task1

导论

机器学习就是让机器具备找一个函数的能力。

机器学习的类别

1. 回归（regression）
    1. 要找的函数的输出是一个数值，一个标量
2. 分类（classification）
    1. 要让机器做选择题，提供选项（class）
3. 结构化学习（structured learning）
    1. 不是做选择题或者输出数值，是产生一个有结构的物体，比如说图画和文章

案例学习

> 以视频的点击次数预测为例介绍下机器学习的运作过程。假设有人想要通过视频平台赚钱，他会在意频道有没有流量，这样他才会知道他的获利。假设后台可以看到很多相关的信息，比如：每天点赞的人数、订阅人数、观看次数。根据一个频道过往所有的信息可以预测明天的观看次数。找一个函数，该函数的输入是后台的信息，输出是隔天这个频道会有的总观看的次数.

分为三个步骤

步骤一：写出一个带有未知参数的函数 f

Info

$$y = b + wx_1$$

1. y是需要预测的数据，比如说是明天的观看人数，x是今天的观看人数
2. b 跟 w 是未知的。带有未知的参数（parameter）的函数称为模型（model）。模型在机器学习里面，就是一个带有未知的参数的函数，特征（feature） x1 是这个函数里面已知的
3. w 称为权重（weight），b 称为偏置（bias）。

步骤二：定义损失（loss），损失也是一个函数，这个函数的输入是 parameter of model

Info

函数 L(b,w)

真实值成为标签（label），真实值为y。估测值为$\hat{y}$

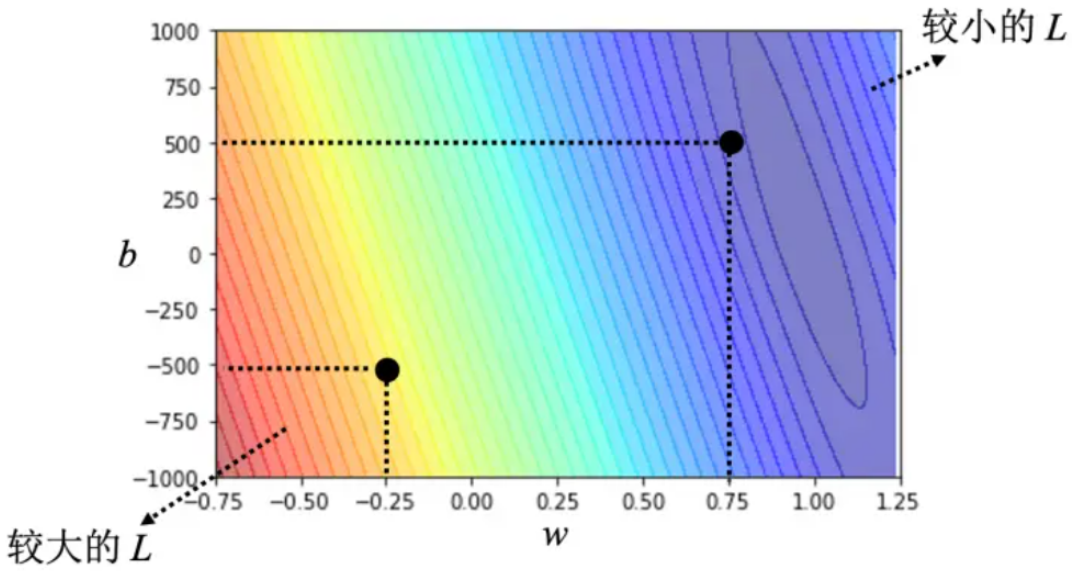
计算真实值y和估测值$\hat{y}$的差值e，有不同的算法比如说

1. 平均绝对误差（Mean Absolute Error，MAE）：$e = |\hat{y} - y|$
2. 均方误差（Mean SquaredError，MSE）：$e = (\hat{y} - y)^2$
3. 有一些任务中 y 和 yˆ 都是概率分布，这个时候可能会选择交叉熵（cross entropy）

