机器学习：优化方法基础(Fundamentals of Optimizing Method)

---

1. 基本概念

大部分机器学习都是最优化问题，而最优化问题的主要目标，就是求函数极小值：

$$\min f(x) \Leftrightarrow \max\big(-f(x)\big)$$

梯度: $\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$ 。

方向导数 $=$ 方向余弦向量 $\cdot$ 梯度。

黑塞矩阵 $\nabla^2 f(x) \triangleq [\frac{\partial^2 f}{\partial x_i \partial x_j}]$ ，为 n 阶对称矩阵，实质上就是对梯度向量的每个分量再次求梯度。

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 x_n} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_n x_1} & \frac{\partial^2 f}{\partial x_n x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

定理一：函数 $f(x)$ 在 x_0 处取极值，则 $\nabla f(x_0) = 0$ 。(必要性)

满足 $\nabla f(x_0) = 0$ 的点，要么是极值点，要么是鞍点，这些点统称为稳定点，即梯度为 0 是极值点的一个必要非充分条件。

定理二：若 f 是二阶可微函数，$\begin{cases} \nabla f(x_0) = 0 \\ \nabla^2 f(x_0) \geqslant 0 \end{cases} \Rightarrow$ 则 f 在 x_0 处取得极值点。

判定极值点 *or* 鞍点方法：黑塞矩阵为正定矩阵，就是极小值点。

($Sylvester$ 判据：如果 A 的所有顺序主子式都 $\geqslant 0$ ，那么 A 是正定矩阵。)

定理三：矩阵可逆推导。

n 阶矩阵 A 可逆，则 $|A| \neq 0$ ； $r(A) = n$(满秩)；列向量 $a_1, \cdots a_n$ 线性无关；对应的齐次方程组只有 0 解； A 的 n 个特征值非 0 。

n 阶矩阵 A 不可逆，则 $|A| = 0$ ； $r(A) < n$；列向量 $a_1, \cdots a_n$ 线性相关；对应的齐次方程组有非 0 解。

2. 函数凹凸性

参考欧美教材定义：

对于函数 $f(x)$ ，对于其定义域内的任意两点 x, y ，以及任意的实数 $0 \leqslant \theta \leqslant 1$ ，都有：

$$f(\theta x + (1-\theta)y) \leqslant \theta f(x) + (1-\theta) f(y)$$

则函数为凸函数。如果上式满足 $<$ ，则为严格凸函数。(曲线/曲面向下凸)

对于函数 $f(x)$，对于其定义域内的任意两点 x, y，以及任意的实数 $0 \leqslant \theta \leqslant 1$，都有：

$$f(\theta x + (1-\theta)y) \geqslant \theta f(x) + (1-\theta)f(y)$$

则函数为凹函数。如果上式满足 $>$，则为严格凹函数。(曲线/曲面向上凸)

对多元函数，假设 $f(x)$ 二阶可导，$f(x)$ 为凸函数的充要条件是：$f''(x) \geqslant 0$ 。

对多元函数，假设 $f(x)$ 二阶可导，如果对应的黑塞矩阵半正定($x^T A x \geqslant 0$；或者 A 的特征值均非负)，则为凸函数；如果黑塞矩阵正定($xAx^T > 0$；或者 A 的特征值均为正)，则为严格凸函数。凹函数为对应的黑塞矩阵半负定...

相关的凸优化内容参考Lesson 9 凸优化。

3. 极值判别法则

一元函数极值的必要条件，极值点处导数为 0 。(费马定理)

一元函数极值的充分条件，假设 x_0 是 $f(x)$ 的驻点(导数为 0)：

- $f''(x_0) > 0$，该点有严格极小值。
- $f''(x_0) < 0$，该点有严格极大值。
- $f''(x_0) = 0$，该点可能是极值点 -> 假设 $f'(x_0) = \cdots = f^{(n-1)}(x_0) = 0$，且 $f^{(n)}(x_0) \neq 0$ 。如果 n 为偶数，则 x_0 是极值点：$f^{(n)} > 0$，为严格极小值；$f^{(n)} < 0$，为严格极大值。如果 n 为奇数，则 x_0 不是极值点。

多元函数极值的必要条件，极值点处梯度为 0 。

多元函数极值的充分条件，假设 $x_0, y_0, \cdots$ 是 $f(x, y, \cdots)$ 的驻点(梯度为 0)：

- 黑塞矩阵正定，该点有严格极小值。
- 黑塞矩阵负定，该点有严格极大值。
- 黑塞矩阵不定，该点不是极值点，称为鞍点。

对于特殊的二元函数，假设某点的黑塞矩阵为 $\begin{pmatrix} A & B \\ C & D \end{pmatrix}$ 。

- $AC - B^2 > 0$，则该点是极值点。如果 $A > 0$，则为极小值；如果 $A < 0$，则为极大值。
- $AC - B^2 < 0$，则意味黑塞矩阵不定，该点不是极值点。

