浅谈机器学习中的梯度下降

引言

对于几乎所有机器学习算法，无论是有监督学习、无监督学习，还是强化学习，最后一般都归结为求解最优化问题。因此，最优化方法在机器学习算法的推导与实现中占据中心地位。

机器学习要求解的数学模型
对于几乎所有机器学习算法，无论是有监督学习、无监督学习，还是强化学习，迁移学习，最后一般都归结为求解求一个目标函数的极值，即最优化问题。因此，最优化方法在机器学习算法的推导与实现中占据中心地位。

对于有监督学习
Minimum Risk
我们要找到一个最佳的映射函数$f(x)$，使得对训练样本的损失函数最小化（最小化经验风险或结构风险）

$$min_\theta \frac{1}{N}\sum_{i=1}^{N} L(\theta, x_i, y_i) + \lambda \cdot \|\theta\|_2^2$$

在这里，N为训练样本数，L是对单个样本的损失函数，θ是要求解的模型参数，是映射函数的参数，x_i 为样本的特征向量，y_i 为样本的标签值。

Maximum Likelihood Estimate
或是找到一个最优的概率密度函数p(x)，使得对训练样本的对数似然函数极大化（最大似然估计）

$$max \sum_{i=1}^{l} \ln p(x_i; \theta)$$

在这里，θ 是要求解的模型参数，是概率密度函数的参数。

对于无监督学习
以聚类算法为例，算法要使得每个类的样本举例类中心的距离之和最小：

$$min_s \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2$$

对于强化学习，是需要找到一个最优策略（从状态到动作的映射函数），使得对于任意状态执行策略决定的动作后收益最大化。

$$a = \pi(s)$$

$$max_\pi V_\pi(s)$$

而迁移学习，是通过某个预训练的模型经过重用或调整后用于另一个任务中，目标和有监督学习类似。

总体来看，机器学习的核心目标是给出一个模型（一般是**映射函数**），然后定义对这个模型好坏的**评价函数**（目标函数），求解目标函数的极大值或者极小值，以确定模型的参数，从而得到我们想要的模型。

最优化算法

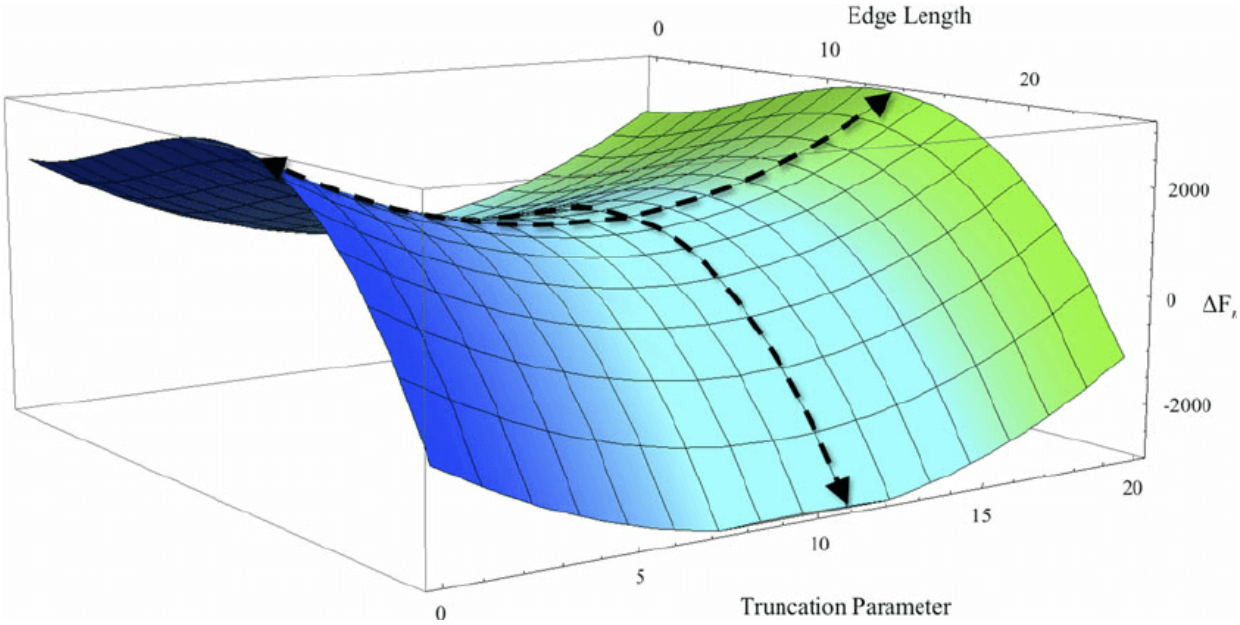
> 对于形式和特点各异的机器学习算法优化目标函数，我们找到了适合它们的各种求解算法。
> https://zhuanlan.zhihu.com/p/42689565

