Gradient Descent

Review

前面预测宝可梦cp值的例子里，已经初步介绍了Gradient Descent的用法：

In step 3, we have to solve the following optimization problem:

$\theta^* = \arg\min_{\theta} L(\theta)$

L : loss function
θ : parameters(上标表示第几组参数，下标表示这组参数中的第几个参数)

假设θ是参数的集合：Suppose that θ has two variables $\{\theta_1, \theta_2\}$

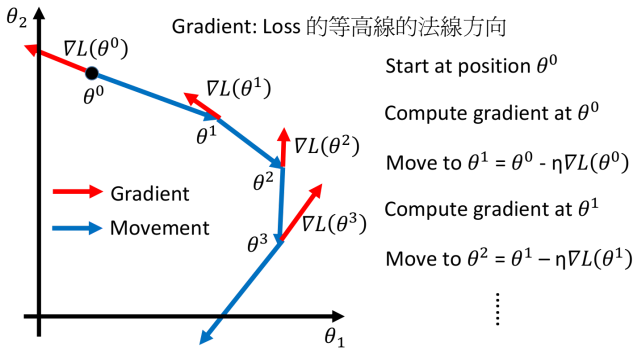
随机选取一组起始的参数：Randomly start at $\theta^0 = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix}$

计算θ处的梯度gradient：$\nabla L(\theta) = \begin{bmatrix} \partial L(\theta_1)/\partial\theta_1 \\ \partial L(\theta_2)/\partial\theta_2 \end{bmatrix}$

$$\begin{bmatrix} \theta_1^1 \\ \theta_2^1 \end{bmatrix} = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix} - \eta \begin{bmatrix} \partial L(\theta_1^0)/\partial\theta_1 \\ \partial L(\theta_2^0)/\partial\theta_2 \end{bmatrix} \Rightarrow \theta^1 = \theta^0 - \eta\nabla L(\theta^0)$$

$$\begin{bmatrix} \theta_1^2 \\ \theta_2^2 \end{bmatrix} = \begin{bmatrix} \theta_1^1 \\ \theta_2^1 \end{bmatrix} - \eta \begin{bmatrix} \partial L(\theta_1^1)/\partial\theta_1 \\ \partial L(\theta_2^1)/\partial\theta_2 \end{bmatrix} \Rightarrow \theta^2 = \theta^1 - \eta\nabla L(\theta^1)$$

下图是将gradient descent在投影到二维坐标系中可视化的样子，图上的每一个点都是$(\theta_1, \theta_2, loss)$在该平面的投影



