基础概念回顾

那么在开始讲解前，我们先回顾一下数学上的一些基本概念。

0.1 函数的近似

假设$y=f(x)$在x_0处可导且导数$y'=f(x_0)\neq 0$, 且$|\triangle x|$很小时（$\triangle$表示增量），有：

$dy=f'(x)\triangle x$

即 $dy\approx\frac{f(x_0+\triangle x)-f(x_0)}{\triangle x}\triangle x$

移项则得 $f(x_0+\triangle x)\approx f(x_0)+f'(x_0)\triangle x$

此时令$x=x_0+\triangle x$, 则有：$f(x)\approx f(x_0)+f'(x_0)(x-x_o)$

0.2 泰勒展开

函数f(x)在x_0处展开的带有拉格朗日型余项的n次泰勒公式如下：

$f(x)=f(x_0)+f'(x_0)(x-x_o)+\frac{f''(x_0)}{2!}(x-x_0)^2+\cdots+\frac{f^n(x_0)}{n!}(x-x_0)^n+R_n(x)$, 其中$R_n(x)$为拉格朗日型余项。

0.3 L1\L2正则

- L1正则化是根据权重的绝对值的总和来惩罚权重。
  公式：$l_1:\Omega(\omega)=||\omega||1=\sum_i^N|w_i|$
- L2正则化是根据权重的平方和来惩罚权重。
  公式：$l_2:\Omega(\omega)=||\omega||2=\sum_i^N w_i^2$

不同的最优化方法

1. 梯度下降法 （Gradient Descent)

梯度下降法是最早最简单，也是最为常用的最优化方法。梯度下降法实现简单，当目标函数是凸函数时，梯度下降法的解是全局解。一般情况下，其解不保证是全局最优解，梯度下降法的速度也未必是最快的。**梯度下降法的优化思想是用当前位置负梯度方向作为搜索方向，因为该方向为当前位置的最快下降方向，所以也被称为是"最速下降法"。最速下降法越接近目标值，步长越小，前进越慢。**

1.1 梯度（Gradient)

导数

【导数】：导数我们都非常熟悉，既可以表示某点的**切线斜率**，也可以表示某点**变化率**，公式如下表示：

$f'(x)=\lim_{\triangle x\to 0}\frac{\triangle y}{\triangle x}=\lim_{\triangle x\to 0}\frac{f(x+\triangle x)-f(x)}{\triangle x}$

当函数是多元的，导数就变成了偏导数：如$f_x(x,y)$表示当y不变时，$f(x,y)$沿着x轴的变化率；那么同理，$f_y(x,y)$表示当x不变时，$f(x,y)$沿着y轴的变化率；其公式表达如下：

$f_x(x,y)=\lim_{\triangle x\to 0}\frac{f(x+\triangle x,y)-f(x,y)}{\triangle x}$

$f_y(x,y)=\lim_{\triangle y\to 0}\frac{f(x,y+\triangle y)-f(x,y)}{\triangle y}$

但是多元函数是一个平面，方向有很多， x轴、y轴只是其中两个方向而已，假如我们需要其他方向的变化率怎么办呢？这时候**方向导数**就有用了，顾名思义，方向导数可以表示任意方向的导数。

> 方向导数

【方向导数】：

设：二次函数$f(x,y)$，方向（单位向量）$u=\cos\theta i+\sin\theta j$，那么该二次函数在这个方向下的导数为：

$D_uf=\lim_{t\to0}\frac{f(x+t\cos\theta,y+t\sin\theta)-f(x,y)}{t}=f_x(x,y)\cos\theta+f_y(x,y)\sin\theta=[f_x(x,y),f_y(x,y)]\begin{bmatrix}\cos\theta\\sin\theta\end{bmatrix}$

我们记为$D_uf=A\times I=abs(A)abs(I)\cos\alpha$, 其中$\alpha$是两向量的夹角。我们可以推断出当$\alpha$等于0时，方向导数$D_uf$达到最大值，此时的方向导数即为梯度。

> 梯度公式

【梯度的定义▽】

梯度的定义：某一函数沿着某点处的方向导数可以以最快速度到达极大值，该方向导数我们定义为该函数的梯度。

$\bigtriangledown=\frac{df(\theta)}{d\theta}$

其中θ是自变量，根据泰勒展开式我们可以得出**梯度下降公式**：

$\theta=\theta_0-\eta\cdot\bigtriangledown f(\theta_0)$

其中η是步长 。[1]