对于多元函数，对各个自变量求偏导数，令它们为0，解方程组，即可达到所有驻点。这都是微积分中所讲授的基础方法。幸运的是，在机器学习中，很多目标函数都是可导的，因此我们可以使用这套方法。

对于一个可导函数，寻找其极值的统一做法是寻找导数为0的点，即费马定理。微积分中的这一定理指出，对于可导函数，在极值点处导数必定为0：$f'(x)=0$，而对于多元函数则是梯度为0。

导数为0的点称为驻点。需要注意的是，导数为0只是函数取得极值的必要不充分条件，它只是疑似极值点。是不是极值，是极大值还是极小值，还需要看更高阶导数。对于一元函数，假设x是驻点：

- $G(X^{(0)})=\left[\begin{array}{cccc}\frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2}\end{array}\right]_{X^{(0)}}$
- $f''(x)>0$ 该点为极小值（Hessian矩阵正定）
- $f''(x)>0$ 该点为极大值（Hessian矩阵负定）
- $f''(x)=0$ 对不起还要看高阶导数（Hessian矩阵不定，则不是极值点）
    - 在导数为0的点处，函数可能不取极值，这称为鞍点



除了极少数问题可以用**暴力搜索**来得到最优解，使用庞大算力计算**公式解**之外，
随机优化算法中还包括例如**模拟退火**（模拟退火算法是解决TSP问题的有效方法之一）、**遗传算法**（通过适应度函数与遗传变异，使用末代作为近似最优解的做法）等
而在我们最常见的将机器学习中使用的优化算法为：
数值优化算法

> 常见的数值优化方法有：AdaGrad，RMSProp，AdaDelta，Adam，SGD
> 刚才提及的牛顿法以外，还有着拟牛顿法、可信域牛顿法、分治法、坐标下降法、SMO、分阶段优化、动态规划等
> https://lumingdong.cn/wp-content/uploads/2018/11/1532531309736972.gif
> https://lumingdong.cn/wp-content/uploads/2018/11/1532533488780334.gif

对绝大多数函数来说，梯度等于 0 的方程组是没法直接解出来的，如方程里面含有指数函数、对数函数之类的超越函数。

不是所有的目标函数都能得到解析解，因此就无法通过数学计算来直接求得其最优解，这个求解最优化参数的过程成为了一个 NP 难问题。既然无法直接求解，那我们可以利用贪心算法的思想，通过不断逼近最优解，来求得其近似值，即数值优化算法。

工程上实现时通常采用的是迭代法，它从一个初始点 x_0 开始，反复使用某种规则从 x_k 移动到下一个点 x_{k+1}：

$$x_{k+1} = h(x_k)$$

构造这样一个数列，直到收敛到梯度为0的点处，即有下面的极限成立：

$$lim_{k \to +\infty} \nabla f(x_k) = 0$$

梯度下降

>

> 梯度下降法沿着梯度的反方向进行搜索，利用了函数的一阶导数信息。
> https://lumingdong.cn/summary-of-gradient-descent-algorithm.html

