0015 | test accuracy: 0.987
Epoch:  8 | train loss: 0.0005 | test accuracy: 0.986
Epoch:  9 | train loss: 0.0029 | test accuracy: 0.984
Epoch:  10 | train loss: 0.0060 | test accuracy: 0.989
Epoch:  11 | train loss: 0.0005 | test accuracy: 0.989
Epoch:  12 | train loss: 0.0000 | test accuracy: 0.987
Epoch:  13 | train loss: 0.6766 | test accuracy: 0.989
Epoch:  14 | train loss: 0.0003 | test accuracy: 0.990
Epoch:  15 | train loss: 0.0004 | test accuracy: 0.988
Epoch:  16 | train loss: 0.0001 | test accuracy: 0.990
Epoch:  17 | train loss: 0.0001 | test accuracy: 0.989
Epoch:  18 | train loss: 0.0000 | test accuracy: 0.988
Epoch:  19 | train loss: 0.7946 | test accuracy: 0.990
```