3.2 线性回归的从零开始实现

在了解了线性回归的背景知识之后，现在我们可以动手实现它了。尽管强大的深度学习框架可以减少大量重复性工作，但若过于依赖它提供的便利，会导致我们很难深入理解深度学习是如何工作的。因此，本节将介绍如何只利用 `Tensor` 和 `autograd` 来实现一个线性回归的训练。

首先，导入本节中实验所需的包或模块，其中的matplotlib包可用于作图，且设置成嵌入显示。

```
%matplotlib inline
import torch
from IPython import display
from matplotlib import pyplot as plt
import numpy as np
import random
```

3.2.1 生成数据集

我们构造一个简单的人工训练数据集，它可以使我们能够直观比较学到的参数和真实的模型参数的区别。设训练数据集样本数为1000，输入个数（特征数）为2。给定随机生成的批量样本特征 $\boldsymbol{X} \in \mathrm{R}^{1000\times 2}$，我们使用线性回归模型真实权重 $\boldsymbol{w} = [2, -3.4]^\mathrm{T}$ 和偏差 $b = 4.2$，以及一个随机噪声项 ϵ 来生成标签

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w} + b + \epsilon$$

其中噪声项 ϵ 服从均值为0、标准差为0.01的正态分布。噪声代表了数据集中无意义的干扰。下面，让我们生成数据集。

```
num_inputs = 2
num_examples = 1000
true_w = [2, -3.4]
true_b = 4.2
features = torch.randn(num_examples, num_inputs,
                       dtype=torch.float32)
labels = true_w[0] * features[:, 0] + true_w[1] * features[:, 1] + true_b
labels += torch.tensor(np.random.normal(0, 0.01, size=labels.size()),
                       dtype=torch.float32)
```

注意， `features` 的每一行是一个长度为2的向量，而 `labels` 的每一行是一个长度为1的向量（标量）。

```
print(features[0], labels[0])
```

输出：

```
tensor([0.8557, 0.4793]) tensor(4.2887)
```

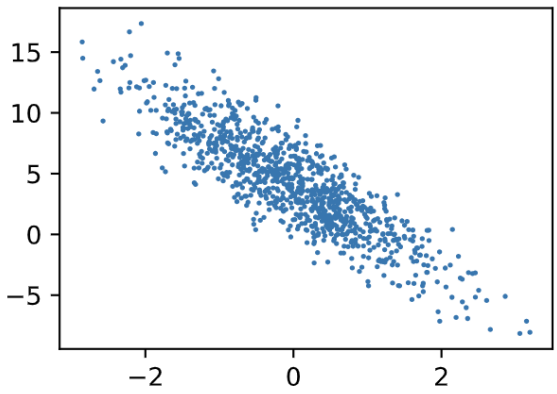
通过生成第二个特征 `features[:, 1]` 和标签 `labels` 的散点图，可以更直观地观察两者间的线性关系。

```
def use_svg_display():
    # 用矢量图显示
    display.set_matplotlib_formats('svg')

def set_figsize(figsize=(3.5, 2.5)):
    use_svg_display()
    # 设置图的尺寸
    plt.rcParams['figure.figsize'] = figsize

# # 在../d2lzh_pytorch里面添加上面两个函数后就可以这样导入
# import sys
# sys.path.append("..")
# from d2lzh_pytorch import *

set_figsize()
plt.scatter(features[:, 1].numpy(), labels.numpy(), 1);
```



我们将上面的 `plt` 作图函数以及 `use_svg_display` 函数和 `set_figsize` 函数定义在 `d2lzh_pytorch` 包里。以后在作图时，我们将直接调用 `d2lzh_pytorch.plt` 。由于 `plt` 在 `d2lzh_pytorch` 包中是一个全局变量，我们在作图前只需要调用 `d2lzh_pytorch.set_figsize()` 即可打印矢量图并设置图的尺寸。

> **原书中提到的 `d2lzh` 里面使用了mxnet，改成pytorch实现后本项目统一将原书的 `d2lzh` 改为 `d2lzh_pytorch` 。**

3.2.2 读取数据

在训练模型的时候，我们需要遍历数据集并不断读取小批量数据样本。这里我们定义一个函数：它每次返回 `batch_size` （批量大小）个随机样本的特征和标签。

```
# 本函数已保存在d2lzh包中方便以后使用
def data_iter(batch_size, features, labels):
    num_examples = len(features)
    indices = list(range(num_examples))
    random.shuffle(indices)  # 样本的读取顺序是随机的
    for i in range(0, num_examples, batch_size):
        j = torch.LongTensor(indices[i: min(i + batch_size, num_examples)]) # 最后一次
        yield  features.index_select(0, j), labels.index_select(0, j)
```

让我们读取第一个小批量数据样本并打印。每个批量的特征形状为(10, 2)，分别对应批量大小和输入个数；标签形状为批量大小。

```
batch_size = 10

for X, y in data_iter(batch_size, features, labels):
    print(X, y)
    break
```