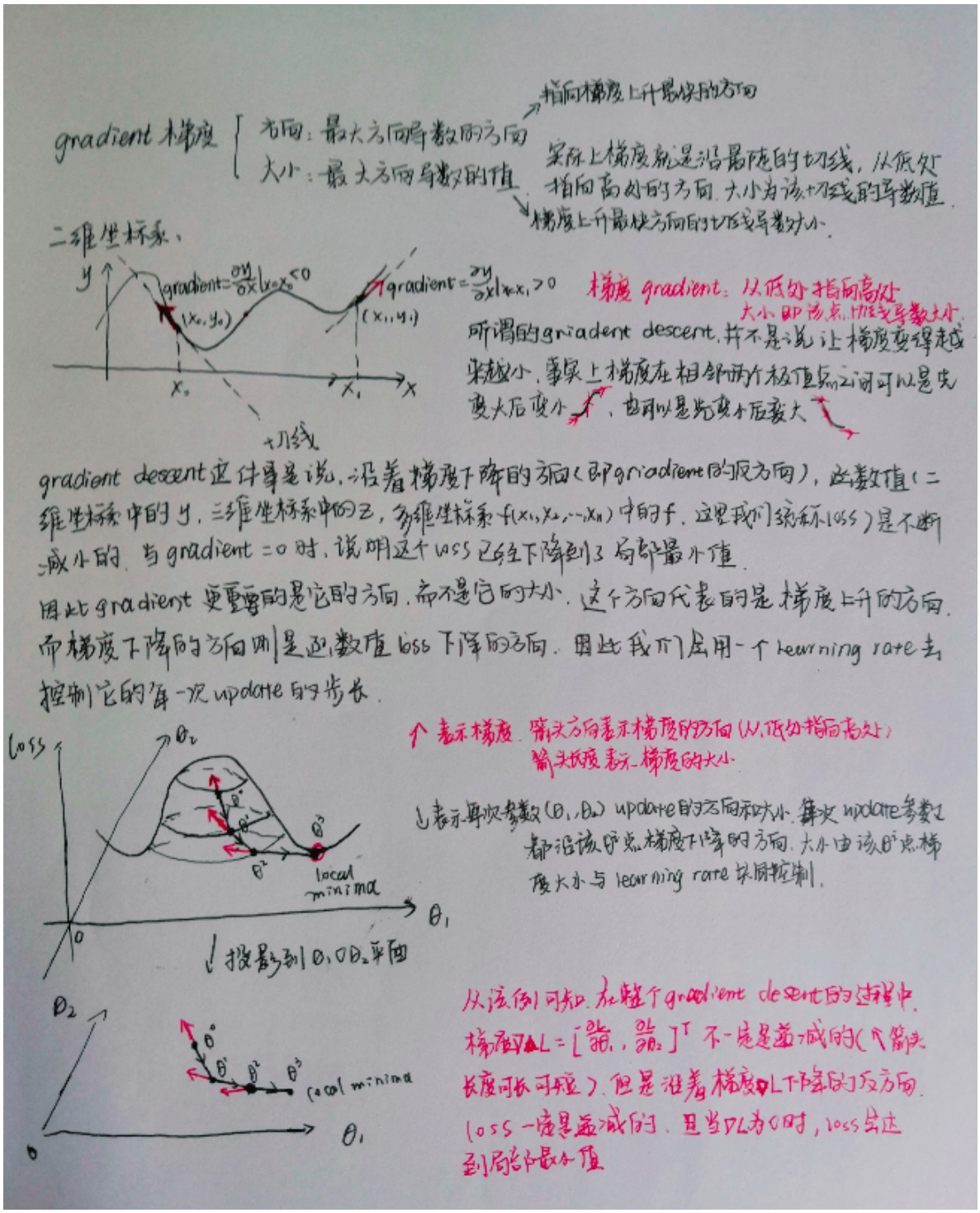
红色箭头是指在(θ_1, θ_2)这点的梯度，梯度方向即箭头方向(从低处指向高处)，梯度大小即箭头长度(表示在θ^i点处最陡的那条切线的导数大小，该方向也是梯度上升最快的方向)

蓝色曲线代表实际情况下参数θ_1和θ_2的更新过程图，每次更新沿着蓝色箭头方向loss会减小，蓝色箭头方向与红色箭头方向刚好相反，代表着梯度下降的方向

因此，在整个gradient descent的过程中，梯度不一定是递减的(红色箭头的长度可以长短不一)，但是沿着梯度下降的方向，函数值loss一定是递减的，且当gradient=0时，loss下降到了局部最小值，总结：梯度下降法指的是函数值loss随梯度下降的方向减小

初始随机在三维坐标系中选取一个点，这个三维坐标系的三个变量分别为$(\theta_1, \theta_2, loss)$，我们的目标是找到最小的那个loss也就是三维坐标系中高度最低的那个点，而gradient梯度可以理解为高度上升最快的那个方向，它的反方向就是梯度下降最快的那个方向，于是每次update沿着梯度反方向，update的步长由梯度大小和learning rate共同决定，当某次update完成后，该点的gradient=0，说明到达了局部最小值

