3.9 多层感知机的从零开始实现

我们已经从上一节里了解了多层感知机的原理。下面，我们一起来动手实现一个多层感知机。首先导入实现所需的包或模块。

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

3.9.1 获取和读取数据

这里继续使用Fashion-MNIST数据集。我们将使用多层感知机对图像进行分类。

```
batch_size = 256
train_iter, test_iter = d2l.load_data_fashion_mnist(batch_size)
```

3.9.2 定义模型参数

我们在3.6节（softmax回归的从零开始实现）里已经介绍了，Fashion-MNIST数据集中图像形状为 28×28，类别数为10。本节中我们依然使用长度为 $28 \times 28 = 784$ 的向量表示每一张图像。因此，输入个数为784，输出个数为10。实验中，我们设超参数隐藏单元个数为256。

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

W1 = torch.tensor(np.random.normal(0, 0.01, (num_inputs, num_hiddens)), dtype=torch.f
b1 = torch.zeros(num_hiddens, dtype=torch.float)
W2 = torch.tensor(np.random.normal(0, 0.01, (num_hiddens, num_outputs)), dtype=torch.
b2 = torch.zeros(num_outputs, dtype=torch.float)

params = [W1, b1, W2, b2]
for param in params:
    param.requires_grad_(requires_grad=True)
```

3.9.3 定义激活函数

这里我们使用基础的 `max` 函数来实现ReLU，而非直接调用 `relu` 函数。

```
def relu(X):
    return torch.max(input=X, other=torch.tensor(0.0))
```

3.9.4 定义模型

同softmax回归一样，我们通过 `view` 函数将每张原始图像改成长度为 `num_inputs` 的向量。然后我们实现上一节中多层感知机的计算表达式。

```
def net(X):
    X = X.view((-1, num_inputs))
    H = relu(torch.matmul(X, W1) + b1)
    return torch.matmul(H, W2) + b2
```

3.9.5 定义损失函数

为了得到更好的数值稳定性，我们直接使用PyTorch提供的包括softmax运算和交叉熵损失计算的函数。

```
loss = torch.nn.CrossEntropyLoss()
```

3.9.6 训练模型

训练多层感知机的步骤和3.6节中训练softmax回归的步骤没什么区别。我们直接调用 `d2lzh_pytorch` 包中的 `train_ch3` 函数，它的实现已经在3.6节里介绍过。我们在这里设超参数迭代周期数为5，学习率为100.0。

> **注：由于原书的mxnet中的 `SoftmaxCrossEntropyLoss` 在反向传播的时候相对于沿batch维求和了，而PyTorch默认的是求平均，所以用PyTorch计算得到的loss比mxnet小很多（大概是maxnet计算得到的1/batch_size这个量级），所以反向传播得到的梯度也小很多，所以为了得到差不多的学习效果，我们把学习率调得成原书的约batch_size倍，原书的学习率为0.5，这里设置成100.0。(之所以这么大，应该是因为d2lzh_pytorch里面的sgd函数在更新的时候除以了batch_size，其实PyTorch在计算loss的时候已经除过一次了，sgd这里应该不用除了)**

```
num_epochs, lr = 5, 100.0
d2l.train_ch3(net, train_iter, test_iter, loss, num_epochs, batch_size, params, lr)
```

输出：

Copy to clipboard

```
epoch 1, loss 0.0030, train acc 0.714, test acc 0.753
epoch 2, loss 0.0019, train acc 0.821, test acc 0.777
epoch 3, loss 0.0017, train acc 0.842, test acc 0.834
epoch 4, loss 0.0015, train acc 0.857, test acc 0.839
epoch 5, loss 0.0014, train acc 0.865, test acc 0.845
```

小结

- 可以通过手动定义模型及其参数来实现简单的多层感知机。
- 当多层感知机的层数较多时，本节的实现方法会显得较烦琐，例如在定义模型参数的时候。