4. 一阶优化方法

4.1. 一阶牛顿法(Newton Method)及其变种

除了求极值点，一阶牛顿法常被用来求解一些非线性方程的求根。这部分主要参考的是计算方法的课件，其原理是寻找目标函数一阶近似后梯度为 0 的点，然后逐步逼近极值：

$$\nabla f(x) = 0$$

证明：利用一阶泰勒展开：

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(\xi)}{2!}(x^* - x_0)^2$$

将 $(x^* - x_0)^2$ 看成高阶无穷小量：

$$0 = f(x^*) \approx f(x_0) + f'(x_0)(x - x_0)$$

得到：

$$x^* \approx x_0 - \frac{f(x_0)}{f'(x_0)}$$

由于上式是泰勒公式近似得到的结果，因此这个解不一定是目标函数的逐点，算法需要从初始点开始反复迭代，直到收敛到驻点处：

$$x_{k+1} = x_k - \eta \frac{f(x_k)}{f'(x_k)}$$

η 为人工设置的学习率(步长)，和梯度下降法相同，为了保证能够忽略泰勒公式中的高阶无穷小项。完整的如下所示：

$$\begin{aligned}
&init\ x_0, k = 0 \\
&while\ k < N: \\
&\quad if\ ||f'(x_k)|| < eps\ then \\
&\quad\quad break \\
&\quad end\ if \\
&\quad d_k = -\frac{f(x_k)}{f'(x_k)} \\
&\quad x_{k+1} = x_k - \eta d_k \\
&\quad k = k + 1 \\
&end\ while
\end{aligned}$$

一阶牛顿法有很多其他变种：

重根加速收敛法(x^* 是 f 的 m 重根)：

$$x_{k+1} = x_k - \frac{mf(x_k)}{f'(x)}$$

$$x_{k+1} = x_k - \frac{f(x_k)f'(x_k)}{[f'(x_k)]^2 - f(x_k)f''(x_k)}$$

正割法(需要两个初值 x_0, x_1)：

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

简化牛顿法(平行弦法)：

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_0)}$$

牛顿下山法(λ 为下山因子)：

$$x_{k+1} = \lambda[x_k - \frac{f(x_k)}{f'(x_k)}] + (1-\lambda)x_k = x_k - \lambda\frac{f(x_k)}{f'(x_k)}$$

4.2. 梯度下降法(Gradient Descent)

优化问题：考虑无约束的最优化问题 $\min\ f(x)$ 。给定初始点 $x^{(0)}$ ，寻找序列 $x^{(1)}, x^{(2)} \cdots$ 使函数沿着该序列单调递减。

$$f(x^{(0)}) \geqslant f(x^{(1)}) \geqslant f(x^{(2)}) \geqslant \cdots$$

梯度下降即构造这样的序列，使函数最终达到一个较为满意的最小值点。

梯度下降法是典型的迭代法之一，这类方法的核心，即为如何定义从上一个点到下一个点的规则，一般利用一阶导数(梯度)或者二阶导数(黑塞矩阵)。

原理：假设当前点为 $x^{(k)}$ ，则下一个点为 $x^{(k+1)} = x^{(k)} + t\Delta x^{(k)}, \quad \Delta x^{(k)} = -\nabla f(x^{(k)})$ 。

定义：下降方向。如果 $\exists\, t$ 使得 $f(x_0 + tv) < f(x_0)$ ，则 v 为 f 在 x_0 的下降方向。

泰勒展开：$f(x_k + \Delta x) - f(x_k) = (\Delta f(x))^T \Delta x + o(||\Delta x||)$

算法需要保证移动到下一个点时，函数值减小，则有：

$$\begin{aligned}
&\Delta x > 0, \nabla f(x_k) \leqslant 0 \\
&\Leftrightarrow \Delta x^T\ \nabla f(x_k) \leqslant 0\ (Taylor) \\
&\Leftrightarrow -\nabla f(x_k)^T\ \nabla f(x_k)\ (令 \Delta x = \nabla f(X_k)) \\
&= -\left|\left|\nabla f(x_k)\right|\right|_2^2 \leqslant 0
\end{aligned}$$

定理：对于连续可导函数 f ，如果 $v^T \nabla f(x_0) < 0$ ，则 v 为下降方向。

$$\begin{aligned}
f(x^{(k+1)}) &= f(x^{(k)} + t\Delta x) \\
&= f(x^{(k)} + t\nabla f(x^{(k)})^T \Delta x) \\
&= f(x^{(k)} - t\left|\left|\nabla f(x^{(k)})\right|\right|)
\end{aligned}$$

夹角：$\Delta x^T\ \nabla f(x_k) = ||\Delta x|| \cdot ||f(x_k)|| \cdot \cos\theta$ 。则 $\theta = \pi$ ，即负梯度方向时，函数值下降得是最快的。

步长 t (学习率)是用来保证 $x + \Delta x$ 在 x 的邻域内，从而可以忽略泰勒公式中的 $o(||\Delta x||)$ 项。

迭代终止条件：函数的梯度值为 0 或接近 0 ，此时认为已经达到了极值点。

算法过程：(eps 为人工指定的接近 0 的正数，N 为最大迭代次数)

$$\begin{aligned}
&init\ x_0, k = 0 \\
&while\ ||\nabla f(x_k) > eps||\ or\ K < N: \\
&\ldots x_{k+1} = x_k - t\nabla f(x_k) \\
&\ldots k = k + 1 \\
&end\ while
\end{aligned}$$