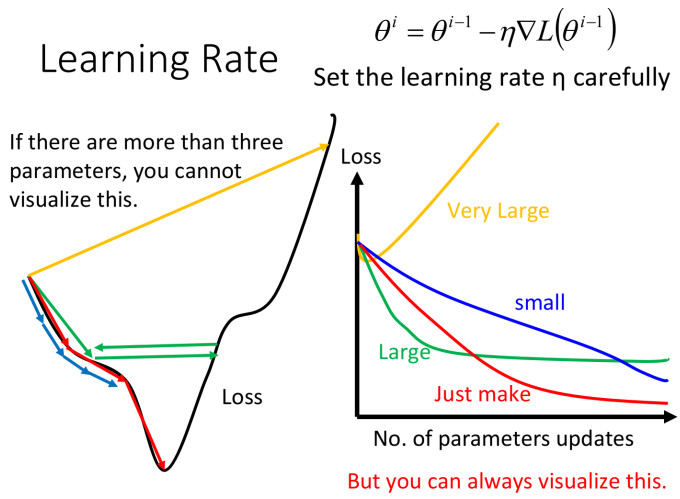
下面是关于gradient descent的一点思考：



Learning rate存在的问题

gradient descent过程中，影响结果的一个很关键的因素就是learning rate的大小

- 如果learning rate刚刚好，就可以像下图中红色线段一样顺利地到达到loss的最小值
- 如果learning rate太小的话，像下图中的蓝色线段，虽然最后能够走到local minimal的地方，但是它可能会走得非常慢，以至于你无法接受
- 如果learning rate太大，像下图中的绿色线段，它的步伐太大了，它永远没有办法走到特别低的地方，可能永远在这个“山谷”的口上振荡而无法走下去
- 如果learning rate非常大，就会像下图中的黄色线段，一瞬间就飞出去了，结果会造成update参数以后，loss反而会越来越大(这一点在上次的demo中有体会到，当lr过大的时候，每次更新loss反而会变大)



当参数有很多个的时候(>3)，其实我们很难做到将loss随每个参数的变化可视化出来(因为最多只能可视化出三维的图像，也就只能可视化三维参数)，但是我们可以把update的次数作为唯一的一个参数，将loss随着update的增加而变化的趋势给可视化出来(上图右半部分)

所以做gradient descent一个很重要的事情是，要把不同的learning rate下，loss随update次数的变化曲线给可视化出来，它可以提醒你该如何调整当前的learning rate的大小，直到出现稳定下降的曲线

Adaptive Learning rates

显然这样手动地去调整learning rates很麻烦，因此我们需要有一些自动调整learning rates的方法

最基本、最简单的大原则是：learning rate通常是随着参数的update越来越小的

因为在起始点的时候，通常是离最低点是比较远的，这时候步伐就要跨大一点；而经过几次update以后，会比较靠近目标，这时候就应该减小learning rate，让它能够收敛在最低点的地方

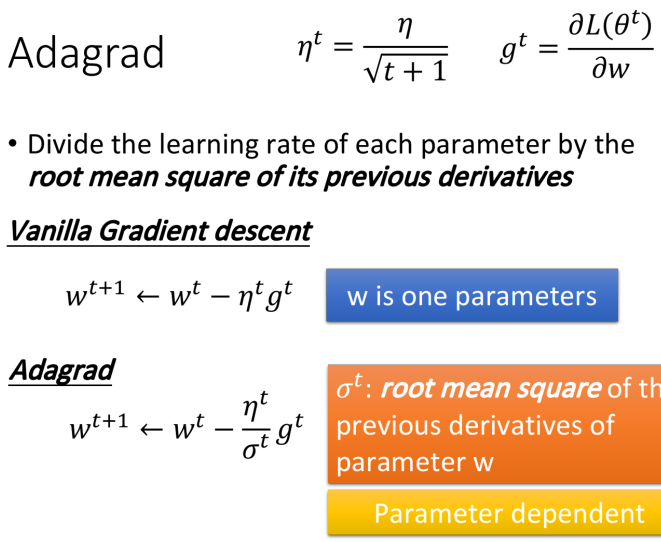
举例：假设到了第t次update，此时$\eta^t = \eta/\sqrt{t+1}$

