线性回归

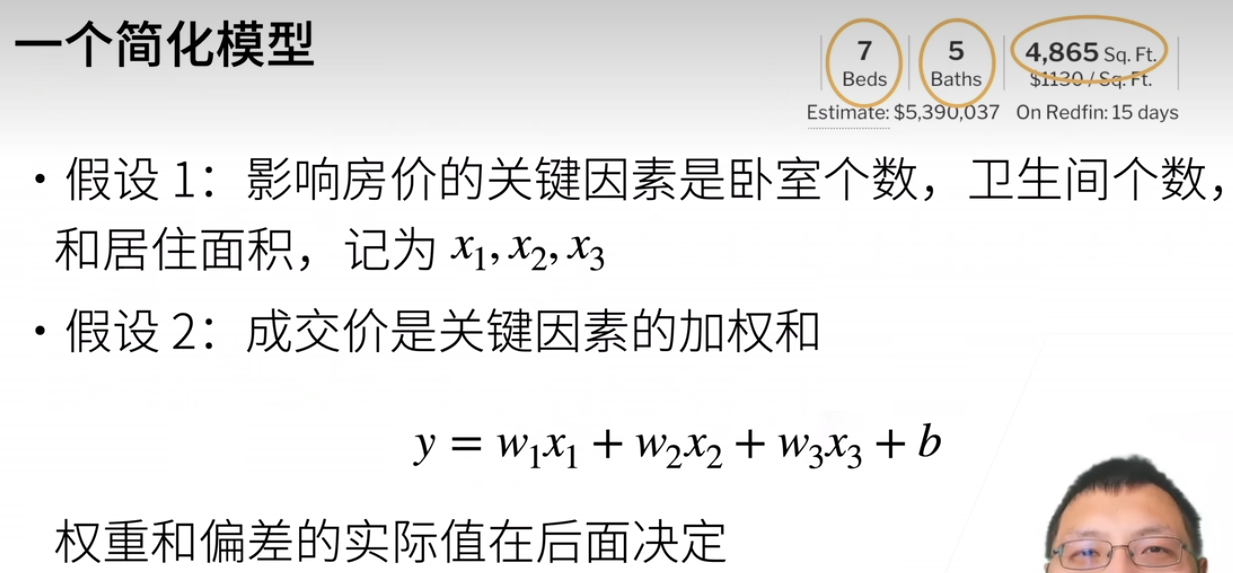
线性回归是一种统计方法，用于分析变量之间的线性关系。它通过构建一个线性方程来预测一个变量（因变量）与一个或多个自变量之间的关系。当函数为未知参数的线性函数时，称为线性回归模型。

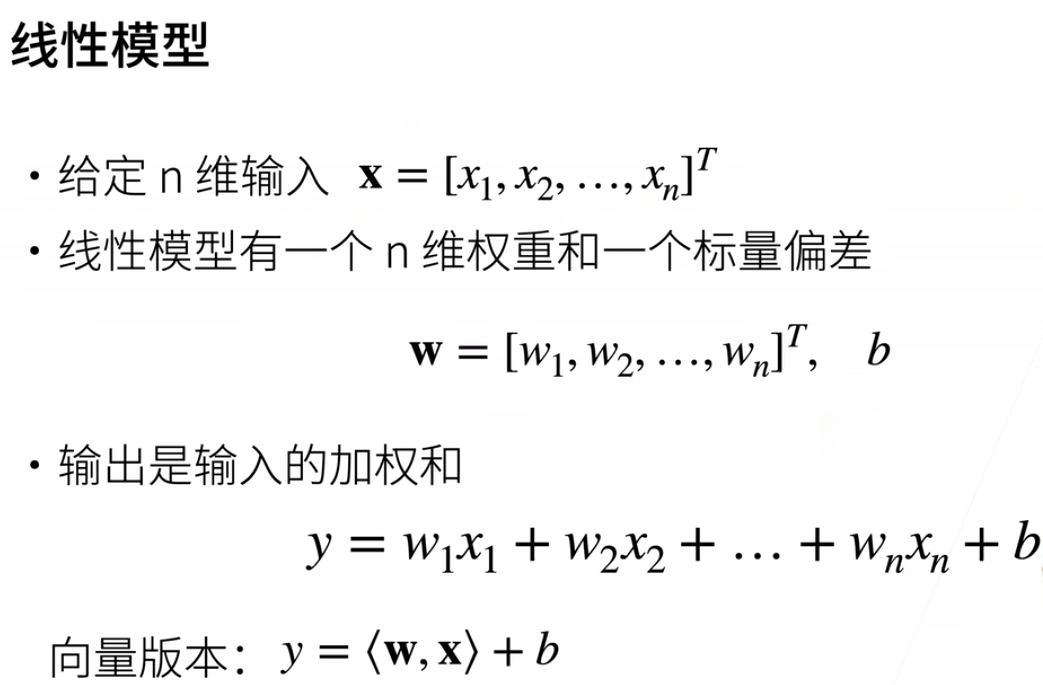
线性回归是机器学习最基础的一个模型，也是我们理解后面所有深度学习的基础。线性回归可以看作是单层神经网络！

