

# 智能之门

神经网络和深度学习入门

(基于Python的实现)



## STEP 2 线性回归

# 第 4 章

## 单入单出的单层神经网络

### 单变量线性回归

4.1 单变量线性回归问题

4.2 最小二乘法

4.3 梯度下降法

4.4 神经网络法

4.5 多样本计算

4.6 梯度下降的三种形式

用线性回归作为学习神经网络的起点，是一个非常好的选择，因为线性回归问题本身比较容易理解，在它的基础上，逐步的增加一些新的知识点，会形成一条比较平缓的学习曲线，或者说是迈向神经网络的第一个小台阶。

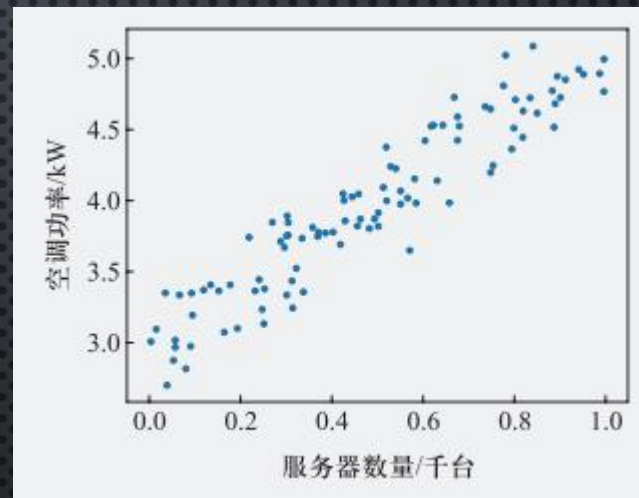
单层的神经网络，其实就是一个神经元，可以完成一些线性的工作，比如拟合一条直线，这用一个神经元就可以实现。当这个神经元只接收一个输入时，就是单变量线性回归，可以在二维平面上用可视化方法理解。



## 4.1 单变量线性回归问题

在互联网建设初期，各大运营商需要解决的问题就是保证服务器所在的机房的温度常年保持在23摄氏度左右。在一个新建的机房里，如果计划部署346台服务器，我们如何配置空调的最大功率？通过一些统计数据（称为样本数据），可以得到相应图表。

样本序号	服务器数量 $X$ /千台	空调功率 $Y$ /kW
1	0.928	4.824
2	0.469	2.950
3	0.855	4.643
...	...	...



通过对上图的观察，我们可以判断它属于一个线性回归问题，而且是最简单的一元线性回归。于是，我们把热力学计算的问题转换成为了一个统计问题，因为实在是不能精确地计算出每块电路板或每台机器到底能产生多少热量。