4.3. 最速下降法

它的思想和梯度下降法类似，但是每次需要计算最佳步长 t^* 。

步长(学习率)确定：$t_k = \arg\min_{t \geqslant 0} f(x_k - t\nabla f(x_k))$ 。

其学习率的计算有两种优化方法：

第一种是取多个典型值$(0.0001, 0.001, 0.01)$，然后分别计算他们的目标函数值，确定最优值。在后续牛顿法这个无法保证函数值下降的二阶优化方法中，学习率的确定将用到典型值方法。

第二种方法是直线搜索算法：以 t 为自变量，直接求上式的驻点，对于有些情况可以得到解析解。直线搜索沿着某一确定的方向在直线上寻找最优步长。在后续拟牛顿法中(DFP, BFGS)，学习率的确定将用到直线搜索算法。

4.4. 随机梯度下降 SGD(Stochastic Gradient Descent)

随机梯度下降算法非常重要，深入理解可以看完这部分基本概念后，去看Lesson 4 监督学习之回归(Linear, NonLinear, Lasso, Ridge, Generalization)中 1.2 的内容。

在使用传统的梯度下降算法时，每次更新参数都需要使用所有的样本。如果对所有的样本均计算一次最后取梯度平均值，当样本总量特别大时，对算法的速度和效率影响非常大。所以就有了随机梯度下降(stochastic gradient descent, SGD)算法，它是对梯度下降法算法的一种改进，即每次只使用部分样本来计算梯度的平均值。

在机器学习和深度学习中，SGD 通常指小批随机梯度下降(mini-batch gradient descent)算法，即每次只随机取一部分样本($M \ll N$)进行优化， 样本的数量一般是 2 的整数次幂，取值范围一般是 $32 \sim 256$ ，以保证计算精度的同时提升计算速度，是优化深度学习网络中最常用的一类算法。

本质上，即损失从整体样本的平均损失，变成了这个 batch 样本的损失：

$$L(w) = \frac{1}{N}\sum_{i=1}^{N} L(w, x_i, y_i) \rightarrow$$

$$L(w) = \frac{1}{M}\sum_{i=1}^{M} L(w, x_i, y_i)$$

SGD 算法及其一些变种，是深度学习中应用最多的一类算法。其在训练过程中，通常会使用一个固定的学习率进行训练。即：

$$g_t = \nabla_{\theta_{t-1}} f(\theta_{t-1})$$
$$\theta_t = -\eta \cdot g_t$$

式中，g_t 是第 t 步的这 M 个样本的平均梯度，η 则是学习率，随机梯度下降算法在优化时，完全依赖于当前 batch 数据计算得到的梯度，而学习率则是调整梯度影响大小的参数，通过控制学习率 η 的大小，一定程度上可以控制网络的训练速度。

SGD 随机采样产生的梯度的期望值是真实的梯度。在具体实现时，DataLoader 每次都要对样本进行 shuffle ，然后均匀分成多个 batch，每份 M 个样本，接下来用每一份分别计算各自 batch 的梯度，最后迭代优化 x_k 。由于每次优化时，并没有使用全部样本来更新，SGD 其实并不能保证每次迭代后目标函数值一定下降，但是能保证整体是呈下降趋势的，能够收敛到局部极值点处。

SGD 虽然在大多数情况下都很有效，但其还存在一些缺点。如很难确定一个合适的学习率 η，而且所有的参数使用同样的学习率可能并不是最有效的方法。针对这种情况，可以采用变化学习率 η 的训练方式，如控制网络在初期以大的学习率进行参数更新，后期以小的学习率进行参数更新。随机梯度下降的另一个缺点就是，其更容易收敛到局部最优解，而且当落入局部最优解后，很难跳出局部最优解的区域。

总结，SGD 的缺点在于，很难确定一个合适的学习率，且容易收敛到局部梯度最小。在第四章第五章中很多算法，都是使用的 SGD 来优化目标函数。在那里我们会详细讲述 SGD 应用于不同的算法。

4.5. 动量的变化

接下来的部分，是深度学习中的一些常用的梯度下降的变种优化算法，可以选择性跳过。

针对随机梯度下降算法的缺点，动量的思想被引入很多神经网络的优化算法中。动量通过模拟物体运动时的惯性来更新网络中的参数，即更新时在一定程度上会考虑之前参数更新的方向，同时利用当前 batch 计算得到的梯度，将两者结合起来计算出最终参数需要更新的大小和方向。在优化时引人动量思想旨在加速学习，特别是面对小而连续且含有很多噪声的梯度。利用动量在一定程度上不仅增加了学习参数的稳定性，而且会更快地学习到收敛的参数。

在引入动量后，网络的参数则按照下面的方式更新:

$$
\begin{aligned}
g_t &= \nabla_{\theta_{t-1}} f(\theta_{t-1}) \\
m_t &= \mu \cdot m_{t-1} + g_t \\
\nabla\theta_t &= -\eta \cdot m_t = -\eta \nabla_{\theta_{t-1}} f(\theta_{t-1}) + \mu m_{t-1}
\end{aligned}
$$