然后可以把每一组的误差e求平均获得误差 L

$$L = \frac{1}{n}\sum_{n} e_n$$

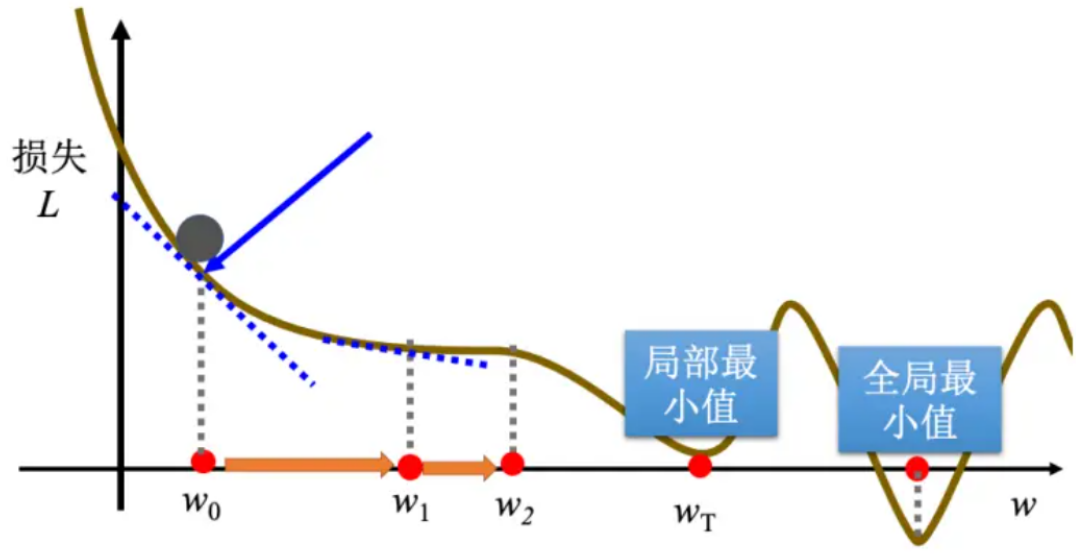
可以调整不同的 w 和不同的 b，求取各种w 和各种 b，组合起来以后，我们可以为不同的 w 跟 b 的组合，都去计算它的损失，就可以画出如图所示的等高线图。在这个等高线图上面，越偏红色系，代表计算出来的损失越大，就代表这一组 w 跟 b 越差。如果越偏蓝色系，就代表损失越小，就代表这一组 w 跟 b 越好，拿这一组 w 跟 b，放到函数里面，预测会越精准。



步骤三：解一个最优化的问题。

找一个 w 跟 b，把未知的参数找一个数值出来，看代哪一个数值进去可以让损失 L 的值最小，就是要找的 w 跟 b，这个可以让损失最小的 w 跟 b 称为 b^* 和 w^* 代表它们是最好的一组 w 跟 b，可以让损失的值最小。梯度下降（gradient descent）是经常会使用优化的方法。这边只有一个参数，所以这个误差表面是 1 维的。

Info



首先要随机选取一个初始的点 w^0。接下来计算

$$\frac{\partial L}{\partial w}\Big|_{w=w^0}$$

，在 w 等于 w^0 的时候，参数 w 对损失的微分。 计算在这一个点，在 w0 这个位置的误差表面的切线斜率，也就是这一条蓝色的虚线。
如果斜率为负，就朝右边调整，反之也成立。调整的每一步的步伐大小取决于两件事

1. 第一件事情是这个地方的斜率，斜率大步伐就跨大一点，斜率小步伐就跨小一点。
2. 另外，学习率（learning rate）η 也会影响步伐大小。学习率是自己设定的，如果 η 设大一点，每次参数更新就会量大，学习可能就比较快。如果 η 设小一点，参数更新就很慢，每次只会改变一点点参数的数值。这种在做机器学习，需要自己设定，不是机器自己找出来的，称为超参数（hyperparameter）。

> Faq
>
> Q: 为什么损失可以是负的?
> A: 损失函数是自己定义的，在刚才定义里面，损失就是估测的值跟正确的值的绝对值。如果根据刚才损失的定义，它不可能是负的。但是损失函数是自己决定的，比如设置一个损失函数为绝对值再减 100，其可能就有负的。这个曲线并不是一个真实的损失，并不是一个真实任务的误差表面。因此这个损失的曲线可以是任何形状。

把 w^0往右移一步，新的位置为w^1，这一步的步伐是 η 乘上微分的结果，即：

$$w^1 \leftarrow w^0 - \eta \frac{\partial L}{\partial w}|_{w=w^0}$$

不断移动w 的位置，最后会停下来。往往有两种情况会停下来。

1. 第一种情况是一开始会设定说，在调整参数的时候，在计算微分的时候，最多计算几次。上限可能会设为 100 万次，参数更新 100 万次后，就不再更新了，更新次数也是一个超参数。
2. 还有另外一种理想上的，停下来的可能是，当不断调整参数，调整到一个地方，它的微分的值就是这一项，算出来正好是 0 的时候，如果这一项正好算出来是 0，0 乘上学习率 η 还是 0，所以参数就不会再移动位置。假设是这个理想的情况，把 w^0 更新到 w^1，再更新到 w^2，最后更新到 w^T有点卡，w^T 卡住了，也就是算出来这个微分的值是 0 了，参数的位置就不会再更新。