## 4.1 单变量线性回归问题

### ➤ 一元线性回归模型

- 回归分析是一种数学模型。当因变量和自变量为线性关系时，它是一种特殊的线性模型。最简单的情形是一元线性回归，由大体上有线性关系的一个自变量和一个因变量组成：

$$Y = a + bX + \varepsilon$$

- 上述模型中 $X$ 是自变量， $Y$ 是因变量， $\varepsilon$ 是随机误差， $a, b$ 是需要学习的参数。

通过对数据的观察，可以大致认为它符合线性回归模型的条件，于是列出了以上公式。若不考虑随机误差，线性回归的主要任务就是找到合适的 $a, b$ 。



## 4.1 单变量线性回归问题

### ➤ 线性回归模型的相关概念和性质

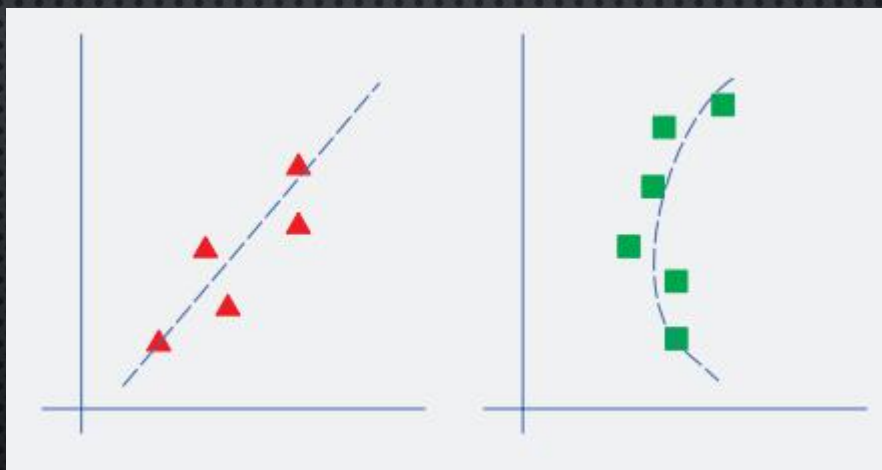
- 通常假定随机误差的均值为0，方差为 $\sigma^2 > 0$ 。
- 若进一步假定随机误差遵从正态分布，就成为正态线性模型。
- 一般地，若有 $k$ 个自变量和1个因变量，则因变量的值分为两部分：一部分由自变量影响，即表示为它的函数，函数形式已知且含有未知参数；另一部分由其他的未考虑因素和随机性影响，即随机误差。
- 当函数为参数未知的线性函数时，称为线性回归分析模型。
- 当函数为参数未知的非线性函数时，称为非线性回归分析模型。
- 当自变量个数大于1时称为多元线性回归。
- 当因变量个数大于1时称为多重线性回归。



## 4.1 单变量线性回归问题

### ➤ 线性回归和非线性回归的区别

- 如图所示，左侧为线性模型，可以看到直线穿过了一组三角形所形成的区域的中心线，并不要求这条直线穿过每一个三角形。右侧为非线性模型，一条曲线穿过了一组矩形所形成的区域的中心线。





## 4.1 单变量线性回归问题

### ➤ 公式表示

- 本系列中的线性回归模型的矩阵表示形式为

$$Y = XW + B$$

- 上述公式中， $X$ 为样本数据矩阵（每行代表样本中单个个体，每列代表各个不同特征）， $W$ 为系数列向量， $B$ 为偏置向量。我们需要学习获得合适的 $W$ 和 $B$ 。



## 4.2 最小二乘法

### ➤ 最小二乘法

- 最小二乘法，也叫做最小平方法（LEAST SQUARE METHOD），它通过最小化误差的平方和寻找数据的最佳函数匹配。利用最小二乘法可以简便地求得未知的数据，并使得这些求得的数据与实际数据之间误差的平方和为最小。最小二乘法还可用于曲线拟合。其他一些优化问题也可通过最小化能量或最小二乘法来表达。

高斯使用的最小二乘法的方法发表于1809年他的著作《天体运动论》中。法国科学家勒让德于1806年独立发明“最小二乘法”，但因不为世人所知而默默无闻。勒让德曾与高斯为谁最早创立最小二乘法原理发生争执。1829年，高斯提供了最小二乘法的优化效果强于其他方法的证明，因此被称为高斯-马尔可夫定理。



## 4.2 最小二乘法

### ➤ 公式原理

- 线性回归试图学得  $z(x_i) = w \cdot x_i + b$ ，使得  $z(x_i) \cong y_i, \forall i$ 。其中  $x_i$  为样本特征值， $y_i$  为样本标签值， $z(x_i)$  为模型预测值。
- 为了学习  $w, b$ ，常常使用均方差损失函数（MSE）作为最小化函数（省略了系数），即试图找到一条直线，使所有样本到直线上的残差的平方和最小：