输出：

```
tensor([[-1.4239, -1.3788],
        [ 0.0275,  1.3550],
        [ 0.7616, -1.1384],
```

```
        [ 0.2967, -0.1162],
        [ 0.0822,  2.0826],
        [-0.6343, -0.7222],
        [ 0.4282,  0.0235],
        [ 1.4056,  0.3506],
        [-0.6496, -0.5202],
        [-0.3969, -0.9951]])
 tensor([ 6.0394, -0.3365,  9.5882,  5.1810, -2.7355,  5.3873,  4.9827,  5.7962,
         4.6727,  6.7921])
```

3.2.3 初始化模型参数

我们将权重初始化成均值为0、标准差为0.01的正态随机数，偏差则初始化成0。

```
w = torch.tensor(np.random.normal(0, 0.01, (num_inputs, 1)), dtype=torch.float32)
b = torch.zeros(1, dtype=torch.float32)
```

之后的模型训练中，需要对这些参数求梯度来迭代参数的值，因此我们要让它们的 `requires_grad=True` 。

```
w.requires_grad_(requires_grad=True)
b.requires_grad_(requires_grad=True)
```

3.2.4 定义模型

下面是线性回归的矢量计算表达式的实现。我们使用 `mm` 函数做矩阵乘法。

```
def linreg(X, w, b):  # 本函数已保存在d2lzh_pytorch包中方便以后使用
    return torch.mm(X, w) + b
```

3.2.5 定义损失函数

我们使用上一节描述的平方损失来定义线性回归的损失函数。在实现中，我们需要把真实值 `y` 变形成预测值 `y_hat` 的形状。以下函数返回的结果也将和 `y_hat` 的形状相同。

```
def squared_loss(y_hat, y):  # 本函数已保存在d2lzh_pytorch包中方便以后使用
    # 注意这里返回的是向量, 另外, pytorch里的MSELoss并没有除以 2
    return (y_hat - y.view(y_hat.size())) ** 2 / 2
```

3.2.6 定义优化算法

以下的 `sgd` 函数实现了上一节中介绍的小批量随机梯度下降算法。它通过不断迭代模型参数来优化损失函数。这里自动求梯度模块计算得来的梯度是一个批量样本的梯度和。我们将它除以批量大小来得到平均值。

```
def sgd(params, lr, batch_size):  # 本函数已保存在d2lzh_pytorch包中方便以后使用
    for param in params:
        param.data -= lr * param.grad / batch_size # 注意这里更改param时用的param.data
```

3.2.7 训练模型

在训练中，我们将多次迭代模型参数。在每次迭代中，我们根据当前读取的小批量数据样本（特征 `X` 和标签 `y` ），通过调用反向函数 `backward` 计算小批量随机梯度，并调用优化算法 `sgd` 迭代模型参数。由于我们之前设批量大小 `batch_size` 为10，每个小批量的损失 `l` 的形状为(10, 1)。回忆一下自动求梯度一节。由于变量 `l` 并不是一个标量，所以我们可以调用 `.sum()` 将其求和得到一个标量，再运行 `l.backward()` 得到该变量有关模型参数的梯度。注意在每次更新完参数后不要忘了将参数的梯度清零。

在一个迭代周期（epoch）中，我们将完整遍历一遍 `data_iter` 函数，并对训练数据集中所有样本都使用一次（假设样本数能够被批量大小整除）。这里的迭代周期个数 `num_epochs` 和学习率 `lr` 都是超参数，分别设3和0.03。在实践中，大多超参数都需要通过反复试错来不断调节。虽然迭代周期数设得越大模型可能越有效，但是训练时间可能过长。而有关学习率对模型的影响，我们会在后面“优化算法”一章中详细介绍。

```
lr = 0.03
num_epochs = 3
net = linreg
```

```
    loss = squared_loss

    for epoch in range(num_epochs):  # 训练模型一共需要num_epochs个迭代周期
        # 在每一个迭代周期中，会使用训练数据集中所有样本一次（假设样本数能够被批量大小整除）。X
        # 和y分别是小批量样本的特征和标签
        for X, y in data_iter(batch_size, features, labels):
            l = loss(net(X, w, b), y).sum()  # l是有关小批量X和y的损失
            l.backward()  # 小批量的损失对模型参数求梯度
            sgd([w, b], lr, batch_size)  # 使用小批量随机梯度下降迭代模型参数

            # 不要忘了梯度清零
            w.grad.data.zero_()
            b.grad.data.zero_()
        train_l = loss(net(features, w, b), labels)
        print('epoch %d, loss %f' % (epoch + 1, train_l.mean().item()))
```

输出：

```
epoch 1, loss 0.028127
epoch 2, loss 0.000095
epoch 3, loss 0.000050
```

训练完成后，我们可以比较学到的参数和用来生成训练集的真实参数。它们应该很接近。

Copy to clipboard

```
print(true_w, '\n', w)
print(true_b, '\n', b)
```

输出：

```
[2, -3.4]
 tensor([[ 1.9998],
        [-3.3998]], requires_grad=True)
4.2
 tensor([4.2001], requires_grad=True)
```

小结

- 可以看出，仅使用 `Tensor` 和 `autograd` 模块就可以很容易地实现一个模型。接下来，本书会在此基础上描述更多深度学习模型，并介绍怎样使用更简洁的代码（见下一节）来实现它们。