线性回归：

线性模型（一个n维权重和一个标量偏差），给定n维输入，输出是输入的加权和

向量表示：y=<w,x>+b，b为标量偏差

线性模型可以看做是单层神经网络，即带权重的层只有一个（输出层）

衡量预估质量：平方损失即衡量真实值与预测值之间的误差

，1/2主要是为了求导的时候方便抵消平方的导数所产生的系数2

训练数据：收集一些数据点来决定参数值（权重和偏差），通常越多越好

参数学习：训练损失

最小化损失学习参数：

1/2来自损失函数，1/n为求平均值

显示解：因为是线性模型故有显示解，损失为一个凸函数，凸函数的最优解满足梯度为0

总结：

线性回归是对n维输入的加权，外加偏差（对输出值的预估）

使用平方损失来衡量预测值和真实值的差异

线性回归有显示解（一般来说，模型都没有显示解，有显示解的模型过于简单，复杂度有限，很难衡量复杂的数据）

线性回归可以看做是单层神经网络（最简单的神经网络）

基础优化算法：

梯度下降：给定初始值w0，重复迭代参数t=1,2,3



η为标量，表示学习率。代表沿着负梯度方向一次走多远，即步长。他是一个超参数（需要人为的指定值）

学习率的选择不能太小，也不能太大（太小会导致计算量大，求解时间长；太大的话或导致函数值振荡，并没有真正的下降）

l为损失函数，梯度是使得函数值增加最快的方向，负梯度就是使函数下降最快的方向

小批量随机梯度下降：梯度下降时，每次计算梯度，要对整个损失函数求导，损失函数是对所有样本的平均损失，所以每求一次梯度，要对整个样本的损失函数进行重新计算，计算量大且耗费时间长，代价太大。