在上述公式中，m_t 为当前动量的累加，μ 属于动量因子，用于调整上一步动量对参数更新时的重要程度。引入动量后，在网络更新初期可利用上一次参数更新，此时下降方向一致， 乘以较大的 μ 能够进行很好的加速。在网络更新后期，随着梯度 g_t 逐渐趋近于 0 ，在局部最小值来回震荡的时候，利用动量使得更新幅度增大，跳出局部最优解的陷阱。

Nesterov 项(Nesterov 动量)是在梯度更新时做出的校正，避免参数更新太快，同时提高灵敏度。在动量中，之前累积的动量 m_t 并不会直接影响当前的梯度 g_t ，所以 Nesterov 的改进就是让之前的动量直接影响当前的动量，即

$$
\begin{aligned}
g_t &= \nabla_{\theta_{t-1}} f(\theta_{t-1} - \eta \cdot \mu \cdot m_{t-1}) \\
m_t &= \mu \cdot m_{t-1} + g_t \\
\nabla\theta_t &= -\eta \cdot m_t
\end{aligned}
$$

Nesterov 动量和标准动量的区别在于，在当前 batch 梯度的计算上，Nesterov 动量的梯度计算是在施加当前速度之后的梯度。所以，Nesterov 动量可以看作是在标准动量方法上添加了一个校正因子，从而提升算法的更新性能。

4.6. 自适应学习率

在训练开始的时候，参数会与最终的最优值点距离较远，所以需要使用较大的学习率，经过几轮训练之后，则需要减小训练学习率。因此，在众多的优化算法中，不仅有通过改变更新时梯度方向和大小的算法，还有一些算法则是优化了学习率等参数的变化，如一 系列 自适应学习率的算法 Adadelta、RMSProp及 Adam 等自适应学习率算法。

很多网站介绍了多种优化算法，如 https://ruder.io/optimizing-gradient-descent/ 在一个通用的问题下求解其路径，其过程截图如图所示。

我们将在下面详细讲解其中常用的算法：AdaGrad, RMSProp, AdaDelta, Adam 算法。

4.6.1. AdaGrad 算法(Adaptive Gradient)

AdaGrad 算法根据前几轮迭代的历史梯度值动态计算步长(学习率)值，且优化向量每一个都有自己的步长：

$$(x_{k+1})_i = (x_k)_i - \eta \frac{(g_k)_i}{\sqrt{\sum_{j=1}^{k}((g_j)_i)^2 + \epsilon}}$$

η 是人工设定的全局学习率，g_k 是第 k 次迭代时的梯度向量，ϵ 是为了避免除以 0 操作而增加的接近于 0 的整数，i 为向量的分量下标，这里的计算针对向量的每个分量分别进行。

与 GD 和 SGD 不同，它多了一个分母项，这个分母项用来累积到本次为止的梯度的历史值信息，用于计算新的步长值。历史导数值的绝对值也大，在改分量上的学习率越久越小。

这种方法存在的问题就是，他需要人工设置全局学习率 η；且随着时间的累积，分母项会越来越大，导致学习率为 0，模型无法更新参数。

4.6.2. RMSProp 算法

RMSProp 算法是对 AdaGrad 算法的改进，避免了长期累积梯度值所导致的学习率趋于 0 的问题。算法维持一个梯度平方累加值的向量 $E[g^2]$，其初始值为 0，更新公式为：

$$E[g^2]_k = \delta E[g^2]_{k-1} + (1-\delta)g_k^2$$

这里 g^2 是对梯度向量的每个分量分别进行平方，δ 是人工设定的衰减系数。和 AdaGrad 直接累加所有历史梯度的平方和不一样，RMSProp 算法将历史梯度的平方按照系数 δ 指数级衰减之后，再累加。本质上使用了移动指数加权平均，其更新公式为：

$$(x_{k+1})_i = (x_k)_i - \eta \frac{(g_k)_i}{\sqrt{(E[g^2]_k)_i + \epsilon}}$$

η 是人工设定的全局学习率。

4.6.3 AdaDelta 算法

AdaDelta 算法也是对 AdaGrad 算法的改进，避免了长期累积梯度值所导致的学习率趋于 0 的问题，不仅如此，它还去掉了对 η 这个全局学习率的依赖。

算法定义了两个 向量，初始值均为 0 (为了不依赖学习率)：

$$E[g^2]_0 = 0 \quad E[\Delta x^2]_0 = 0$$

$E[g^2]$ 和 RMSProp 算法一致，为梯度平方值，更新公式为：

$$E[g^2]_k = \rho E[g^2]_{k-1} + (1-\rho)g_k^2$$

接下来计算 RMS 向量：

$$RMS[g]_k = \sqrt{E[g^2]_k + \epsilon}$$

然后计算更新值：

$$\Delta x_k = -\frac{RMS[\Delta x]_{k-1}}{RMS[g]_k} g_k$$
$$= -\frac{\sqrt{E[\Delta x^2]_k + \epsilon}}{\sqrt{E[g^2]_k + \epsilon}} g_k$$

$E[\Delta x^2]$ 是优化变量更新值的平方累加值，更新公式为

$$E[\Delta x^2]_k = \rho E[\Delta x^2]_{k-1} + (1-\rho)[\Delta x_k^2$$

整体公式为：

$$(x_{k+1})_i = (x_k)_i - \eta \frac{\sqrt{(E[\Delta x^2]_k)_i + \epsilon}}{\sqrt{(E[g^2]_k)_i + \epsilon}} (g_k)_i$$