对于近似计算，泰勒公式是一个非常好的工具。
因此，梯度下降法和牛顿法便被发明了出来，这些数值优化算法一般都利用了目标函数的导数信息，如一阶导数和二阶导数。如果采用一阶导数，则称为一阶优化算法。如果使用了二阶导数，则称为二阶优化算法，梯度下降法本质其实是泰勒公式的一阶展开，而牛顿法则是泰勒公式的二阶展开。

这里只说梯度下降法：
根据函数的一阶泰勒展开，在负梯度方向，函数值是下降的。

$$f(x)=f(x_0)+\frac{f^{'}(x_0)}{1!}(x-x_0)+\frac{f^{''}(x_0)}{2!}(x-x_0)^2+\frac{f^{'''}(x_0)}{3!}(x-x_0)^3+\cdots$$

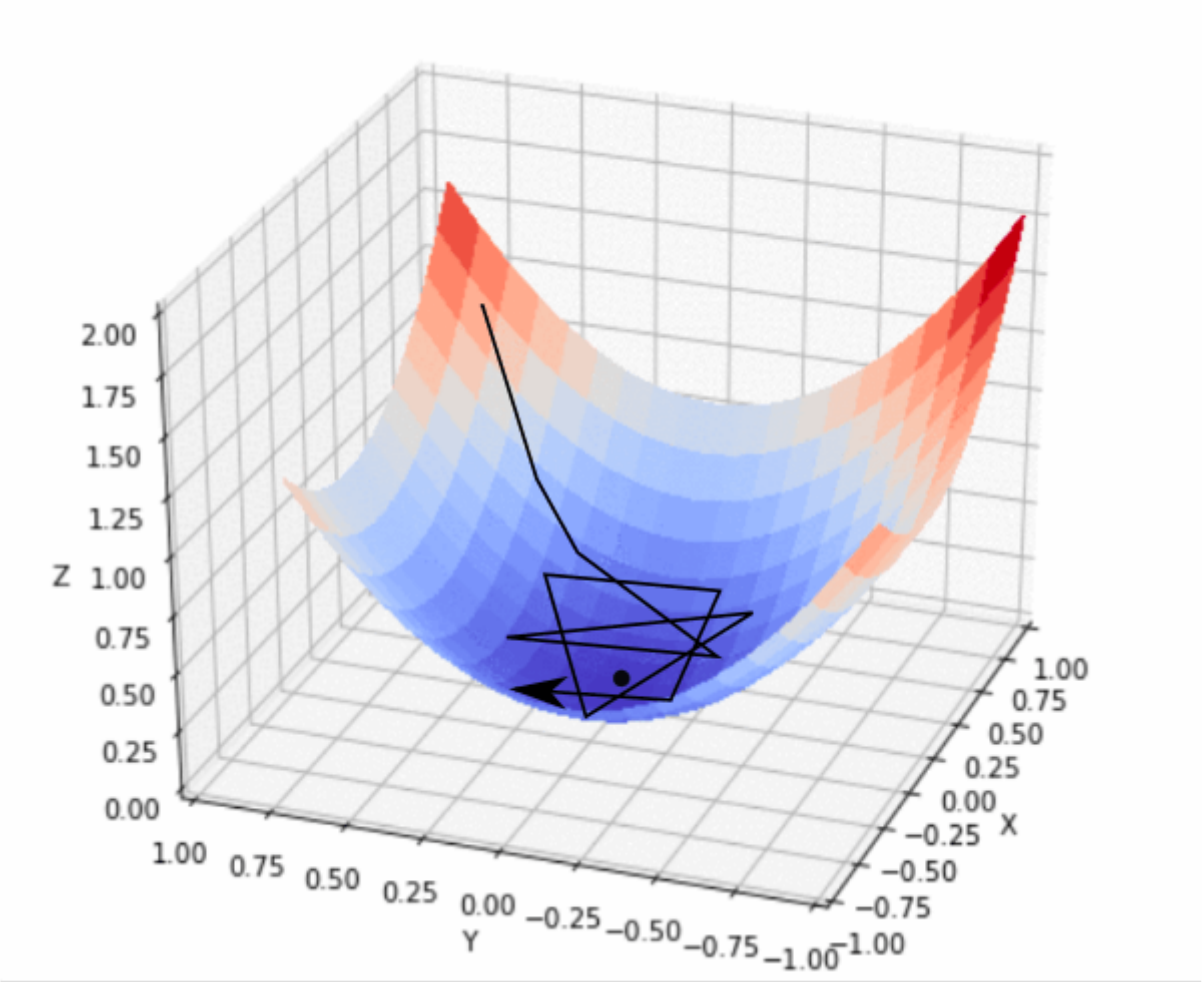
$$f(x)\approx f(x_0)+\frac{f^{'}(x_0)}{1!}(x-x_0)$$