这种方法使所有参数以同样的方式同样的learning rate进行update，而最好的状况是每个参数都给他不同的learning rate去update

Adagrad

> Divide the learning rate of each parameter by the root mean square(方均根) of its previous derivatives

Adagrad就是将不同参数的learning rate分开考虑的一种算法(adagrad算法update到后面速度会越来越慢，当然这只是adaptive算法中最简单的一种)



这里的w是function中的某个参数，t表示第t次update，g^t表示Loss对w的偏微分，而σ^t是之前所有Loss对w偏微分的方均根(根号下的平方均值)，这个值对每一个参数来说都是不一样的

$$Adagrad$$

$$w^1 = w^0 - \frac{\eta^0}{\sigma^0} \cdot g^0 \quad \sigma^0 = \sqrt{(g^0)^2}$$

$$w^2 = w^1 - \frac{\eta^1}{\sigma^1} \cdot g^1 \quad \sigma^1 = \sqrt{\frac{1}{2}[(g^0)^2 + (g^1)^2]}$$

$$w^3 = w^2 - \frac{\eta 2}{\sigma^2} \cdot g^2 \quad \sigma^2 = \sqrt{\frac{1}{3}[(g^0)^2 + (g^1)^2 + (g^2)^2]}$$

$$\cdots$$

$$w^{t+1} = w^t - \frac{\eta^t}{\sigma^t} \cdot g^t \quad \sigma^t = \sqrt{\frac{1}{1+t}\sum_{i=0}^{t}(g^i)^2}$$

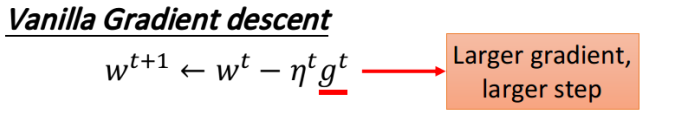
由于η^t和σ^t中都有一个$\sqrt{\frac{1}{1+t}}$的因子，两者相消，即可得到adagrad的最终表达式：

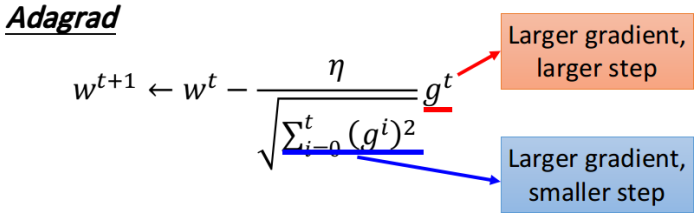
$$w^{t+1} = w^t - \frac{\eta}{\sum_{i=0}^{t}(g^i)^2} \cdot g^t$$

Adagrad的contradiction解释

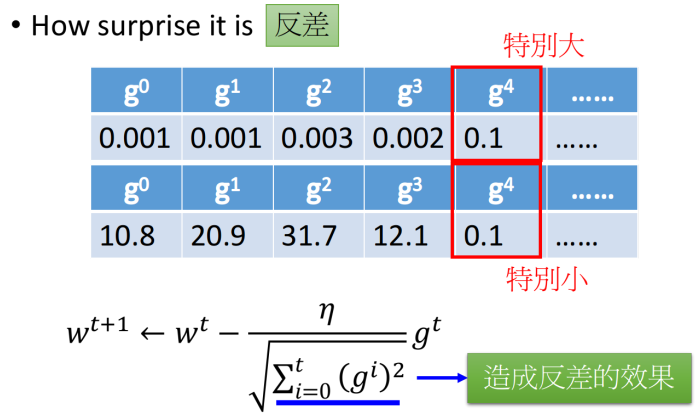
Adagrad的表达式$w^{t+1} = w^t - \frac{\eta}{\sum_{i=0}^{t}(g^i)^2} \cdot g^t$里面有一件很矛盾的事情：