可以看出，和 AdaGrad 算法比，AdaDelta 算法处理考虑历史梯度的平方和。和 RMSProp 算法比，除了沿用它的历史梯度的衰减机制，来避免分母项过大(导致学习率为 0)；还考虑了历史变量的平方累加值，以便于去掉对人工学习率 η 设置的依赖。

4.7. Adam 算法(Adaptive Moment Estimation)

Adam 优化器结合了前面优化器的思想，综合考虑了动量项和自适应学习率。Adam 算法在 RMSProp 算法基础上对小批量随机梯度也做了指数加权移动平均。Adam 算法可以看做是 RMSProp 算法与动量法的结合。

其主要原理是，对梯度的一阶矩估计(First Moment Estimation ，即梯度的均值)和二阶矩估计(Second Moment Estimation ，即梯度的未中心化的方差)进行综合考虑，计算出每一次迭代更新步长。

Adam 算法使用了动量变量 $\boldsymbol{v}_k$ 和 RMSProp 算法中小批量随机梯度按元素平方的指数加权移动平均变量 $\boldsymbol{m}_k$，并在时间步 0 将它们中每个元素初始化为 0 。

给定超参数 $0 \leq \beta_1 < 1$ （算法作者建议设为 0.9 ），时间步 k 的动量变量 $\boldsymbol{m}_k$ 即小批量随机梯度 $\boldsymbol{g}_k$ 的指数加权移动平均：

$$(\boldsymbol{m}_k)_i \leftarrow \beta_1 (\boldsymbol{m}_{k-1})_i + (1-\beta_1)(\boldsymbol{g}_k)_i$$

和 RMSProp 算法中一样，给定超参数 $0 \leqslant \beta_2 < 1$ （算法作者建议设为 0.999 ）， 将小批量随机梯度按元素平方后的项 $(\boldsymbol{g}_k)_i^2$ 做指数加权移动平均得到学习率 $\boldsymbol{v}_k$ ，学习率衰减机制，可以确保分母项不会过大导致最后学习率为 0：

$$(\boldsymbol{v}_k)_i \leftarrow \beta_2 (\boldsymbol{v}_{k-1})_i + (1-\beta_2)\ (\boldsymbol{g}_k)_i \odot (\boldsymbol{g}_k)_i$$

类似于 AdaDelta 算法，将 $\boldsymbol{v}_0$ 和 $\boldsymbol{m}_0$ 中的元素都初始化为 0 。在时间步 k 我们得到 $\boldsymbol{m}_k = (1-\beta_1)\sum_{i=1}^{k} \beta_1^{k-i} \boldsymbol{g}_i$ 。引入动量思想，即将过去各时间步小批量随机梯度的权值相加，得到 $(1-\beta_1)\sum_{i=1}^{k} \beta_1^{k-i} = 1-\beta_1^k$ 。需要注意的是，当 k 较小时，过去各时间步小批量随机梯度权值之和会较小。例如，当 $\beta_1 = 0.9$ 时，$\boldsymbol{m}_1 = 0.1\boldsymbol{g}_1$ 。为了消除这样的影响，对于任意时间步 k ，我们可以将 $\boldsymbol{m}_k$ 再除以 $1-\beta_1^k$ ，从而使过去各时间步小批量随机梯度权值之和为 1 。这也叫作偏差修正。在 Adam 算法中，我们对变量 $\boldsymbol{m}_k$ 和 $\boldsymbol{v}_k$ 均作偏差修正：

$$(\hat{\boldsymbol{m}}_k)_i \leftarrow \frac{(\boldsymbol{m}_k)_i}{1-\beta_1^k}$$

$$(\hat{\boldsymbol{v}}_k)_i \leftarrow \frac{(\boldsymbol{v}_k)_i}{1-\beta_2^k}$$

接下来， Adam 算法使用以上偏差修正后的变量 $\hat{\boldsymbol{m}}_k \leftarrow \frac{\boldsymbol{m}_k}{1-\beta_1^k}$ 和 $\hat{\boldsymbol{v}}_k$，按照 AdaGrad 算法的思路，将模型参数中每个元素的学习率通过按元素运算重新调整：

$$(\boldsymbol{g}_k')_i \leftarrow \frac{\eta(\hat{\boldsymbol{m}}_k)_i}{\sqrt{(\hat{\boldsymbol{v}}_k)_i}+\epsilon}$$

和上面介绍的几种优化算法一致， η 是全局学习率， ϵ 是为了维持数值稳定性而添加的常数，如 10^{-8} 。**这样，和 AdaGrad 算法、RMSProp 算法以及 AdaDelta 算法一样，目标函数自变量中每个元素都分别拥有自己的学习率。同时，Adam 也引入了动量项，保证了收敛速度。**