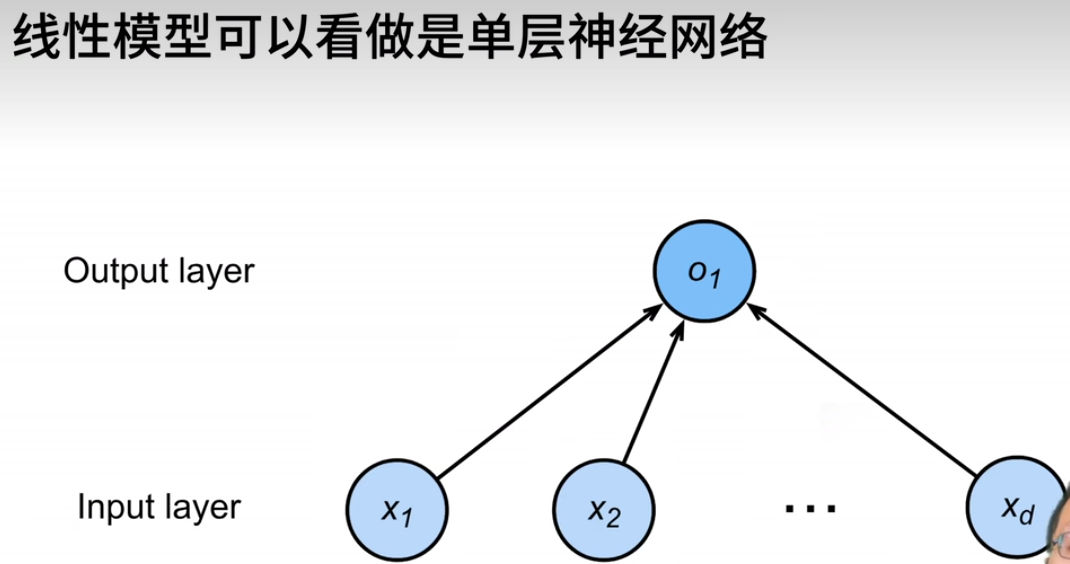
回归（regression）是能为一个或多个自变量与因变量之间关系建模的一类方法。在自然科学和社会科学领域，回归经常用来表示输入和输出之间的关系。机器学习的大部分问题都与预测有关，当我们想预测一个数值时，就会涉及到回归问题。常见的例子有：预测价格，预测需求等。

为了解释线性回归，我们举一个实际的例子：我们希望根据房屋的面积（平方英尺）和房龄（年）来估算房屋价格（美元）。为了开发一个能预测房价的模型，我们需要收集一个真实的数据集。这个数据集包括了房屋的销售价格、面积和房龄。在机器学习的术语中，该数据集称为训练数据集（trainingdataset）或训练集（training set）。每行数据（比如一次房屋交易相对应的数据）称为样本（sample），也可以称为数据点（data point）或数据样本（datainstance）。我们把试图预测的目标（比如预测房屋价格）称为标签（label）或目标（target）。预测所依据的自变量（面积和房龄）称为特征（feature）或协变量（covariate）。 通常，我们使用n来表示数据集中的样本数。对索引为i的样本，其输入表示为，其对应的标签是y(i)。

