3.10 多层感知机的简洁实现

下面我们使用PyTorch来实现上一节中的多层感知机。首先导入所需的包或模块。

```
import torch
from torch import nn
from torch.nn import init
import numpy as np
import sys
sys.path.append("..")
import d2lzh_pytorch as d2l
```

3.10.1 定义模型

和softmax回归唯一的不同在于，我们多加了一个全连接层作为隐藏层。它的隐藏单元个数为256，并使用ReLU函数作为激活函数。

```
num_inputs, num_outputs, num_hiddens = 784, 10, 256

net = nn.Sequential(
        d2l.FlattenLayer(),
        nn.Linear(num_inputs, num_hiddens),
        nn.ReLU(),
        nn.Linear(num_hiddens, num_outputs),
        )

for params in net.parameters():
    init.normal_(params, mean=0, std=0.01)
```

3.10.2 读取数据并训练模型

我们使用与3.7节中训练softmax回归几乎相同的步骤来读取数据并训练模型。

> **注：由于这里使用的是PyTorch的SGD而不是d2lzh_pytorch里面的sgd，所以就不存在3.9节那样学习率看起来很大的问题了。**

```
batch_size = 256
train_iter, test_iter = d2l.load_data_fashion_mnist(batch_size)
loss = torch.nn.CrossEntropyLoss()

optimizer = torch.optim.SGD(net.parameters(), lr=0.5)

num_epochs = 5
d2l.train_ch3(net, train_iter, test_iter, loss, num_epochs, batch_size, None, None, o
```

输出：

Copy to clipboard

```
epoch 1, loss 0.0030, train acc 0.712, test acc 0.744
epoch 2, loss 0.0019, train acc 0.823, test acc 0.821
epoch 3, loss 0.0017, train acc 0.844, test acc 0.842
epoch 4, loss 0.0015, train acc 0.856, test acc 0.842
epoch 5, loss 0.0014, train acc 0.864, test acc 0.818
```

小结

- 通过PyTorch可以更简洁地实现多层感知机。