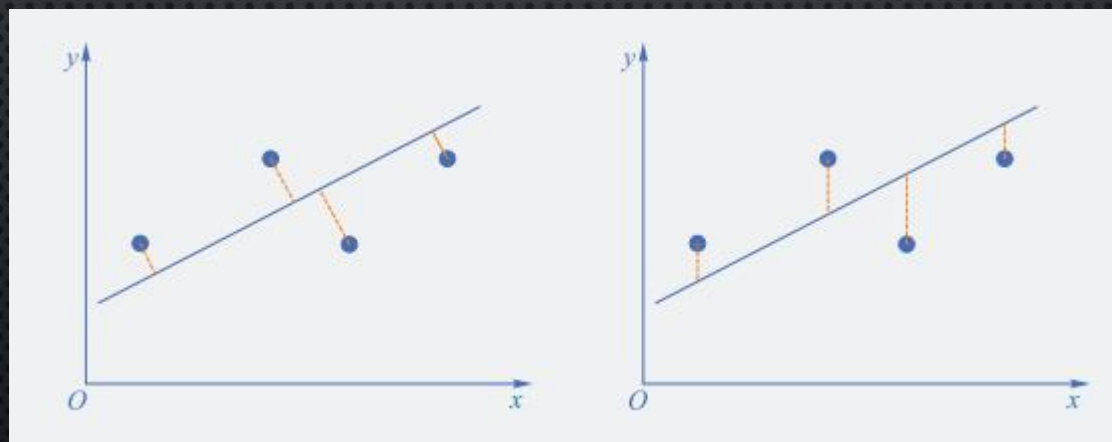
$$J = \frac{1}{2m} \sum_{i=1}^m (z(x_i) - y_i)^2 = \frac{1}{2m} \sum_{i=1}^m (y_i - wx_i - b)^2$$



## 4.2 最小二乘法

圆形点是样本点，直线是当前的拟合结果。如左图所示，我们是要计算样本点到直线的垂直距离，需要再根据直线的斜率来求垂足然后再计算距离，这样计算起来很慢；但实际上，在工程上我们通常使用的是右图的方式，即样本点到直线的竖直距离，因为这样计算很方便，用一个减法就可以了。

因为图中的几个点不在一条直线上，所以不存在一条直线能同时穿过它们。所以，我们只能想办法让总体误差最小，就意味着整体偏差最小，那么最终的那条直线就是要求的结果。





## 4.2 最小二乘法

### ➤ 数学推导

- 欲使损失函数最小化，需置 $J$ 对 $w, b$ 的一阶偏导数为零，即：

$$\frac{\partial J}{\partial w} = \frac{1}{m} \sum_{i=1}^m (y_i - wx_i - b)(-x_i) = 0, \quad \frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^m (y_i - wx_i - b) = 0$$

- 从第二个式子可得

$$b = \frac{1}{m} \sum_{i=1}^m (y_i - wx_i) = \bar{y} - w\bar{x}$$

- 将此式代回第一个式子，得到

$$0 = \frac{1}{m} \sum_{i=1}^m (y_i - wx_i - \bar{y} + w\bar{x})x_i = \frac{1}{m} \sum_{i=1}^m x_i(y_i - \bar{y}) - \frac{w}{m} \sum_{i=1}^m x_i(x_i - \bar{x})$$



## 4.2 最小二乘法

- 从而可以解得：

$$w = \frac{\sum_{i=1}^m x_i (y_i - \bar{y})}{\sum_{i=1}^m x_i (x_i - \bar{x})} = \frac{m \sum_{i=1}^m x_i y_i - \sum_{i=1}^m x_i \sum_{i=1}^m y_i}{m \sum_{i=1}^m x_i^2 - \left(\sum_{i=1}^m x_i\right)^2}$$

- 事实上，上述 $w$ 的表达式还可写成以下形式：

$$w = \frac{\sum_{i=1}^m y_i (x_i - \bar{x})}{\sum_{i=1}^m x_i^2 - \frac{1}{m} \left(\sum_{i=1}^m x_i\right)^2}$$

- 其正确性源于以下恒等式（第二项存在对称性）：

$$\sum_{i=1}^m x_i \bar{y} = \frac{1}{m} \sum_{i=1}^m x_i \sum_{i=1}^m y_i = \sum_{i=1}^m y_i \bar{x}$$



## 4.3 梯度下降法

### ➤ 数学原理

- 预设函数： $z_i = x_i \cdot w + b$
- 损失函数：均方差函数。

梯度下降法与最小二乘法相比，二者的模型及损失函数是相同的，都是一个线性模型加均方差损失函数，模型用于拟合，损失函数用于评估效果。区别在于，最小二乘法从损失函数求导，直接求得数学解析解，而梯度下降以及后面的神经网络，都是利用导数传递误差，再通过迭代方式一步一步（用近似解）逼近真实解。