**1.模型**

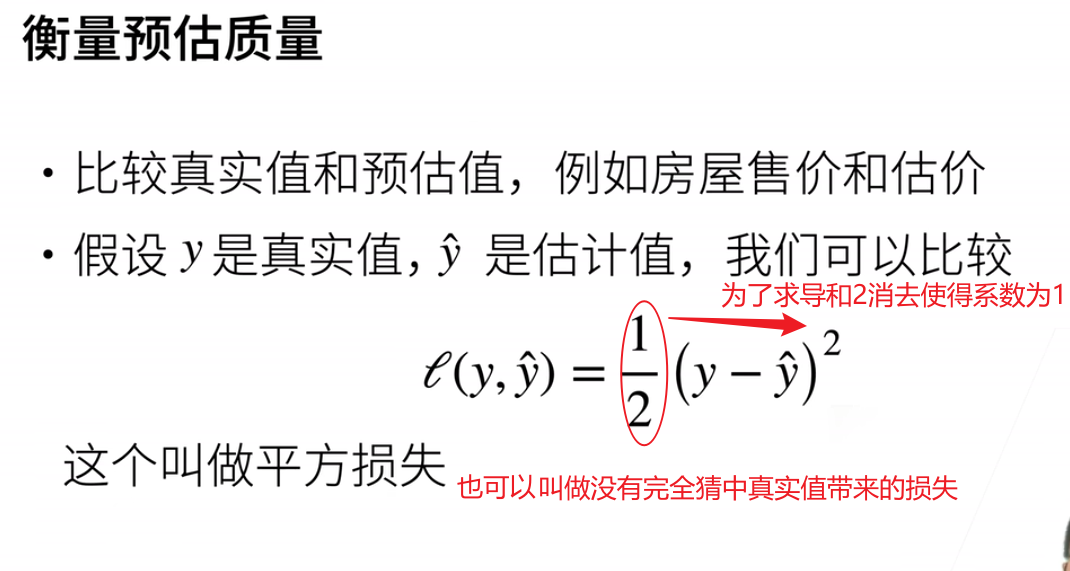


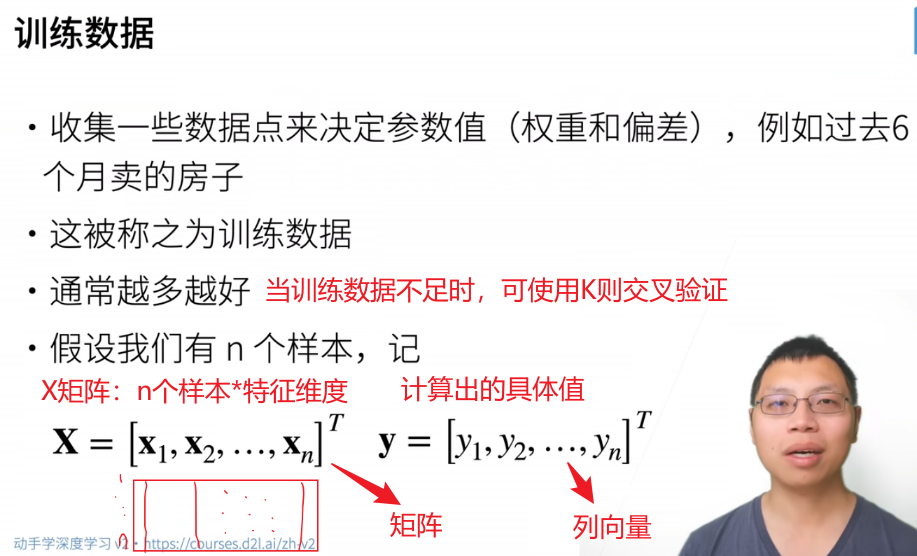
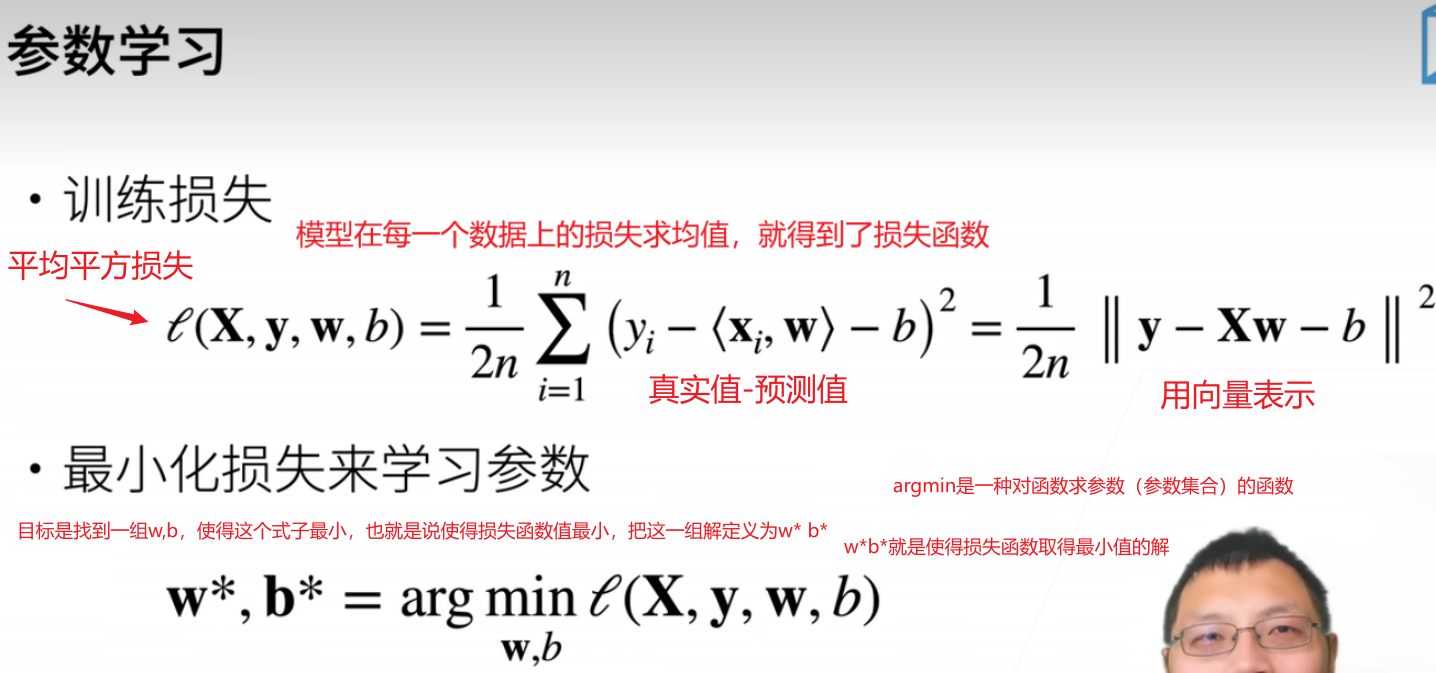
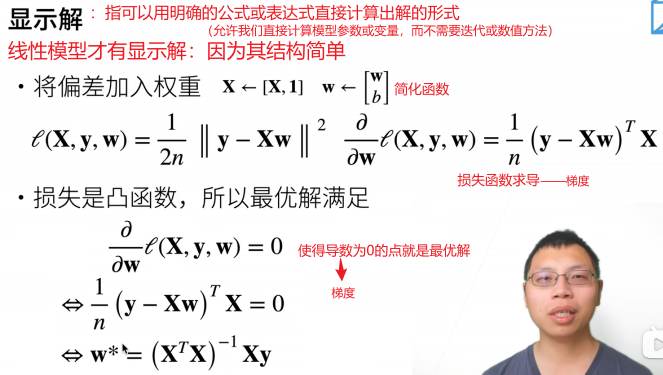






**2.预测**





线性模型具有显式解的原因主要在于其数学结构的简单性。具体来说，线性模型的形式通常可以表示为：

y=Xβ+ϵ

其中 *y*是响应变量，*X* 是设计矩阵，*β* 是待估参数，*ϵ* 是误差项。

1. **线性方程组**：线性模型可以转化为线性方程组，形式为*Aβ*=*b*，通过代数方法可以求解。
2. **最小二乘法**：在线性回归中，常用最小二乘法来估计参数，其目标是最小化残差平方和。通过求导并设为零，可以得到显式解：

*β*^​=(*XTX*)-1XTy

1. **矩阵运算**：线性模型中涉及的矩阵运算（如求逆）在数学上是可行的，只要 *XTX* 是可逆的。
2. **可解析性**：与非线性模型相比，线性模型的解析形式较为简单，便于推导和计算。