用b个样本的平均损失来近似所有样本的平均损失，当b足够大的时候，能够保证一定的精确度

，b为批量大小，也是一个重要的超参数

线性回归的实现需要导入的包：

%matplotlib inline

import random

import torch

from d2l import torch as d2l

random：导入random包用于随机初始化权重

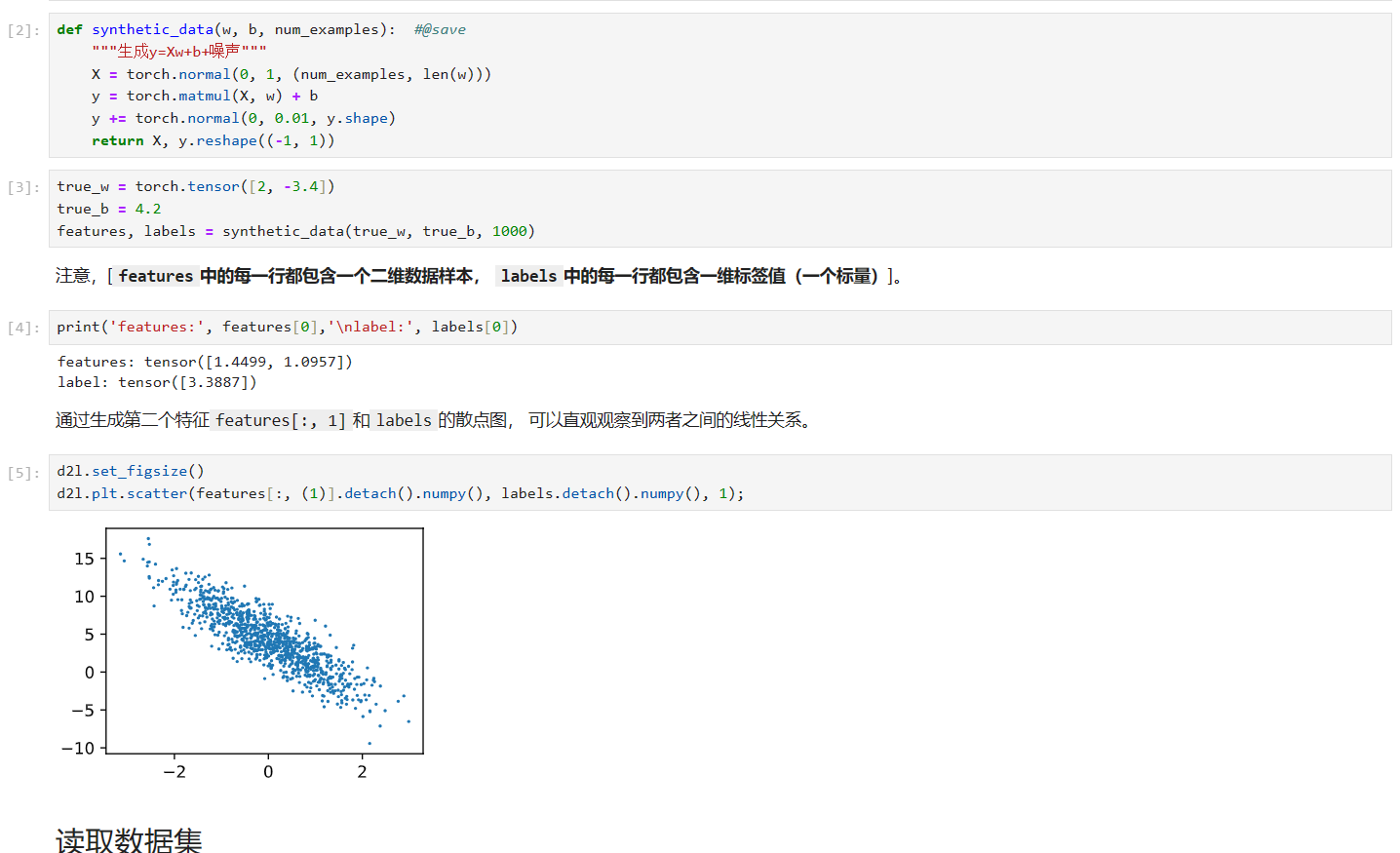
d2l：将用过的或者实现过的算法放在d2l的包里面

matplotlib inline：在plot的时候默认是嵌入到matplotlib中

报错 No module named 'matplotlib' ：在命令行中使用 pip install matplotlib 安装 matplotlib 包即可

报错 No module named 'd2l' ：在命令行中使用 pip install d2l 安装 d2l 包即可，安装完成之后可能需要重新打开程序才能生效

根据带有噪声的线性模型构造一个人造的数据集：



构造人造数据集的好处是知道真实的w和b

X = torch.normal(0,1,(num\_examples,len(w)))：X 是一个均值为 0 ，方差为 1 的随机数，他的行数等于样本数，列数等于w的长度