## 4.3 梯度下降法

### ➤ 梯度下降

- 由梯度公式

$$\frac{\partial loss_i}{\partial z_i} = z_i - y_i$$

- 通过链式法则可计算得

$$\frac{\partial loss_i}{\partial w} = \frac{\partial loss_i}{\partial z_i} \cdot \frac{\partial z_i}{\partial w} = (z_i - y_i)x_i, \quad \frac{\partial loss_i}{\partial b} = \frac{\partial loss_i}{\partial z_i} \cdot \frac{\partial z_i}{\partial b} = z_i - y_i$$

- 参数更新

$$w \leftarrow w - \eta \frac{\partial loss_i}{\partial w}, \quad b \leftarrow b - \eta \frac{\partial loss_i}{\partial b}$$

- 对样本*i*作循环更新，成为全样本梯度下降的一步反向传播；更新至收敛条件满足即可。



## 4.4 神经网络法

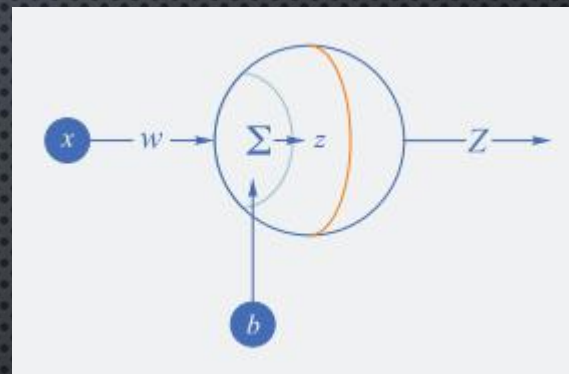
➤ 在梯度下降法中，我们简单讲述了一下神经网络做线性拟合的原理，即：

- 初始化权重值
- 根据权重值放出一个解
- 根据均方差函数求误差
- 误差反向传播给线性计算部分以调整权重值
- 是否满足终止条件？不满足的话跳回第2步



## 4.4 神经网络法

建立神经网络，尝试用一个最简单的单层单点神经元，如右图。下面，我们用这个最简单的线性回归的例子，来说明神经网络中最重要的反向传播和梯度下降的概念、过程以及代码实现。



### ➤ 输入层

- 此神经元在输入层只接受一个输入特征，经过参数  $w$ ,  $b$  的计算后，直接输出结果。这样一个简单的“网络”，只能解决简单的一元线性回归问题，而且由于是线性的，不需要定义激活函数，大大简化了程序，而且便于大家循序渐进地理解各种知识点。

### ➤ 权重 $w$ 和 $b$

- 因为是一元线性问题，所以  $w$  和  $b$  都是一个标量。



## 4.4 神经网络法

### ➤ 输出层

- 输出层为1个神经元，线性预测公式为： $z_i = x_i \cdot w + b$

### ➤ 损失函数

- 损失函数使用均方差函数。

### ➤ 反向传播

- 由于单层无激活函数的神经网络只能处理线性回归问题，因而梯度下降与反向传播的相关计算与上节完全相同。其实神经网络法和梯度下降法在本质上是一样的，只是神经网络法使用了一个崭新的编程模型，即以神经元为中心的代码结构设计，便于以后的功能扩充。