> 泰勒多项式的项数越多（阶数越高）则函数局部线性近似就越精确，不过阶数越高也就意味着算法的复杂度越高，对于追求泛化性能的机器学习来说，通过增加很高的复杂度多出来的一丁点精确度几乎没有什么实际意义。因此，优化算法常用的就是一阶和二阶展开，即梯度下降和牛顿法，因牛顿法使用二阶展开，收敛速度要更快一些。

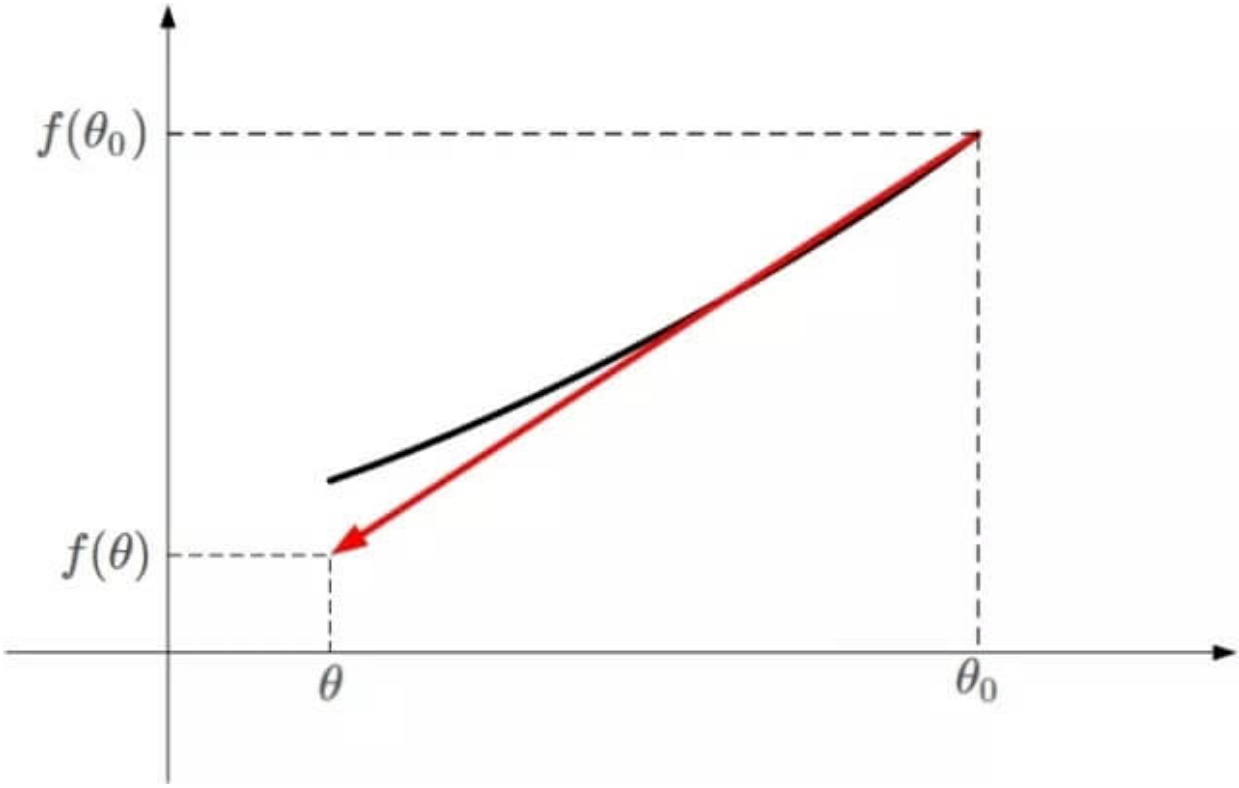
梯度下降法的迭代公式为：

$$x_{k+1}=x_k-\alpha\nabla f(x_k)$$

学习率 α 是一个超参数，用于控制下降或上升步幅的大小。在梯度确定之后，学习率是梯度下降算法中唯一一个必须足够重视的参数。实际上，学习率的设定是梯度下降算法中的一大挑战，学习率设定过小会导致收敛太慢，也可能让算法陷入局部极小值跳不出来；设定过大又容易导致收敛震荡甚至偏离最优点。
只要学习率 α 设置的足够小，并且没有到达梯度为0的点处，每次迭代时函数值一定会下降。



需要设置学习率为一个非常小的正数的原因是要保证迭代之后的 x_{k+1} 位于迭代之前的值 x_k 的邻域内，从而可以忽略泰勒展开中的高次项，保证迭代时函数值下降。



$$f(\theta) \approx f(\theta_0) + (\theta - \theta_0) \cdot \nabla f(\theta_0)$$

$$\textbf{define}\ v \triangleq \frac{\theta-\theta_0}{\|\theta-\theta_0\|_2}$$

$$\theta-\theta_0 = \alpha v, \|v\|_2 = 1$$

局部下降的目的是希望每次 θ 更新，都能让函数值 $f(\theta)$ 变小。也就是说，上式中，我们希望

$$f(\theta) < f(\theta_0)$$