y += torch.normal(0,0.01,y.shape)：给y加上了一个均值为0，方差为0.01形状和y相同的噪声

return X,y.reshape((-1,1))：最后把X和y做成一个列向量返回

true\_w：真实的 w

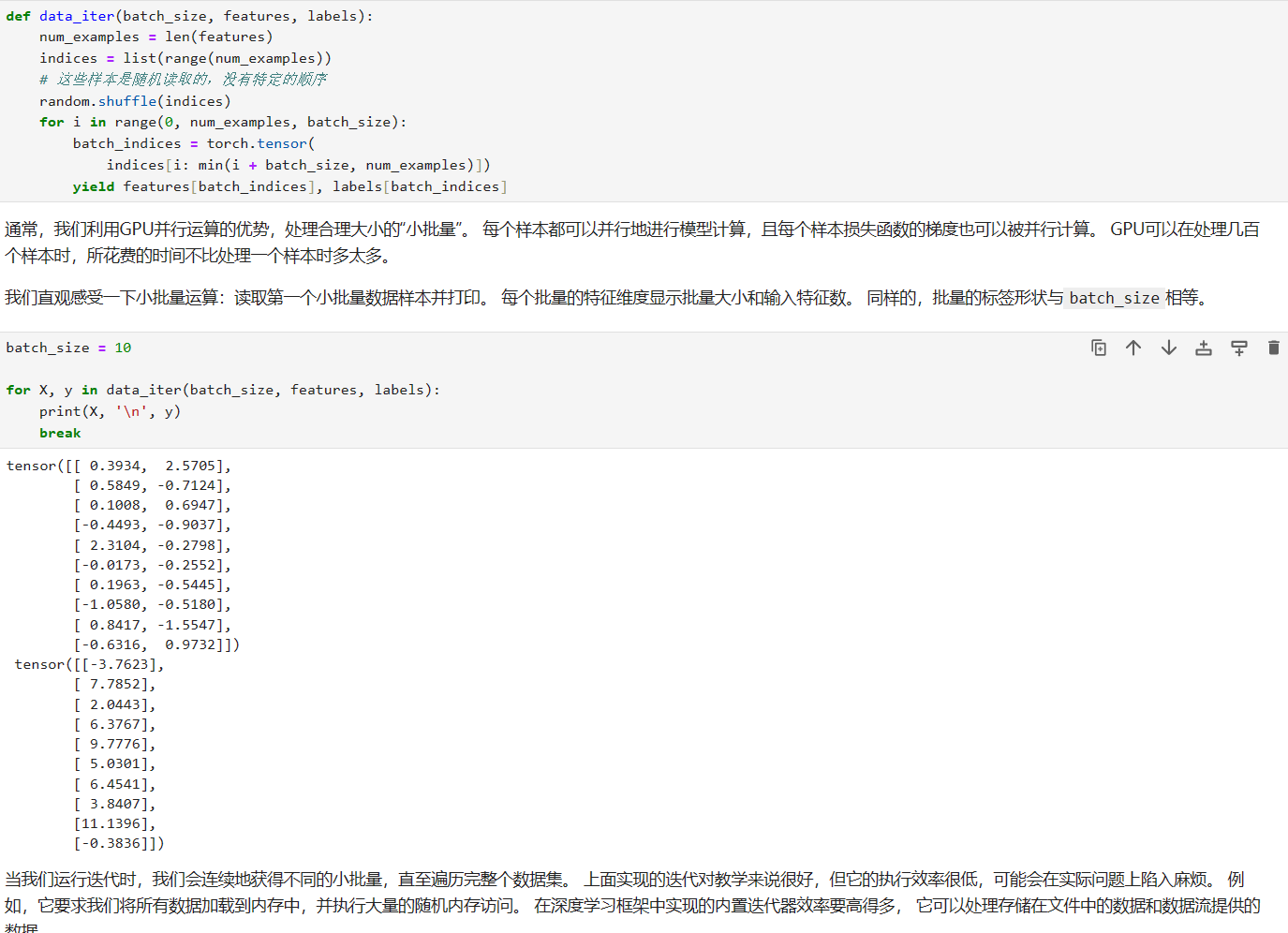
true\_b：真实的 b

features,labels = synthetic\_data(true\_w,true\_b,1000)，根据函数来生成特征和标注

detach()：在pytorch的一些版本中，需要从计算图中detach出来才能转到numpy中去

实现一个函数读取的小批量：

定义一个data\_iter函数，该函数接收批量大小、特征矩阵和标签向量作为输入，生成大小为batch\_size的小批量



batch\_size：批量大小

num\_examples：样本数

random.shuffle(indices)：将下标打乱，实现对样本的随机访问

for i in range(0,num\_examples,batch\_size)：从 0 开始到样本数结束，每次跳批量大小

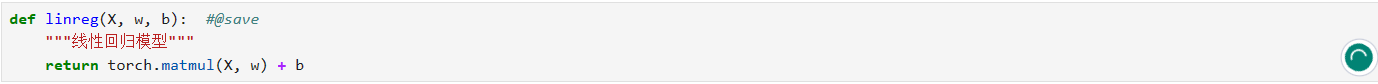
yield features[batch\_indices],labels[batch\_indices]：通过indices，每次产生随机顺序的特征和其对应的随即顺序标号