对于初学神经网络的人来说，可视化的训练过程及结果，可以极大地帮助理解神经网络的原理，PYTHON 的 MATPLOTLIB 库提供了非常丰富的绘图功能。



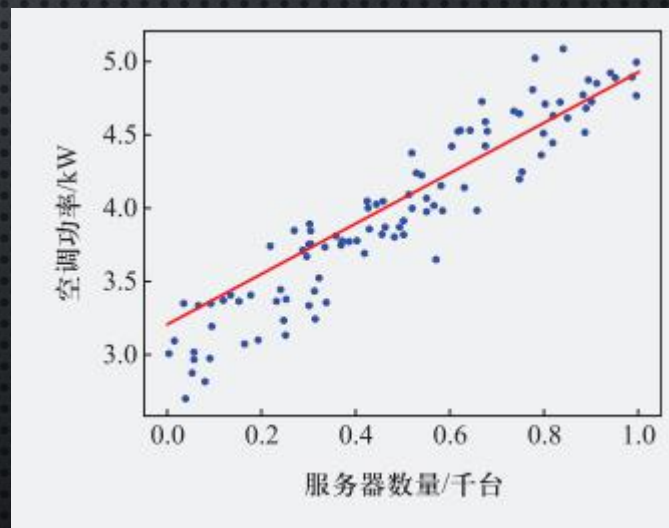
## 4.4 神经网络法

### ➤ 工作原理

- 首先假设这个问题是个线性问题，因而有了公式 $z = xw + b$ 。本节用神经元的编程模型把梯度下降法包装了一下，这样就进入了神经网络的世界，从而可以有成熟的方法论可以解决更复杂的问题，比如多个神经元协同工作、多层神经网络的协同工作等等。

神经网络的任务就是找到一根直线（注意我们首先假设这是线性问题），让该直线穿过样本点阵，并且所有样本点到该直线的距离的平方的和最小。可以想象成每一个样本点都有一根橡皮筋连接到直线上，连接点距离该样本点最近，所有的橡皮筋形成一个合力，不断地调整该直线的位置。

该合力具备两种调节方式：如果上方的拉力大一些，直线就会向上平移一些，这相当于调节 $b$ 值；如果侧方的拉力大一些，直线就会向侧方旋转一些，这相当于调节 $w$ 值。直到该直线处于平衡位置时，也就是线性拟合的最佳位置了。





## 4.5 多样本计算

### ➤ 单样本计算存在一些缺点：

- 前后两个相邻的样本，很有可能会对反向传播产生相反的作用而互相抵消。
- 在样本数据量大时，逐个计算会花费很长的时间。

多样本计算涉及矩阵运算，而所有的深度学习框架都对矩阵运算做了优化，会大幅提升运算速度。



## 4.5 多样本计算

### ➤ 前向计算

- 多个样本同时计算，可用矩阵表示： $Z_{m \times 1} = X_{m \times p}W_{p \times 1} + b_{m \times 1}$

### ➤ 梯度计算和反向传播

- 损失函数依然为均方差函数，计算梯度公式为

$$\frac{\partial J}{\partial w} = \sum_{i=1}^m \frac{\partial J}{\partial z_i} \cdot \frac{\partial z_i}{\partial w} = \frac{1}{m} X^T (Z - Y)$$
$$\frac{\partial J}{\partial b} = \sum_{i=1}^m \frac{\partial J}{\partial z_i} \cdot \frac{\partial z_i}{\partial b} = \frac{1}{m} (Z - Y)$$

- 使用PYTHON中NUMPY矩阵运算进行反向传播即可。



## 4.6 梯度下降的三种形式

对开头的例子，使用三种方法求得的 $w, b$ 如下表所示，原始值为 $w = 2, b = 3$ 。

方法	$w$	$b$
最小二乘法	2.056 827	2.965 434
梯度下降法	1.71 629 006	3.19 684 087
神经网络法	1.71 629 006	3.19 684 087