从几何意义上来说，梯度向量就是函数变化增加最快的地方。

具体来说，对于函数$f(x,y)$，在给定点(x_0,y_0)处，沿着梯度向量$\begin{bmatrix}\frac{\partial f}{\partial x_0}&\frac{\partial f}{\partial y_0}\end{bmatrix}$的方向就是该函数增加最快的地方。

换个理解方式：沿着梯度向量的方向，更加容易找到函数的最大值。那么反过来说，沿着梯度向量相反的方向（去负号），则就是更加容易找到函数的最小值。

1.2 梯度下降的直观解释

比如我们在一座大山上的某处位置，由于我们不知道怎么下山，于是决定走一步算一步，也就是在每走到一个位置的时候，求解当前位置的梯度，沿着梯度的负方向，也就是当前最陡峭的位置向下走一步，然后继续求解当前位置梯度，向这一步所在位置沿着最陡峭最易下山的位置走一步。这样一步步的走下去，一直走到觉得我们已经到了山脚。当然这样走下去，有可能我们不能走到山脚，而是到了某一个局部的山峰低处（局部最小值）。

从上面的解释可以看出，梯度下降不一定能够找到全局的最优解，有可能是一个局部最优解。当然，如果损失函数是凸函数，梯度下降法得到的解就一定是全局最优解。

1.3 梯度下降法迭代公式

了解了梯度下降的基本原理，我们来继续讲一下梯度下降法的具体实现。

设$f(x)$是R^n上具有一阶连续偏导数的函数，我们的目标就是求解一个无约束最优化问题：$f^*=\min_{x\in R^n}f(x)$，其中f^*就是目标函数要求的极小值。

而梯度下降法是一种迭代算法。首先，我们需要选取适当的初值$x^{(0)}$，然后不断迭代，更新x的值，进行目标函数的极小化，直到收敛。根据1.2可知，**负梯度方向是使函数下降最快的的方向**，所以在迭代的每一步，我们需要以负梯度方向更新x的值，从而达到减少函数值的目的。

举个栗子，在梯度下降法的第k+1次迭代值为：$x^{(k+1)}=x^{(k)}+\lambda_kp_k$

其中$p_k=-\bigtriangledown f\left(x^{(k)}\right)$，是搜索方向，即负梯度方向；

其中λ_k是**搜索步长(即stride即学习率lr)**，需要满足$f\left(x^{(k)}+\lambda_kp_k\right)=\min_{\lambda\geq0}f\left(x^{(k)}+\lambda_kp_k\right)$

> 梯度下降法步骤

【输入】： 目标函数$f(x)$, 其梯度函数$g\left(x^{(k)}\right)=\nabla f\left(x^{(k)}\right)$, 计算精度$\epsilon$

【输出】： $f(x)$的极小点$x^{\star}$

【步骤】：

1. 取初始值$x^{(0)} \in R^{n}$, 置k=0
2. 计算$f\left(x^{(k)}\right)$
3. 计算梯度$g_{k}=g\left(x^{(k)}\right)$
   当$\left|g_{k}\right| \leq \varepsilon$ 时，停止迭代，且令$x^{\star}=x^{(k)}$
   否则，令搜索方向$p_{k}=-g\left(x^{(k)}\right)$等于负梯度方向，然后求$\lambda_{k}$,使： $f\left(x^{(k)}+\lambda_{k} p_{k}\right)=\min _{\lambda \geq 0} f\left(x^{(k)}+\lambda_{k} p_{k}\right)$
4. 下一步，置$x^{(k+1)}=x^{(k)}+\lambda_{k} p_{k}$，然后继续计算$f\left(x^{(k+1)}\right)$
   当$\left|f\left(x^{(k+1)}\right)-f\left(x^{(k)}\right)\right| \leq \varepsilon$ 或 $\left|x^{(k+1)}-x^{(k)}\right| \leq \varepsilon$时，停止迭代，且令$x^{\star}=x^{(k+1)}$
   否则k=k+1，转步骤3。