$$f(\theta)-f(\theta_0) \approx \alpha v \cdot \nabla f(\theta_0) < 0$$

因为 α 为标量，且一般设定为正值，所以可以忽略，不等式变成了：

$$v \cdot \nabla f(\theta_0) < 0$$

$$\theta_{k+1} = \theta_k - \alpha_k \nabla J(\theta_k) \Rightarrow J(\theta_k) \geq J(\theta_{k+1}), \quad 0 \leq k \leq n$$

$$J(\theta_0) \geq J(\theta_1) \geq J(\theta_2) \geq \cdots \geq J(\theta_n)$$

批量梯度下降（BGD, Batch gradient descent）又名香草梯度下降（Vanilla gradient descent），批量梯度下降在对参数执行更新时，在每次迭代中使用所有的样本。
小批量梯度下降（Mini-batch gradient descent ，MBGD）有时也会写作小批量随机梯度下降（Mini-batch stochastic gradient descent，mini-batch SGD），实践中往往也把 mini-batch SGD 简称 SGD：

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} L(\theta, x_i, y_i) + \lambda \cdot \|\theta\|_2^2$$

$$L(\theta) \approx \frac{1}{M}\sum_{i=1}^{M} L(\theta, x_i, y_i) + \lambda \cdot \|\theta\|_2^2,\ M \ll N$$