最小二乘法可以得到数学解析解，结果是可信的。但是使用梯度下降和神经网络两种方法，得到的值准确度很低。二者本质相同，只是梯度下降法没有使用神经元模型。

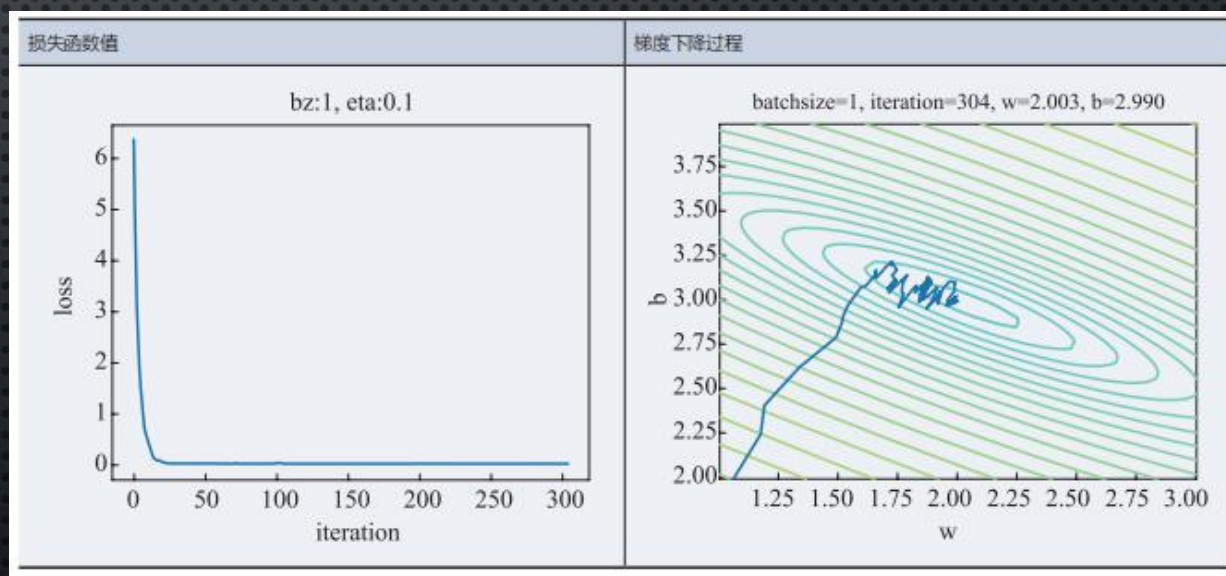
难道神经网络法有什么问题吗？接下来需要研究一下如何调整神经网络的训练过程，先从最简单的梯度下降的三种形式说起。



## 4.6 梯度下降的三种形式

### ➤ 梯度下降的三种形式

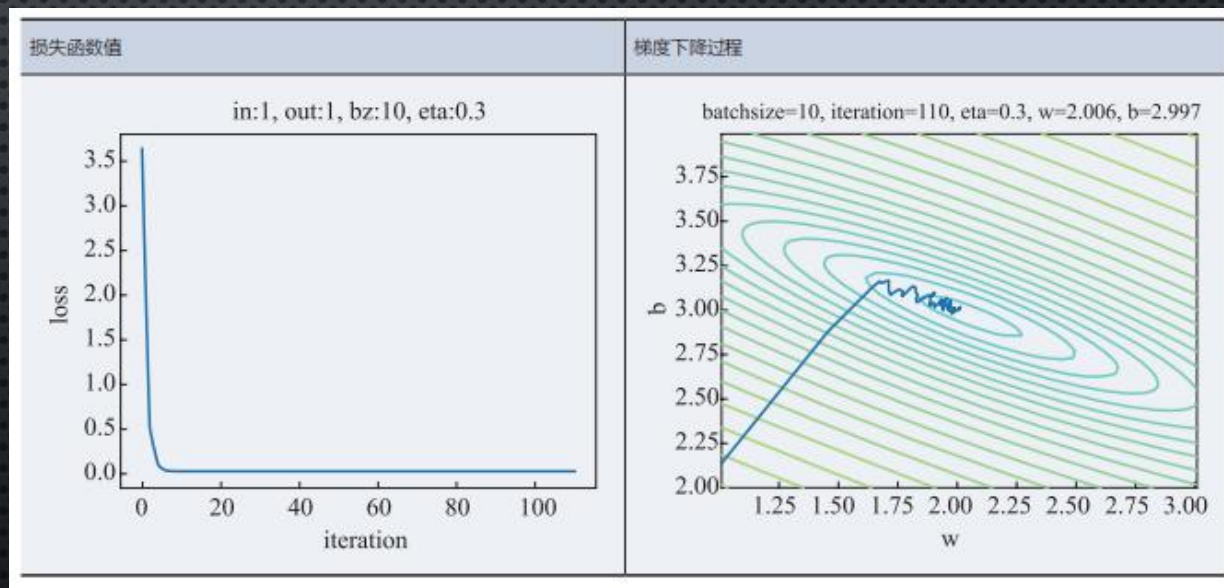
- 单样本随机梯度下降  
(STOCHASTIC GRADIENT DESCENT)：每次使用一个样本数据进行一次训练，更新一次梯度，重复以上过程。
  - ✓ 训练开始时损失值下降很快，随机性大，找到最优解的可能性大。
  - ✓ 受单个样本的影响大，损失函数值波动大，到后期徘徊不前，在最优解附近震荡；且不能并行计算。