> 具体例子

单变量示例

现有函数$J(x)=x^{2}$

那么导数（单变量情况下也就是他的梯度）为$J'(x)=2x$

所以梯度函数就是$g\left(x^{(k)}\right)=\nabla f\left(x^{(k)}\right)=2x^{(k)}$

设初始点$x^{0}=1$，设置搜索步长（即学习率）$\alpha=0.4$

根据梯度下降\公式，迭代过程如下：

$x^{0}=1$

$x^{1}=x^{0}-\alpha J'\left(x^{0}\right)=1-0.4*2=0.2$

$x^{2}=x^{1}-\alpha J'\left(x^{1}\right)=0.2-0.4*0.4=0.04$

$x^{3}=x^{2}-\alpha J'\left(x^{2}\right)=0.04-0.4*0.08=0.008$

$x^{4}=x^{3}-\alpha J'\left(x^{3}\right)=0.008-0.4*0.016=0.0016$

$J'\left(x^{4}\right)=2*0.0016=0.0032 \approx 0$ 基本找到最低点

多变量示例

现有函数$J(x)=x^{2}+y^{2}$

那么他的梯度函数为$\nabla J(x, y)=\nabla J(\theta)=<2x, 2y>$

设初始点$\theta^{0}=(1,3)$，设置搜索步长（即学习率）$\alpha=0.1$

根据梯度下降公式，迭代过程如下：

$\theta^{0}=(1,3)$

$\theta^{1}=\theta^{0}-\alpha \nabla J\left(\theta^{0}\right)=\left[0.8\ 2.4\right]$

$\theta^{2}=\theta^{1}-\alpha \nabla J\left(\theta^{1}\right)=\left[0.64\ 1.92\right]$

$\theta^{3}=\theta^{2}-\alpha \nabla J\left(\theta^{2}\right)=\left[0.512\ 1.536\right]$

.....

$\nabla J\left(\theta^{100}\right) \approx 0$ 此时梯度接近0，因此在k=100时达到了局部最小值。

1.4 梯度下降算法的调优

在使用梯度下降时，需要进行调优。哪些地方需要调优呢?

1. 算法的步长选择： 步长取值取决于数据样本，可以多取一些值，从大到小，分别运行算法，看看迭代效果，如果损失函数在变小，说明取值有效，否则要增大步长。步长太大，会导致迭代过快，甚至有可能错过最优解。步长太小，迭代速度太慢，很长时间算法都不能结束。所以算法的步长需要多次运行后才能得到一个较为优的值。

2. 算法参数的初始值选择： 初始值不同，获得的最小值也有可能不同，因此梯度下降求得的只是局部最小（当然如果损失函数是凸函数则一定是最优解）。由于有局部最优解的风险，需要多次用不同初始值运行算法，关键损失函数的最小值，选择损失函数最小化的初值。

3. 归一化： 由于样本不同特征的取值范围不一样，可能导致迭代很慢，为了减少特征取值的影响，可以对特征数据归一化。

1.5 梯度下降法的缺点

\1. 靠近极小值时收敛速度减慢，如下图所示；

\2. 直线搜索时可能会产生一些问题；

\3. 寻找的是局部最优，可能会“之字形”地下降。

1.6 梯度下降法大家族（BGD，SGD，MBGD）

> **批量梯度下降法（Batch Gradient Descent）**

批量梯度下降法，是梯度下降法最常用的形式，具体做法也就是在更新参数时使用所有的样本来进行更新。 这样一来每迭代一步，都要用到训练集所有的数据，如果数据量很大，那么可想而知这种方法的迭代速度会很慢。所以，这就引入了另外一种方法，随机梯度下降。

> **随机梯度下降法（Stochastic Gradient Descent）**

随机梯度下降（Stochastic Gradient Descent）每次迭代只用到了一个样本，在样本量很大的情况下，常见的情况是只用到了其中一部分样本数据即可迭代到最优解。因此随机梯度下降比批量梯度下降在计算量上会大大减少。SGD有一个缺点是，其噪音较BGD要多，使得SGD并不是每次迭代都向着整体最优化方向。而且SGD因为每次都是使用一个样本进行迭代，因此最终求得的最优解往往不是全局最优解，而只是局部最优解。但是大的整体的方向是向全局最优解的，最终的结果往往是在全局最优解附近。