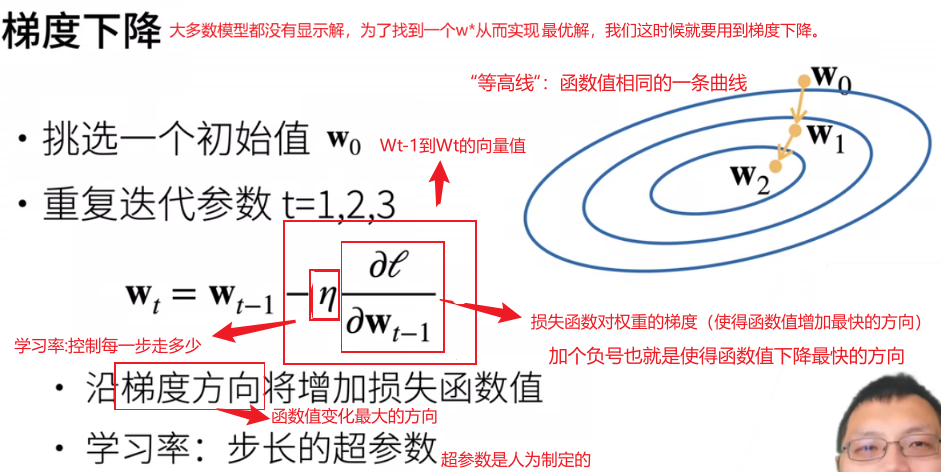
可以简单的理解为显示解就是函数，明确给出某个变量的值；隐式解就是方程，可能无法解出某个变量。

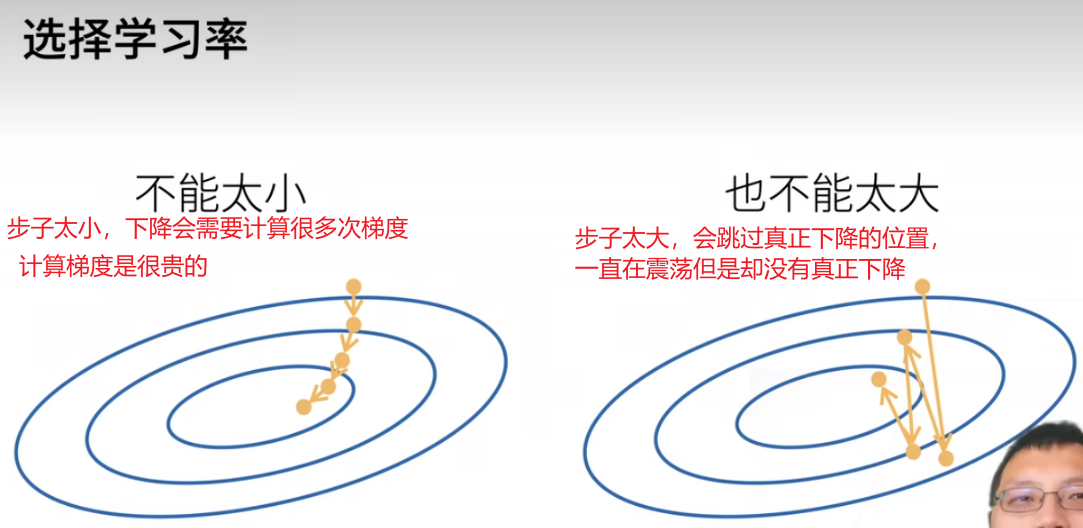
迭代是一种计算过程，其中通过重复应用某个算法或公式来逐步逼近问题的解。每次迭代都会使用上一次的结果作为下一次的输入，直到满足某个停止条件为止。