yield是python中的一个迭代器

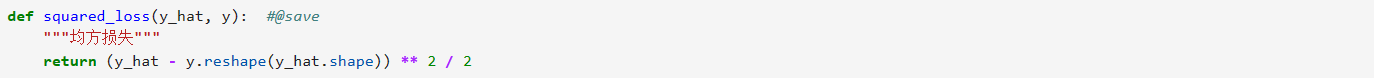
初始化模型参数：



定义模型：



定义损失函数：

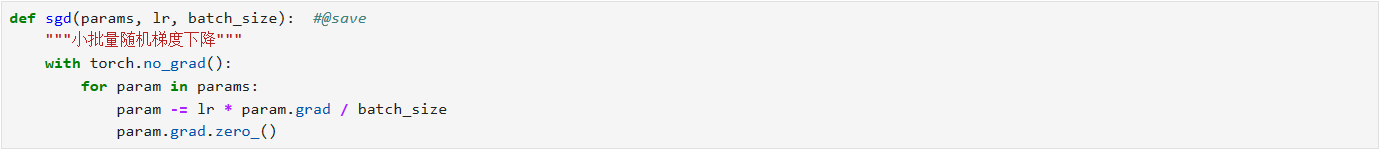


y\_hat：预测值

y：真实值

虽然 y\_hat 和 y 元素个数是一样的，但是可能他们一个是行向量一个是列向量，因此需要使用reshape进行统一

定义优化算法：



params：给定的所有参数，包含 w 和 b ，他是一个list

lr：学习率

param.grad.zero\_()：手动将梯度设置成 0 ，在下一次计算梯度的时候就不会和上一次相关了

训练过程：

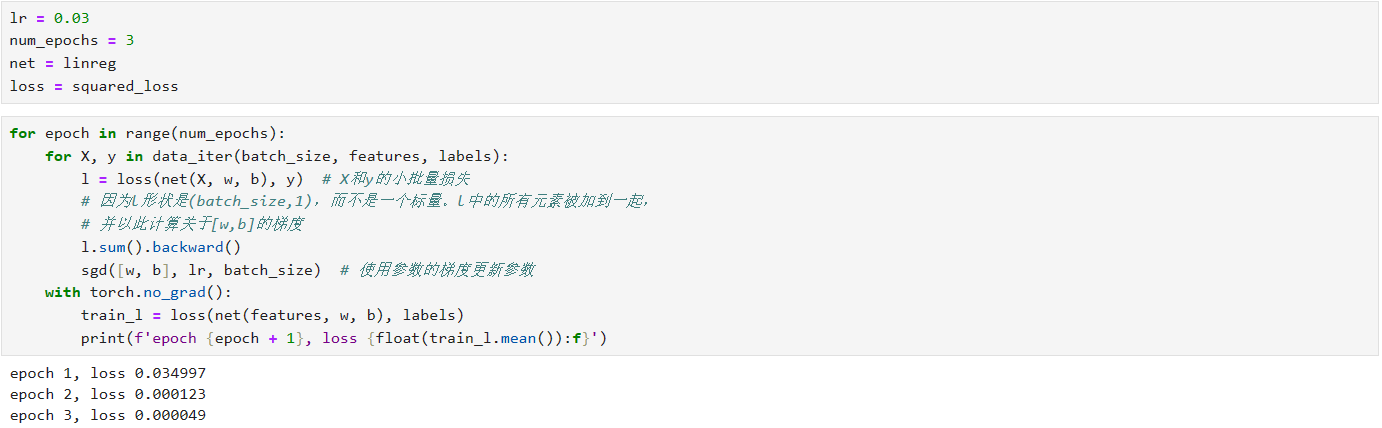
执行以下循环：

初始化参数

重复以下训练，直到完成

计算梯度

更新参数



num\_epoch=3：将整个数据扫描三遍

net：之前定义的模型

loss：均方损失

每一次对数据扫描一遍，扫描的时候拿出一定批量的X和y﻿

比较真实参数和通过训练学到的参数来评估训练的成功程度：