> **小批量梯度下降法（Mini-batch Gradient Descent）**

小批量梯度下降法是批量梯度下降法和随机梯度下降法的折衷，也就是对于m个样本，我们采用 x个样子来迭代，1<x<m 。

以多元线性回归为例子

设有线性函数 $f(x) = \omega_0 + \omega_1 x_1 + \cdots + \omega_n x_n = \omega^T x$

那么损失函数就是真实值跟预测值的MSE: $J(\omega) = \sum_{i=1}^{n} \left(\omega^T x^i - y^i\right)^2$

那么梯度就是 $\nabla J(\omega) = \frac{\partial J(\omega)}{\partial \omega} = \sum_{i=1}^{n} 2\left(\omega^T x^i - y^i\right) * x_j^i$

那么批量梯度下降法（Batch Gradient Descent）的更新规则就是：

$\omega_j = \omega_j - 2\alpha \sum_{i=1}^{n} \left(\omega^T x^i - y^i\right) * x_j^i$

小批量梯度下降法（Mini-batch Gradient Descent）的更新规则就是：

$\omega_j = \omega_j - 2\alpha \sum_{i=1}^{m} \left(\omega^T x^i - y^i\right) * x_j^i$

而随机梯度下降法（Stochastic Gradient Descent）的更新规则则为：

$\omega_j = \omega_j - 2\alpha \left(\omega^T x^i - y^i\right) * x_j^i$

根据loss图像我们可以看到，BGD的loss是一直下降的，而SGD的loss虽然整体是下降的，但是中途有升有降 [2]

1.7 批量梯度下降法python实现

```python
def train(X, y, W, B, alpha, max_iters):
    '''
    使用了所有的样本进行梯度下降
    X: 训练集,
    y: 标签,
    W: 权重向量,
    B: bias,
    alpha: 学习率,
    max_iters: 最大迭代次数.
    '''
    dW = 0 # 权重梯度收集器
    dB = 0 # Bias梯度的收集器
    m = X.shape[0] # 样本数

    for i in range(max_iters):
        dW = 0 # 每次迭代重置
        dB = 0
        for j in range(m):
            # 1. 迭代所有的样本
            # 2. 计算权重和bias的梯度保存在w_grad和b_grad,
            # 3. 通过增加w_grad和b_grad来更新dW和dB
            W = W - alpha * (dW / m) # 更新权重
            B = B - alpha * (dB / m) # 更新bias
    return W, B
```

2. 牛顿法（Newton's method）

牛顿法是一种在实数域和复数域上近似求解方程的方法。方法使用函数f(x)的泰勒级数的前面几项来寻找方程f(x)=0的根。牛顿法最大的特点就在于它的收敛速度很快。

2.1 牛顿法的求解方程

并不是所有的方程都有求根公式，或者求根公式很复杂，导致求解困难。利用牛顿法，可以**迭代求解**。原理是利用泰勒公式 [3]，在x_0处展开到一阶，即
$f(x)=f(x_0)+f'(x_0)(x-x_0)$ **【式2.1】**

然后求解$f(x)=0$，解得$x=x_1=x_0-\frac{f(x_0)}{f'(x_0)}$。
这是利用泰勒公式一阶展开求得的近似解，所以x_1并不能让$f(x)=0$
只能说$f(x_1)$的值比$f(x_0)$的值更接近于0， 同理，$f(x_2)$的值比$f(x_1)$的值更接近于0，etc...
所以迭代的想法因此而生，进而推出了公式：$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}$
通过迭代，该式在$f(x^*)=0$的时候收敛，此时我们得方程根。

2.2 牛顿法的迭代公式

同样地，上述求解方程的过程也可以看作是一个**无约束最优化问题**：$\min_{x\subset R^n} f(x)$，我们要找的也就是目标函数的极小点x^*。

设，$f(x)$具有二阶连续偏导数，那么我们就可以对**【式2.1】**两边取导数，得：
$f'(x)=f'(x_0)+f''(x_0)(x-x_0)$,设$\triangle x=x-x_0$
$\Leftrightarrow f'(x)=f'(x_0)+f''(x_0)\triangle x$
根据微积分的性质，f(x)取最小值时，有f'(x)=0，我们把这个性质代入上面的式子，有:
$0=f'(x_0)+f''(x_0)\triangle x$
$\Leftrightarrow \triangle x=-\frac{f'(x_0)}{f''(x_0)}$