## 4.6 梯度下降的三种形式

- 小批量样本梯度下降  
(MINI-BATCH GRADIENT DESCENT)：选择一小部分样本进行训练，更新一次梯度，然后再选取另外一小部分样本进行训练，再更新一次梯度。
  - ✓ 不受单样本噪声影响，训练速度较快。
  - ✓ 批大小的选择很关键，会影响训练结果。





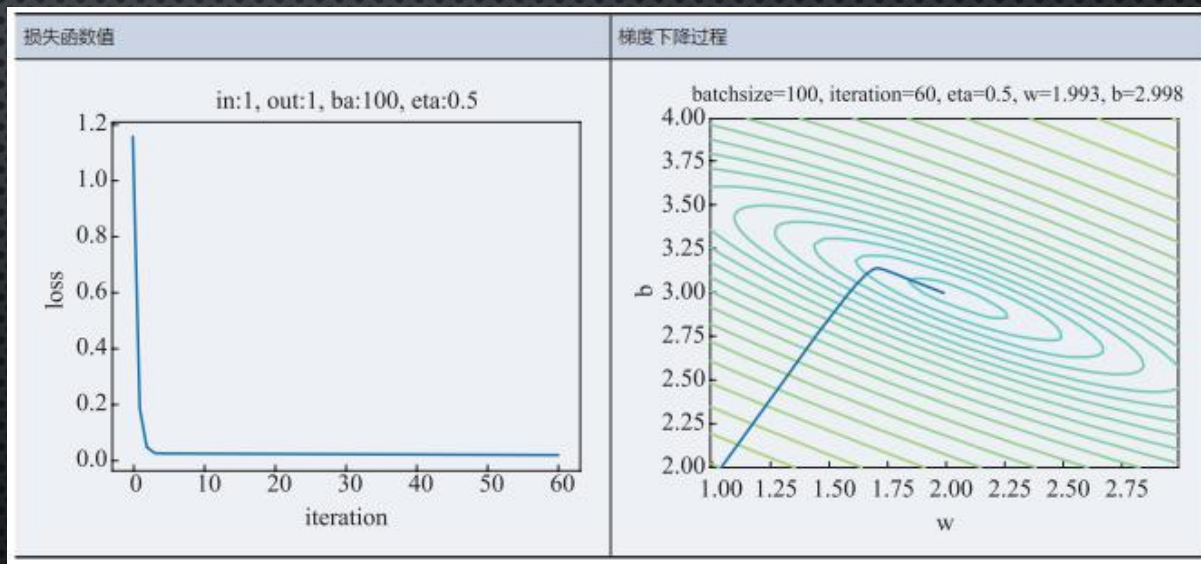
## 4.6 梯度下降的三种形式

- 小批量的大小通常由以下几个因素决定：
  - ✓ 更大的批量会计算更精确的梯度，但是回报却是小于线性的。
  - ✓ 极小批量通常难以充分利用多核架构。这决定了最小批量的数值，低于这个值的小批量处理不会减少计算时间。
  - ✓ 如果批量处理中的所有样本可以并行地处理，那么内存消耗和批量大小成正比。对于多硬件设施，这是批量大小的限制因素。
  - ✓ 某些硬件上使用特定大小的数组时，运行时间会更少，尤其是GPU，通常使用2的幂数作为批量大小可以更快，如32、64、128、256，大模型时尝试用16。
  - ✓ 可能是由于小批量在学习过程中加入了噪声，会带来一些正则化的效果。泛化误差通常在批量大小为1时最好。因为梯度估计的高方差，小批量使用较小的学习率，以保持稳定性，但是降低学习率会使迭代次数增加。
- 在实际工程中，我们通常使用小批量梯度下降形式。



