3.7 softmax回归的简洁实现

我们在3.3节（线性回归的简洁实现）中已经了解了使用Pytorch实现模型的便利。下面，让我们再次使用Pytorch来实现一个softmax回归模型。首先导入所需的包或模块。

python

```
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
```

3.7.1 获取和读取数据

我们仍然使用Fashion-MNIST数据集和上一节中设置的批量大小。

python

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

3.7.2 定义和初始化模型

在3.4节（softmax回归）中提到，softmax回归的输出层是一个全连接层，所以我们用一个线性模块就可以了。因为前面我们数据返回的每个batch样本 `x` 的形状为(batch_size, 1, 28, 28), 所以我们要先用 `view()` 将 `x` 的形状转换成(batch_size, 784)才送入全连接层。

python

```
num_inputs = 784
num_outputs = 10

class LinearNet(nn.Module):
    def __init__(self, num_inputs, num_outputs):
        super(LinearNet, self).__init__()
```

```
            self.linear = nn.Linear(num_inputs, num_outputs)
        def forward(self, x): # x shape: (batch, 1, 28, 28)
            y = self.linear(x.view(x.shape[0], -1))
            return y

    net = LinearNet(num_inputs, num_outputs)
```

我们将对 `x` 的形状转换的这个功能自定义一个 `FlattenLayer` 并记录在 `d2lzh_pytorch` 中方便后面使用。

python

```
    # 本函数已保存在d2lzh_pytorch包中方便以后使用
    class FlattenLayer(nn.Module):
        def __init__(self):
            super(FlattenLayer, self).__init__()
        def forward(self, x): # x shape: (batch, *, *, ...)
            return x.view(x.shape[0], -1)
```

这样我们就可以更方便地定义我们的模型：

python

```
    from collections import OrderedDict

    net = nn.Sequential(
        # FlattenLayer(),
        # nn.Linear(num_inputs, num_outputs)
        OrderedDict([
            ('flatten', FlattenLayer()),
            ('linear', nn.Linear(num_inputs, num_outputs))
        ])
    )
```

然后，我们使用均值为0、标准差为0.01的正态分布随机初始化模型的权重参数。

python

```
    init.normal_(net.linear.weight, mean=0, std=0.01)
    init.constant_(net.linear.bias, val=0)
```

3.7.3 softmax和交叉熵损失函数

如果做了上一节的练习，那么你可能意识到了分开定义softmax运算和交叉熵损失函数可能会造成数值不稳定。因此，PyTorch提供了一个包括softmax运算和交叉熵损失计算的函数。它的数值稳定性更好。

python

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

3.7.4 定义优化算法

我们使用学习率为0.1的小批量随机梯度下降作为优化算法。

python

```
optimizer = torch.optim.SGD(net.parameters(), lr=0.1)
```

3.7.5 训练模型

接下来，我们使用上一节中定义的训练函数来训练模型。

python

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

输出：

```
epoch 1, loss 0.0031, train acc 0.745, test acc 0.790
epoch 2, loss 0.0022, train acc 0.812, test acc 0.807
epoch 3, loss 0.0021, train acc 0.825, test acc 0.806
epoch 4, loss 0.0020, train acc 0.832, test acc 0.810
epoch 5, loss 0.0019, train acc 0.838, test acc 0.823
```

小结

- PyTorch提供的函数往往具有更好的数值稳定性。
- 可以使用PyTorch更简洁地实现softmax回归。

> 注：本节除了代码之外与原书基本相同，原书传送门