最后，使用 $\boldsymbol{g}_k'$ 迭代自变量：

$$\begin{aligned}\boldsymbol{x}_k &\leftarrow \boldsymbol{x}_{k-1} - \boldsymbol{g}_k' \\ \boldsymbol{x}_k &\leftarrow \boldsymbol{x}_{k-1} - \eta\frac{\boldsymbol{m}_k}{\sqrt{\hat{\boldsymbol{v}}_k}+\epsilon} \\ &= \boldsymbol{x}_{k-1} - \eta\frac{\sqrt{1-\beta_2^k}}{1-\beta_1^k}\frac{\boldsymbol{m}_k}{\sqrt{\boldsymbol{v}_k}+\epsilon}\end{aligned}$$

$$for\ each\ variation: (\boldsymbol{x}_k)_i \leftarrow (\boldsymbol{x}_{k-1})_i - \eta\frac{\sqrt{1-\beta_2^k}}{1-\beta_1^k}\frac{(\boldsymbol{m}_k)_i}{\sqrt{(\boldsymbol{v}_k)_i}+\epsilon}$$

5. 二阶优化方法

梯度下降法及其变种只利用了一阶导数信息，收敛速度很慢。通常情况下，利用二阶导数的信息可以加快收敛速度，比如说牛顿法和拟牛顿法。

5.1. 二阶牛顿法(Newton Method)

二阶牛顿法的优化思路类似一阶牛顿法。但是和一阶牛顿法不同，二阶牛顿法是寻找目标函数二阶近似后梯度为 0 的点，然后逐步逼近极值：

$$\nabla f(x) = 0$$

利用二阶泰勒展开($\nabla^2 f(x_0)$ 为 x_0 处的黑塞矩阵)：

$$f(x) = f(x_0) + \nabla f(x_0)^T(x-x_0) + \frac{1}{2}(x-x_0)^T\nabla^2 f(x_0)(x-x_0) + o(||x-x_0||)^2$$

将目标近似为二次函数，对两边同时求梯度：

$$\nabla f(x) \approx \nabla f(x_0) + \nabla^2 f(x_0)\cdot(x-x_0)$$

令 $\nabla f(x)$ 为 0：

$$x = x_0 - (\nabla^2 f(x_0))^{-1} \nabla f(x_0)$$

将梯度向量简写为 g，黑塞矩阵简写为 H，则上式可写为

$$x = x_0 - H^{-1} g$$

由于上式是泰勒公式近似得到的结果，因此这个解不一定是目标函数的逐点，需要从初始点开始反复迭代，直到收敛到驻点处：

$$x_{k+1} = x_k - \eta H_k^{-1} g_k$$

$-H^{-1}g$ 称为牛顿方向，η 为人工设置的学习率(步长)，和梯度下降法相同，为了保证能够忽略泰勒公式中的高阶无穷小项。

$$\begin{aligned}
&init\ x_0, k = 0 \\
&while\ k < N: \\
&\quad count\ g_k, H_k \\
&\quad if\ ||g_k|| < eps\ then \\
&\quad\quad break \\
&\quad end\ if \\
&\quad d_k = -H_k^{-1} g_k \\
&\quad x_{k+1} = x_k - \eta d_k \\
&\quad k = k + 1 \\
&end\ while
\end{aligned}$$

牛顿法无法保证每次迭代时目标函数下降。所以为了确定合适的学习率，通常使用直线搜索技术，即让 η 取一些离散值$(0.0001, 0.001, 0.01)$，选择能使得 $f(x_k + \eta d_k)$ 最小化的学习率作为最优步长，保证迭代后函数值充分下降。

d_k 直接计算，直接求逆计算量太大，所以改为用一些迭代法求下列方程组，得到 d_k，可以减小计算量。

$$H_k d_k = -g_k$$

有时候黑塞矩阵不可逆，所以牛顿法可能失效。

5.2. 拟牛顿法(Quasi-Newton Methods)

牛顿法的核心思路是计算黑塞矩阵后直接求逆，而是通过构造一个近似黑塞矩阵，或者近似逆黑塞矩阵的正定对称矩阵，然后用改矩阵用牛顿法。

将 $f(x)$ 在 x_{k+1} 泰勒展开，忽略高次项：

$$f(x) \approx f(x_{k+1}) + \nabla f(x_{k+1})^T (x - x_{k+1}) + \frac{1}{2} (x - x_{k+1})^T \nabla^2 f(x_{k+1}) (x - x_{k+1})$$

依旧对两边同时求梯度

$$\nabla f(x) \approx \nabla f(x_{k+1}) + \nabla^2 f(x_{k+1}) \cdot (x - x_{k+1})$$

在这一步，不是令 $\nabla f(x)=0$，而是令 $x=x_k$：

$$\nabla f(x_{k+1})-\nabla f(x_k)\approx\nabla^2 f(x_{k+1})(x_{k+1}-x_k)$$

等价于：

$$g_{k+1}-g_k\approx H_{k+1}(x_{k+1}-x_k)$$

如果令 $s_k=x_{k+1}-x_k\quad y_k=g_{k+1}-g_k$，则上式简写为：

$$y_k\approx H_{k+1}s_k$$

如果 H_{k+1} 可逆，上式等价于：

$$s_k\approx H_{k+1}^{-1}y_k$$

上面两个式子称为拟牛顿条件，拟牛顿算法近似代替黑塞矩阵和逆黑塞矩阵需要满足这个条件。根据两点和它们之间的梯度值，就能近似得到当前点的近似黑塞矩阵或者其近似逆矩阵。要求是近似矩阵对称且正定。