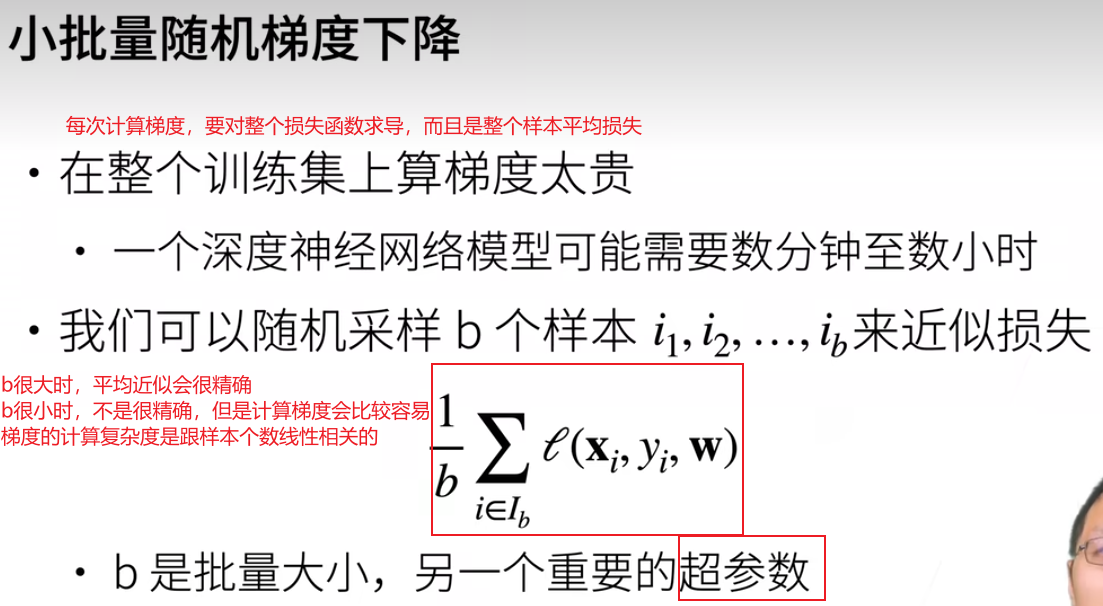
在数学和计算中，迭代常用于以下场景：

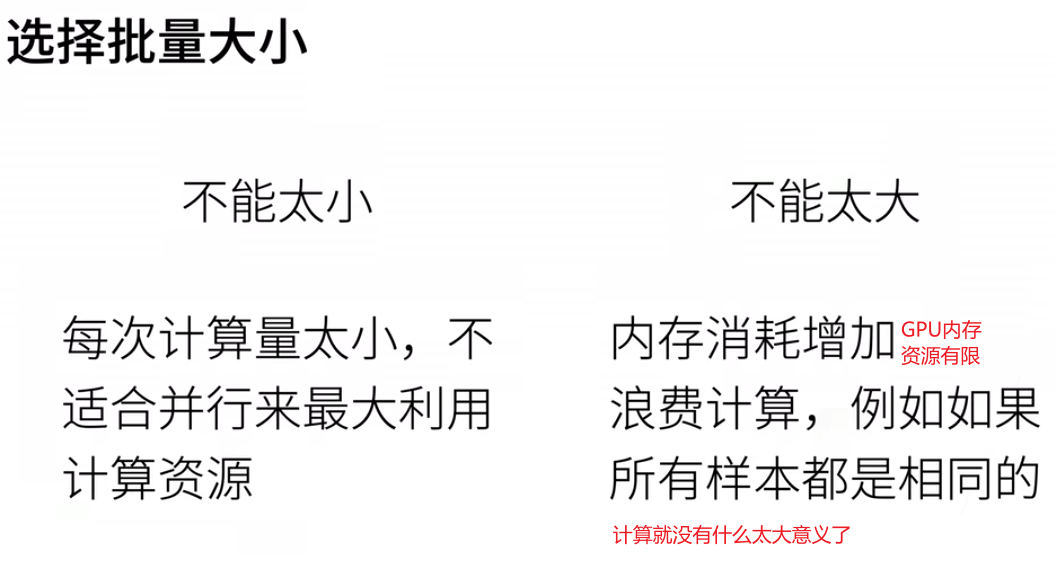
1. 数值方法：如求解方程、优化问题等。
2. 算法设计：如梯度下降法、牛顿法等。