我们在做gradient descent的时候，希望的是当梯度值即微分值g^t越大的时候(此时斜率越大，还没有接近最低点)更新的步伐要更大一些，但是Adagrad的表达式中，分母表示梯度越大步伐越大，分子却表示梯度越大步伐越小，两者似乎相互矛盾



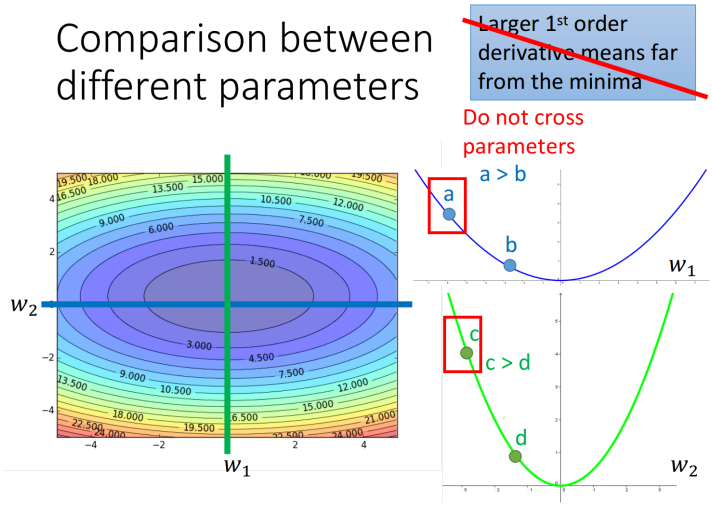


在一些paper里是这样解释的：Adagrad要考虑的是，这个gradient有多surprise，即反差有多大，假设t=4的时候g^4与前面的gradient反差特别大，那么g^t与$\sqrt{\frac{1}{t+1}\sum_{i=0}^{t}(g^i)^2}$之间的大小反差就会比较大，它们的商就会把这一反差效果体现出来



gradient越大，离最低点越远这件事情在有多个参数的情况下是不一定成立的

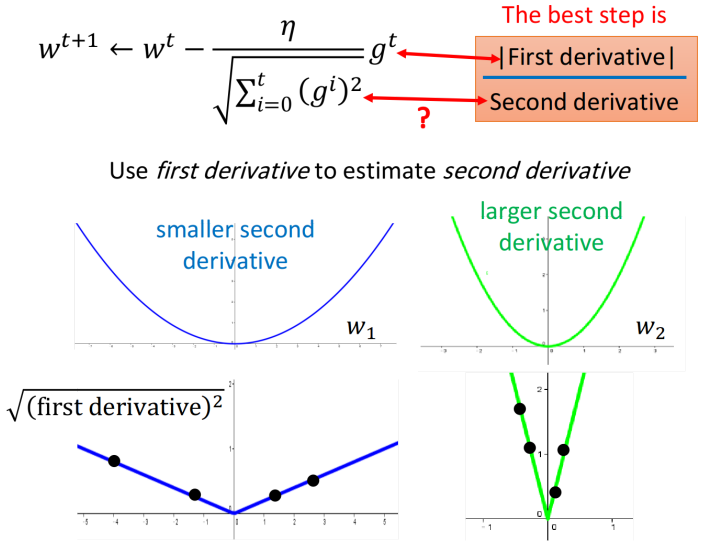
如下图所示，w1和w2分别是loss function的两个参数，loss的值投影到该平面中以颜色深度表示大小，分别在w2和w1处垂直切一刀(这样就只有另一个参数的gradient会变化)，对应的情况为右边的两条曲线，可以看出，比起a点，c点距离最低点更近，但是它的gradient却越大