这样我们就得到了牛顿法的参数迭代更新公式如下:
$x_{n+1}=x_n-\frac{f'(x_n)}{f''(x_n)}=x_n-\frac{\nabla f(x_n)}{H(f(x_n))}$ **【式2.2】**
其中H就是f(x)的Hessian海塞矩阵 [3]

> 牛顿法步骤

【输入】： 目标函数$f(x)$, 其梯度函数$g\left(x^{(k)}\right)=\nabla f\left(x^{(k)}\right)$, 计算精度$\epsilon$

【输出】： $f(x)$的极小点$x^\star$

【步骤】：

1. 取初始值$x^{(0)}\in R^n$, 置k=0
2. 计算梯度$g_k=g\left(x^{(k)}\right)$
3. 若$|g_k|\leq\varepsilon$，停止迭代，求得近似解$x^\star=x^{(k)}$
4. 计算 $H_k=H\left(x^{(k)}\right)$
5. 置$x^{(k+1)}=x^{(k)}-\frac{g_k}{H_k}$
6. 置k=k+1，转步骤2。

> 具体例子

用牛顿法求解$\min f(x) = \frac{1}{2}x_1^2 + \frac{9}{2}x_2^2, x^0 = [9\ 1]$

- 计算梯度$\nabla f(x) = \left[\frac{\partial f}{\partial x_1}\ \frac{\partial f}{\partial x_2}\right] = [x_1\ 9x_2]$
- 计算海塞矩阵$Hf(x) = \left[\frac{\partial^2 f}{\partial x_1^2}\quad \frac{\partial^2 f}{\partial x_1 \partial x_2}\ \frac{\partial^2 f}{\partial x_2 \partial x_1}\quad \frac{\partial^2 f}{\partial x_2^2}\right] = [1\quad 0\ 0\quad 9]$
- 所以$x^1 = x^0 - \frac{\nabla f(x^0)}{H(f(x^0))} = [9\ 1] - [1\quad 0\ 0\quad 9]^{-1}[9\ 9] = [0\ 0]$
- $|\nabla f(x^1)| = 0 \le \varepsilon$,所以迭代终止，极小点为$x^1 = [0\ 0]$

2.3 牛顿法与梯度下降法的差异

总的来说，梯度法和牛顿法有如下区别：

- 梯度下降法和牛顿法相比，两者都是迭代求解，不过梯度下降法是梯度求解，而牛顿法是用二阶的海森矩阵的逆矩阵求解。相对而言，使用牛顿法收敛更快（迭代更少次数）。但是每次迭代的时间比梯度下降法长。（至于为什么牛顿法收敛更快，通俗来说梯度下降法每次只从你当前所处位置选一个坡度最大的方向走一步，牛顿法在选择方向时，不仅会考虑坡度是否够大，还会考虑你走了一步之后，坡度是否会变得更大。所以，可以说牛顿法比梯度下降法看得更远一点，能更快地走到最底部。）
- 牛顿法对初始值有一定要求，在非凸优化问题中（如神经网络训练），牛顿法很容易陷入鞍点（牛顿法步长会越来越小），而梯度下降法则很容易逃离鞍点（因此在神经网络训练中一般使用梯度下降法，高维空间的神经网络中存在大量鞍点）
- 梯度下降法在靠近最优点时会震荡，因此步长调整在梯度下降法中是必要的，具体有adagrad, adadelta, rmsprop, adam等一系列自适应学习率的方法。

2.4 牛顿法python实现

```python
def newton(f, x, iters):
    """
    实现牛顿法
    :param f: 原函数
    :param x: 初始值
    :param iters: 遍历的最大epoch
    :return:
    """
    Hessian_T = np.linalg.inv(hessian(f, x))
    H_G = np.matmul(Hessian_T, jacobian(f, x))
    x_new = x - H_G
    print("第1次迭代后的结果为:", x_new)
    for i in range(1, iters):
        Hessian_T = np.linalg.inv(hessian(f, x_new))
        H_G = np.matmul(Hessian_T, jacobian(f, x_new))
        x_new = x_new - H_G
        print("第"+str(i+1)+"次迭代后的结果为:", x_new)
    return x_new
```

3. 总结

最基本的最优化方法就介绍到这里，其余的还有拟牛顿法、DFP(Davidon-Fletcher-Powell)算法、BFGS(Broyden-Fletcher-Goldfard-Shano)算法等，有需求的朋友自己查。