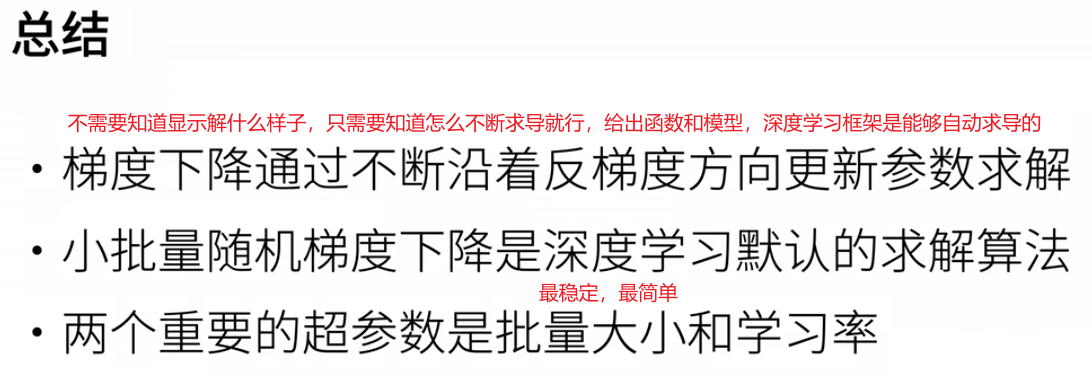
迭代的优点在于能够处理复杂问题，而不需要直接求解显式解。

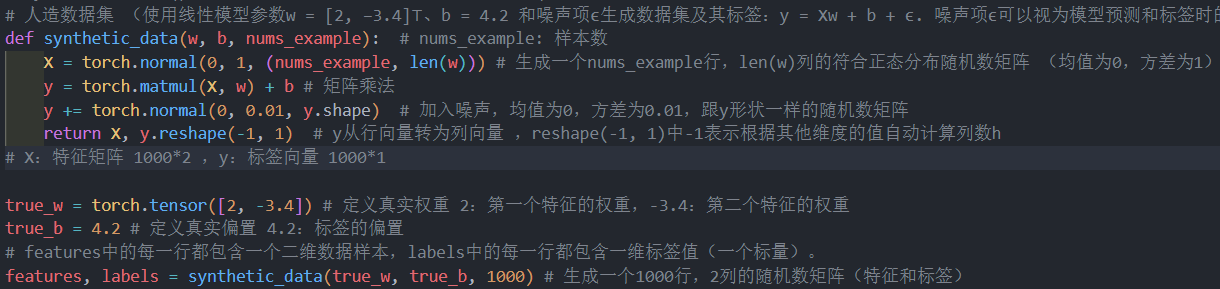
**3.基础优化算法**









**代码解释：**

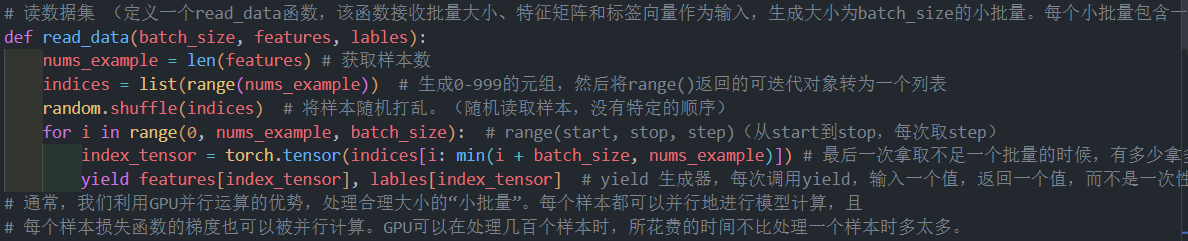
这里定义了个函数来构造一个数据集，然后给参数赋值实现了一个1000行（1000个样本）

**reshape**是 PyTorch 中用于改变张量形状的方法。reshape 方法可以重新定义张量的维度，而 -1 是一个特殊的值，表示根据其他维度的大小自动计算该维度的大小。

具体来说，reshape(-1, 1) 将张量重新调整为形状为 (n, 1)，其中 n 是张量中元素的总数。-1 表示这个维度的大小由其他维度的大小自动计算得出。

在机器学习和统计学中，噪声通常指的是数据中的随机误差或不确定性。噪声项 ϵ 在这个上下文中表示的是模型预测和实际标签之间的潜在观测误差。具体来说，噪声项 ϵ 是一个随机变量，它模拟了数据生成过程中的随机性或不确定性。

在生成人造数据集时，我们使用线性模型 y = Xw + b 来生成标签 y，然后通过在 y 上加上一个服从正态分布的噪声项 ϵ 来模拟实际观测中的不确定性。这个噪声项 ϵ 的均值是0，方差是0.01，形状与 y 相同。