5.3. 拟牛顿 - DFP 算法

主要使用 $s_k\approx H_{k+1}^{-1}y_k$，构造近似黑塞逆矩阵。构造近似矩阵：

$$H_{k+1}=H_k+E_k$$

每次迭代时更新此近似矩阵。E_k 称为校正矩阵。根据条件 $s_k\approx H_{k+1}^{-1}y_k$，则有：

$$(H_k+E_k)y_k=s_k$$

上式变形后得到：

$$E_ky_k=s_k-H_ky_k$$

DFP 算法令初始矩阵为单位矩阵 I，更新近似矩阵：

$$H_{k+1}=H_k+\alpha_ku_ku_k^T+\beta_kv_kv_k^T$$

即校正矩阵为 $E_k=\alpha_ku_ku_k^T+\beta_kv_kv_k^T$。$u_k,v_k$ 为待定的 n 维向量，α_k,β_k 为待定系数。显然上式构造的 H_k 是一个对称矩阵。根据上面的拟牛顿条件，得到

$$(\alpha_ku_ku_k^T+\beta_kv_kv_k^T)y_k=s_k-H_ky_k$$

此方程的解不唯一，取某些特殊值从而简化问题的求解，令：

$$\alpha_ku_ku_k^Ty_k=s_k\quad\beta_kv_kv_k^Ty_k=-H_ky_k$$
$$u_k=s_k\quad v_k=H_ky_k$$

带入上面的方程，得到

$$\alpha_k s_k s_k^T y_k = \alpha_k s_k (s_k^T y_k) = \alpha_k (s_k y_k) s_k^T = s_k$$
$$\beta_k H_k y_k (H_k y_k)^T y_k = \beta_k H_k y_k y_k^T H_k^T y_k = \beta_k H_k y_k y_k^T H_k y_k$$
$$= \beta_k H_k y_k (y_k^T H_k^T y_k) = \beta_k (y_k^T H_k y_k) H_k y_k = -H_k y_k$$

上面两个结果利用了矩阵乘法结合律以及 H_k 是对称矩阵，从而解得：

$$\alpha_k = \frac{1}{s_k^T y_k} \quad \beta_k = -\frac{1}{y_k^T H_k y_k}$$

将上面的解带入更新近似矩阵的公式，即可得：

$$H_{k+1} = H_k + \frac{s_k s_k^T}{s_k^T y_k} - \frac{H_k y_k (H_k y_k)^T}{y_k^T H_k y_k}$$

此公式能保证 H_k 的对称正定性，由于构造的是黑塞矩阵逆矩阵的近似($H_{k+1}^{-1} \approx (H_k + E_k)$)，所以可以直接将其与梯度向量相乘从而得到拟牛顿方向。

$$\begin{aligned}
&init\ x_0, H_0 = I, k = 0 \\
&while\ k < N: \\
&\quad d_k = -H_k g_k \\
&\quad 直线搜索得到步长\ \eta_k \\
&\quad x_{k+1} = x_k - \eta_k d_k \\
&\quad if\ ||g_{k+1}|| < eps\ then \\
&\quad\quad break \\
&\quad end\ if \\
&\quad y_k = g_{k+1} - g_k \\
&\quad H_{k+1} = H_k + \frac{s_k s_k^T}{s_k^T y_k} - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k} \\
&\quad k = k + 1 \\
&end\ while
\end{aligned}$$

5.4. 拟牛顿 - BFGS 算法

主要使用 $y_k \approx H_{k+1} s_k$ ，构造近似黑塞矩阵。构造近似矩阵：

$$B_{k+1} = B_k + \Delta B_k$$

每次迭代时更新此近似矩阵。ΔB_k 称为校正矩阵。根据条件 $y_k \approx H_{k+1} s_k$ ，则有：

$$(B_k + \Delta B_k) s_k = y_k$$

上式变形后得到：

$$\Delta B_k s_k = y_k - B_k y_k$$

BFGS 算法同样令初始矩阵为单位矩阵 I ，更新近似矩阵：

$$B_{k+1} = B_k + \alpha_k u_k u_k^T + \beta_k v_k v_k^T$$

即校正矩阵为 $\Delta B_k = \alpha_k u_k u_k^T + \beta_k v_k v_k^T$ 。u_k, v_k 为待定的 n 维向量，α_k, β_k 为待定系数。显然上式构造的 H_k 是一个对称矩阵。根据上面的拟牛顿条件，得到

$$(\alpha_k u_k u_k^T + \beta_k v_k v_k^T) s_k = y_k - B_k y_k$$

此方程的解不唯一，取某些特殊值从而简化问题的求解，令：

$$\alpha_k u_k u_k^T s_k = y_k \quad \beta_k v_k v_k^T s_k = -B_k s_k$$
$$u_k = y_k \quad v_k = B_k s_k$$

带入上面的方程，得到：

$$\alpha_k y_k y_k^T s_k = \alpha_k y_k (y_k^T s_k) = \alpha_k (y_k^T s_k) y_k = y_k$$
$$\beta_k (B_k s_k)^T s_k B_k s_k = \beta_k s_k^T B_k^T s_k B_k s_k = \beta_k (s_k^T B_k s_k) B_k s_k = -B_k s_k$$