> Warning
>
> 梯度下降有一个很大的问题，没有找到真正最好的解，没有找到可以让损失最小的 w。在图所示的例子里面，把 w 设定在最右侧红点附近这个地方可以让损失最小。但如果在梯度下降中，w^0 是随机初始的位置，也很有可能走到 w^T 这里，训练就停住了，无法再移动 w 的位置。右侧红点这个位置是真的可以让损失最小的地方，称为全局最小值（global minima），而 w^T 这个地方称为局部最小值（local minima），其左右两边都比这个地方的损失还要高一点，但是它不是整个误差表面上面的最低点。
> 所以常常可能会听到有人讲到梯度下降不是个好方法，这个方法会有局部最小值的问题，无法真的找到全局最小值。事实上局部最小值是***一个假问题***，在做梯度下降的时候，真正面对的难题不是局部最小值。有两个参数的情况下使用梯度下降，其实跟刚才一个参数没有什么不同。如果一个参数没有问题的话，可以很快的推广到两个参数。假设有两个参数，随机初始值为 w^0, b^0。要计算 w, b 跟损失的微分，计算在 w = w^0 的位置，b = b^0的位置，要计算 w 对 L 的微分，计算 b 对 L 的微分
>
> $$\frac{\partial L}{\partial w}|_{w=w^0,b=b^0}$$
>
> $$\frac{\partial L}{\partial b}|_{w=w^0,b=b^0}$$
>
> 计算完后更新 w 跟 b，把 w^0 减掉学习率乘上微分的结果得到w^1，把 b^0 减掉学习率乘微分的结果得到 b^1。
>
> $$w^1 \leftarrow w^0 - \eta \frac{\partial L}{\partial w}|_{w=w^0,b=b^0}$$
>
> $$b^1 \leftarrow b^0 - \eta \frac{\partial L}{\partial b}|_{w=w^0,b=b^0}$$
>
> 在深度学习框架里面，比如 PyTorch 里面，算微分都是程序自动帮计算的。就是反复同样的步骤，就不断的更新 w 跟 b，期待最后，可以找到一个最好的 w，$w*$ 跟最好的 $b*$. 如图 1.5 所示，随便选一个初始的值，先计算一下 w 对 L 的微分，跟计算一下 b 对 L 的微分，接下来更新 w 跟 b，更新的方向就是 ∂L/∂w，乘以 η 再乘以一个负号，∂L/∂b，算

出这个微分的值，就可以决定更新的方向，可以决定 w 要怎么更新。把 w 跟 b 更新的方向结合起来，就是一个向量，就是红色的箭头，再计算一次微分，再决定要走什么样的方向，把这个微分的值乘上学习率，再乘上负号，我们就知道红色的箭头要指向那里，就知道如何移动 w 跟 b 的位置，一直移动，期待最后可以找出一组不错的 w, b。

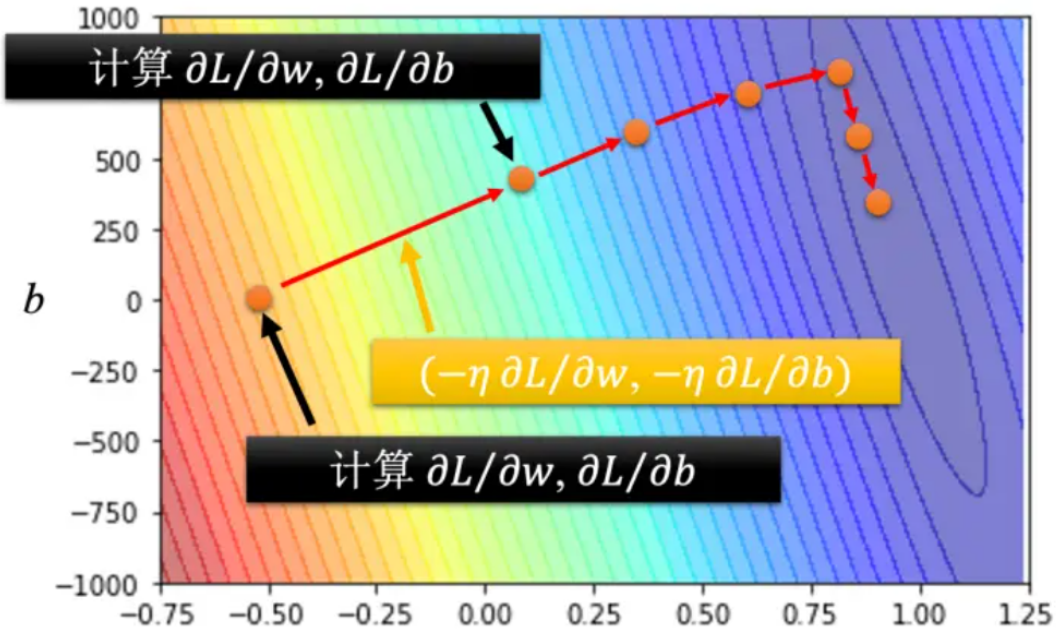


图 1.5　梯度下降优化的过程