加入噪声项的目的是为了使生成的数据集更加接近真实世界中的数据，因为真实数据往往存在随机误差。通过这种方式，我们可以测试和验证机器学习模型在处理包含噪声的数据时的性能和鲁棒性。

从特征矩阵和标签向量中读取数据，随机打乱数据，并生成指定批量大小的小批量数据，最后一次不满批量的时候直接把剩下的作为最后一批量。

**yield**：生成器函数，每次调用时返回一个包含特征和标签的小批量数据。

GPU可以并行处理数据，以此加速训练进程

y是人造数据集中的标签向量，是根据随机特征矩阵X与指定权重true\_w还有指定偏置向量b以及随机噪声经过矩阵计算得到的目标值。

y\_hat是每一批次中的特征矩阵X与随机初始化的权重w还有偏置为0的向量b经过矩阵计算得到的标签向量

用它们之间所产生的差值，也就是损失，来衡量这个权重w和偏置b拟合数据集的程度。

不断通过优化算法sgd来更新这两个参数，来优化模型，使得模型在训练数据上的表现越来越好。

**torch.no\_grad()** 是一个上下文管理器，用于临时禁用梯度计算。在更新参数时，不需要计算梯度，因此使用 torch.no\_grad() 可以加速计算并节省显存。

param.grad 是参数的梯度，表示参数对损失函数的偏导数，用于指示参数更新的方向。

更新公式 param -= lr \* param.grad / batch\_size 是梯度下降算法的核心，用于更新参数值。

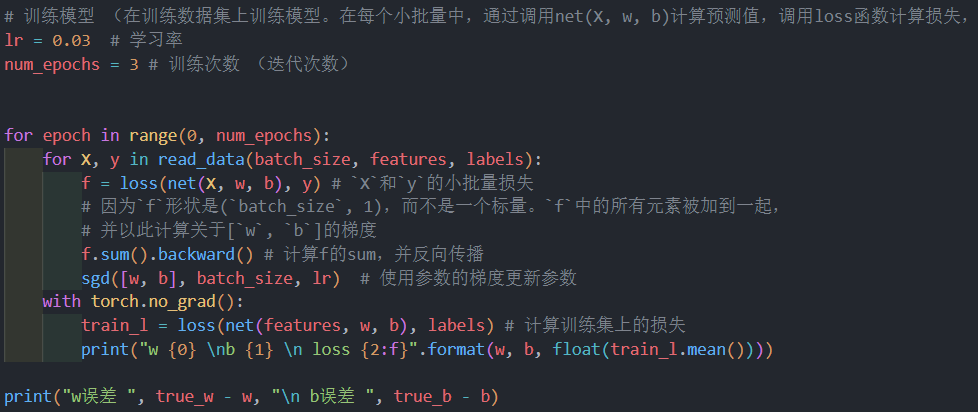
param -= 表示将参数值减去更新量，即向梯度的反方向移动。

lr \* param.grad 表示根据梯度方向和步长计算更新量。

/ batch\_size 是为了求平均梯度，而不是求总和。在批量梯度下降中，每个参数的梯度是通过整个批次的数据计算得到的。为了得到每个参数的平均梯度，需要将梯度除以批次大小。

**grad** 函数是 PyTorch 中用于获取张量（Tensor）的梯度属性的方法。在训练神经网络时，梯度用于计算损失函数相对于模型参数的导数，从而指导参数的更新。

grad.zero\_() 是为了梯度清零，这样下一次计算梯度的时候就不会跟上一次的梯度相关了

这段代码大致过程是首先定义学习率和训练次数，然后按批次进行前向传播计算预测值，然后计算损失，接着进行反向传播计算梯度，最后使用优化算法更新参数。

**Sum()**：损失函数的输出是一个标量值，表示模型的预测结果与真实值之间的差异。计算损失函数的梯度时，需要一个标量值作为输入。需要先对张量进行求和操作，得到一个标量值，然后再对这个标量值调用.backward()方法进行反向传播，计算梯度。

**backward()**：这个方法用于执行反向传播。在PyTorch中，当你对一个包含梯度信息的张量调用.backward()方法时，PyTorch会自动计算该张量对图中所有可训练参数的梯度。这个操作会沿着计算图从输出节点向输入节点传播，计算每个参数的梯度。

**mean()** ： Python 中用于计算一组数值的平均值的函数