收敛性证明

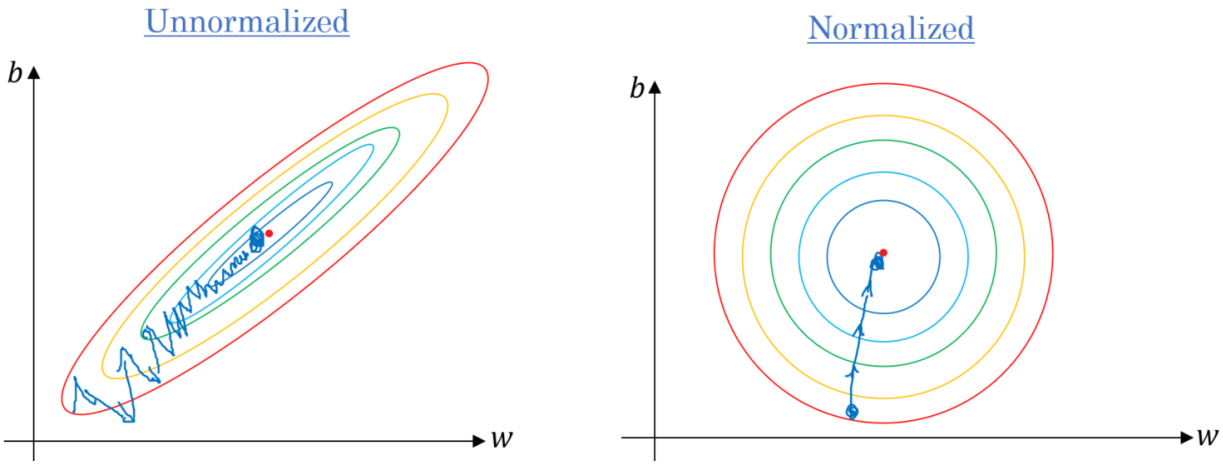
有序列$\{x_t\}$，如果序号t趋于无穷时，满足以下条件：

$$\lim_{t\to\infty} \frac{x_{t+1} - x^*}{x_t - x^*} = \mu$$

则称该方法收敛到$x*$，收敛率为$\mu, \mu \in (0,1)$。也称为以μ收敛到x^*。
当t趋于无穷时，代价函数$f(x_t)$收敛到最优解$f(x^*)$，需要基于 $Lipschitz$ 连续 与 β 平滑的性质可以证明收敛率为 $\epsilon_t = O(1/t)$。
https://blog.csdn.net/shenxiaolu1984/article/details/52577996

其它问题

尺度缩放



$$\frac{x-\mu}{\sigma}$$

标准化可以使用 z-score 标准化方式，即让数据分布满足μ=0 和σ=1，故也称零-均值标准化。

动量处理波动

(a) SGD without momentum　　(b) SGD with momentum

$$\nu_t = \gamma\nu_{t-1} + \eta\nabla_\theta J(\theta)$$

$$\theta = \theta - \nu_t$$

利用惯性，即当前梯度与上次梯度进行加权，如果方向一致，则累加导致更新步长变大；如果方向不同，则相互抵消中和导致更新趋向平衡。

Adam

Adaptive Moment Estimation (自适应矩估计 Adam) 是另外一种为每个参数提供自适应学习率的方法。

同 RMSprop、Adadelta 相比，Adam 在对梯度平方 (二阶矩) 估计的基础上增加了对梯度 (一阶矩) 的估计，使得整个学习过程更加稳定。

$$E[g]t = \beta_1 E[g]_{t-1} + (1-\beta_1)g_t$$

$$E[g^2]t = \beta_2 E[g^2]_{t-1} + (1-\beta_2)g_t^2$$

由于 $E[g]_t$ 和 $E[g^2]_t$ 通常都初始化为 0， 因此在训练初期或者 β_1 ，β_2 接近 1 时，估计的期望往往偏向于 0。

为了解决这种偏置，增加了偏基矫正：

$$\hat{E[g]}_t = \frac{E[g]_t}{1-\beta_1}$$

$$\hat{E[g^2]}_t = \frac{E[g^2]_t}{1-\beta_2}$$

于是最终的更新公式为：

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{E[g^2]}_t + \epsilon}} \hat{E[g]}_t$$

通常 β_1 ，β_2 分别被设置为 0.9 和 0.999。