上面两个结果利用了矩阵乘法结合律以及 H_k 是对称矩阵，从而解得：

$$\alpha_k = \frac{1}{y_k^T s_k} \quad \beta_k = -\frac{1}{s_k^T B_k s_k}$$

将上面的解带入更新近似矩阵的公式，即可得：

$$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k (B_k s_k)^T}{s_k^T B_k s_k}$$

此公式能保证 B_k 的对称正定性，由于构造的是黑塞矩阵的近似($B_{k+1} \approx (B_k + \Delta B_k)$)，所以还需要求解方程组，以得到拟牛顿方向。而 B_k 是正定对称矩阵，因此可以采用高效的方法求解此线性方程组

$$\begin{aligned}
&init\ x_0, B_0 = I, k = 0 \\
&while\ k < N: \\
&\quad d_k = -B_k^{-1} g_k \\
&\quad \text{直线搜索得到步长 } \eta_k \\
&\quad x_{k+1} = x_k - \eta_k d_k \\
&\quad if\ ||g_{k+1}|| < eps\ then \\
&\quad\quad break \\
&\quad end\ if \\
&\quad y_k = g_{k+1} - g_k \\
&\quad B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k (B_k s_k)^T}{s_k^T B_k s_k} \\
&\quad k = k + 1 \\
&end\ while
\end{aligned}$$

B_k 是个 $n \times n$ 的矩阵，计算量很大。因此改进的 L-BFGS 算法(有限存储)，思路是不储存完整的 B_k ，只储存向量 s_k, y_k 。

DFP 和 BFGS 算法实际上互成对偶，本质上将 s_k 和 y_k 的角色做了对换。

6. 分治法

6.1. 坐标下降法(Coordinate Descent)

对于多元函数的优化问题，坐标下降法每次只针对一个分量进行优化，而让其他的分量固定不定。算法依次优化每一个变量，直至收敛，假定优化问题为：

$$\min f(x), x = (x_1, x_2, \cdots, x_n)$$

算法流程如下：

$$\begin{aligned}
&init\ x_0\\
&while\ 没有收敛：\\
&\quad for\ i = 1, 2, \cdots, n:\\
&\quad\quad solve\ \min_{x_i} f(x)\\
&\quad end\ for\\
&end\ while
\end{aligned}$$

算法对于当前点，固定其他坐标轴方向对应的分量，而只沿一个坐标轴方向进行一维搜索，求解一元函数的极值。一个周期的一维搜索迭代相当于一次 GD ，完成对变量的每个分量的一次更新。

其典型应用是求解线性模型的训练问题，在 python 的库 liblinear 中有实现；求解非负矩阵的分解也有应用。

其典型的缺点是对非光滑(不可导)的多元目标函数无法有效处理，以及难以并行化。对于目标函数 $f(x, y) = |x + y| + 3|y - z|$ ，对于点 $(-2, -2)$ ，无法用坐标分治法来保证目标函数值下降。

6.2. SMO 算法(Sequential Minimal Optimization)

详见Lesson 6 支撑向量机线性不可分的支撑向量机求解部分。

6.3. 分阶段优化(AdaBoost)

详见Lesson 13 集成学习集成学习的 $AdaBoost$ 算法。

6.4. Logistic 回归中的坐标下降法

详见Lesson 5 监督学习之分类(Perceptron, Fisher, Logistic, Softmax, Bayes) $Logistic$ 回归中坐标下降法。

7. 黄金搜索算法

问题：对于 $[a, b]$ 的连续函数 $f(x)$ ， 求 $\min f(x), \quad x \in [a, b]$ 。

方法一： $\frac{b-a}{\varepsilon}$ 次计算 f 的值

优化：

> 计算 f 的次数尽可能少。
>
> 希望能够不断缩小定义域范围。

$$
\begin{aligned}
&1.f(x_1) \leqslant f(x_2),\ \text{保留}[a, x_2] \\
&2.f(x_1) > f(x_2),\ \text{保留}[x_1, b] \\
&c = \frac{x_2 - a}{b - a} = \frac{b - x_1}{b - a} \\
&x_2 = (1 - c)a + cb \quad x_1 = ca + (1 - c)b
\end{aligned}
$$

如何确定 c？

$$
\begin{aligned}
x_2^{'} = x_1 &= (1 - c)b + ca \quad (\text{与}x_1\text{重合}) \\
&= (1 - c)a + cx_2 \\
&= (1 - c)a + c(1 - c)a + c^2 b
\end{aligned}
$$

所以：

$$
\begin{aligned}
(1 - c)b + ca &= (1 - c^2)a + c^2 b \\
(c^2 + c - 1)a &= (c^2 + c - 1)b \\
\Rightarrow c^2 + c - 1 = 0 \quad c &= \frac{-1 + \sqrt{5}}{2} \\
c &= 0.618
\end{aligned}
$$

8. 机器学习的分类

监督学习，无监督学习，半监督学习，自监督学习，元学习，迁移学习，强化学习，集成学习，深度学习，对抗学习...