## 4.6 梯度下降的三种形式

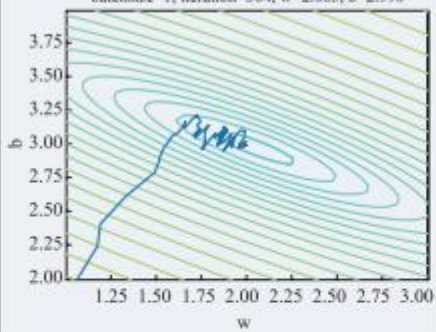
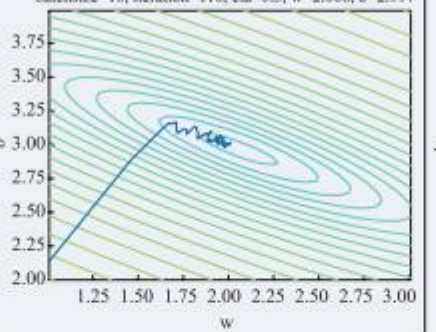
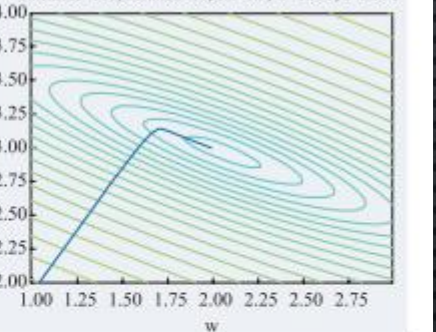
- 全批量样本梯度下降 (FULL-BATCH GRADIENT DESCENT)：每次使用全部数据集进行一次训练，更新一次梯度，重复以上过程。
  - ✓ 受单个样本的影响最小，一次计算全体样本速度快，损失函数值没有波动，到达最优点平稳，方便并行计算。
  - ✓ 数据量较大时不能实现（内存限制），训练过程变慢。初始值不同，可能导致获得局部最优解，并非全局最优解。





## 4.6 梯度下降的三种形式

### ➤ 三种方法的比较

方法	单样本	小批量	全批量
结果			
梯度下降过程图解	<p>batchsize=1, iteration=304, w=2.003, b=2.990</p> 	<p>batchsize=10, iteration=110, eta=0.3, w=2.006, b=2.997</p> 	<p>batchsize=100, iteration=60, eta=0.5, w=1.993, b=2.998</p> 
批大小	1	10	100
学习率	0.1	0.3	0.5
迭代次数	304	110	60
epoch	3	10	60
结果	w=2.003, b=2.990	w=2.006, b=2.997	w=1.993, b=2.998

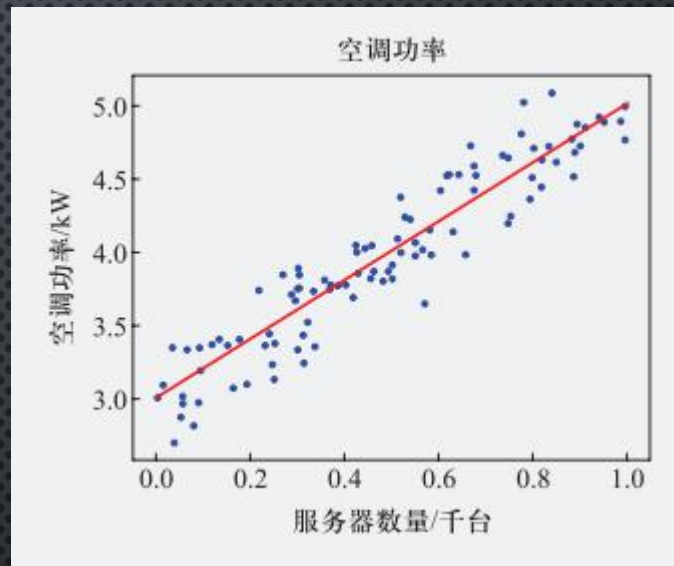


## 4.6 梯度下降的三种形式

- 相关概念

- ✓ BATCH SIZE: 批大小, 一次训练的样本数量。
- ✓ ITERATION: 迭代, 一次正向传播和一次反向传播。
- ✓ EPOCH: 所有样本被使用了一次, 叫作一个 EPOCH。

右图为较理想的拟合效果图。



- QUESTION

- ◆ 用纸笔推算一下矩阵运算的维度, 假设:  $X \in R^{4 \times 2}, W \in R^{2 \times 3}, B \in R^{1 \times 3}$ 。



THE END

谢谢！