动手学深度学习-线性回归的简单实现

本节介绍如何使用深度学习框架实现线性回归模型。

一、生成数据集

```
import numpy as np
import torch
from torch.utils import data
from d2l import torch as d2l

true_w = torch.tensor([2,-3.4])
true_b = 4.2

# 调用d2l包中生成数据函数，features是X，一个1000行2列的张量
# labels是一个1000行一列的张量 也就是y
features,labels = d2l.synthetic_data(true_w,true_b,1000)
```

二、读取数据集

调用框架中现有的API来读取数据。我们每次只选取部分数据集进行训练。将上面的features和labels作为API的参数进行传递，并通过数据迭代器指定batch_size。此外布尔值is_train表示是否希望数据迭代器对象在每个迭代周期内打乱数据。

```
def load_array(data_arrays,batch_size,is_train=True):
    """构造一个pytorch数据迭代器"""
    dataset = data.TensorDataset(*data_arrays)  # TensorDataset对张量进行打包

    # dataloader进行数据封装
    return data.DataLoader(dataset,batch_size,shuffle=is_train)

batch_size = 10

# 返回的是一个迭代器  每次加载batch_size批量的数据
data_iter = load_array((features,labels),batch_size)

next(iter(data_iter))
```

```
[tensor([[ 0.4959,  0.8714],
         [ 0.7823, -1.7682],
         [-0.1917,  0.1726],
         [-2.0061, -1.2517],
         [-0.2063, -0.1480],
         [ 0.8134, -0.9006],
         [ 0.1648,  0.4840],
```

```
          [ 0.4924, -0.0704],
          [-0.6901, -0.2232],
          [-0.2761, -1.3795]]),
 tensor([[ 2.2264],
          [11.7656],
          [ 3.2199],
          [ 4.4502],
          [ 4.2977],
          [ 8.8826],
          [ 2.8716],
          [ 5.4167],
          [ 3.5887],
          [ 8.3447]])]
```

关于pytorch的TensorDataset：对张量tensor进行打包。

三、定义模型

对于标准深度学习模型，我们可以使用框架预先定义好的层，我们只需要关注那些层用来构建模型，而不必关注层的实现细节。我们首先定义一个模型变量net,它是一个Sequential类的实例。Sequential类将多各层串联到一起。当给定输入数据时，Sequential实例将数据传入到第一层，然后将第一层的输出作为第二层的输入。但是线性回归模型只有一层，称之为全连接层。

在Pytorch中，全连接层在Linear类中定义。将两个参数传递到nn.Linear中。第一个指定输入特征形状，即2，第二个指定输出特征形状，输出特征形状为单个标量，因此为1。

```
# 定义模型 全连接层
from torch import nn
# nn 是神经网络的缩写
net = nn.Sequential(nn.Linear(2,1))
```

四、初始化模型参数

在使用net之前，需要初始化模型参数。一般指定每一个权重参数w从均值为0，标准差为0.01的正态分布中随机采样，偏置b = 0

由于线性回归是单层网络结构，所以我们使用net[0]选择网络中的第一个图层，然后使用weight.data和bias.data方法访问参数。然后再设定参数，那么权重w就使用normal(0,0.01)进行初始化（normal中的两个参数：均值，标准差）。

```
net[0].weight.data.normal_(0,0.01)
net[0].bias.data.fill_(0)
```

五、定义损失函数

计算均方误差使用的是MSELoss类，也成为**平方L2范数**。默认情况下，它返回所有样本损失的平均值。

六、定义优化算法

小批量随机梯度下降算法是一种优化神经网络的标准工具，指定需要优化的参数，然后设置lr(学习率).

```
trainer = torch.optim.SGD(net.parameters(),lr = 0.03)
```

七、训练

在每一个迭代周期中，我们将完整遍历一次数据集（train_data）,不停地从中获取一个小批量的输入和相应的标签。对于每一个小批量，在进行如下步骤：

- 铜鼓哦调用net(X)生成预测结果，然后与标签，计算损失函数（前向传播）。
- 进行反向传播计算梯度。
- 通过调用优化器更新模型参数。梯度下降法

```
num_epochs = 3
for epoch in range(num_epochs):
    for X, y in data_iter:
        l = loss(net(X) ,y)
        trainer.zero_grad()
        l.backward()
        trainer.step()
    l = loss(net(features), labels)
    print(f'epoch {epoch + 1}, loss {l:f}')

epoch 1, loss 0.000261
epoch 2, loss 0.000097
epoch 3, loss 0.000097
```