实际上，对于一个二次函数$y = ax^2 + bx + c$来说，最小值点的$x = -\frac{b}{2a}$，而对于任意一点x_0，它迈出最好的步伐长度是$|x_0 + \frac{b}{2a}| = |\frac{2ax_0+b}{2a}|$(这样就一步迈到最小值点了)，联系该函数的一阶和二阶导数$y' = 2ax + b$、$y'' = 2a$，可以发现the best step is $|\frac{y'}{y''}|$，也就是说他不仅跟一阶导数(gradient)有关，还跟二阶导师有关，因此我们可以通过这种方法重新比较上面的a和c点，就可以得到比较正确的答案

再来回顾Adagrad的表达式：$w^{t+1} = w^t - \frac{\eta}{\sum_{i=0}^{t}(g^i)^2} \cdot g^t$

g^t就是一次微分，而分母中的$\sum_{i=0}^{t}(g^i)^2$反映了二次微分的大小，所以Adagrad想要做的事情就是，在不增加任何额外运算的前提下，想办法去估测二次微分的值



Stochastic Gradicent Descent

随机梯度下降的方法可以让训练更快速，传统的gradient descent的思路是看完所有的样本点之后再构建loss function，然后去update参数；而stochastic gradient descent的做法是，看到一个样本点就update一次，因此它的loss function不是所有样本点的error平方和，而是这个随机样本点的error平方

Stochastic Gradient Descent

$$L = \sum_n \left(\hat{y}^n - \left(b + \sum w_i x_i^n\right)\right)^2$$

Loss is the summation over all training examples

◆ ***Gradient Descent*** $\theta^i = \theta^{i-1} - \eta \nabla L\left(\theta^{i-1}\right)$

◆ ***Stochastic Gradient Descent*** Faster!

Pick an example x^n

$$L^n = \left(\hat{y}^n - \left(b + \sum w_i x_i^n\right)\right)^2 \qquad \theta^i = \theta^{i-1} - \eta \nabla L^n\left(\theta^{i-1}\right)$$

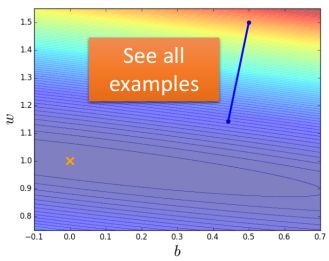
Loss for only one example

stochastic gradient descent与传统gradient descent的效果对比如下：

Stochastic Gradient Descent

Gradient Descent

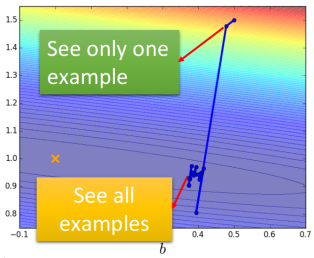
Update after seeing all examples



Stochastic Gradient Descent

Update for each example

If there are 20 examples, 20 times faster.



Feature Scaling

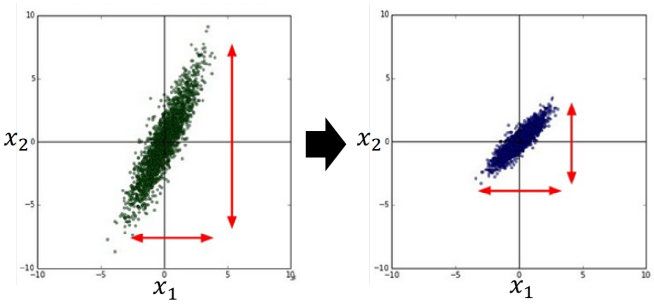
概念介绍

特征缩放，当多个特征的分布范围很不一样时，最好将这些不同feature的范围缩放成一样

Feature Scaling

Source of figure: http://cs231n.github.io/neural-networks-2/

$$y = b + w_1 x_1 + w_2 x_2$$

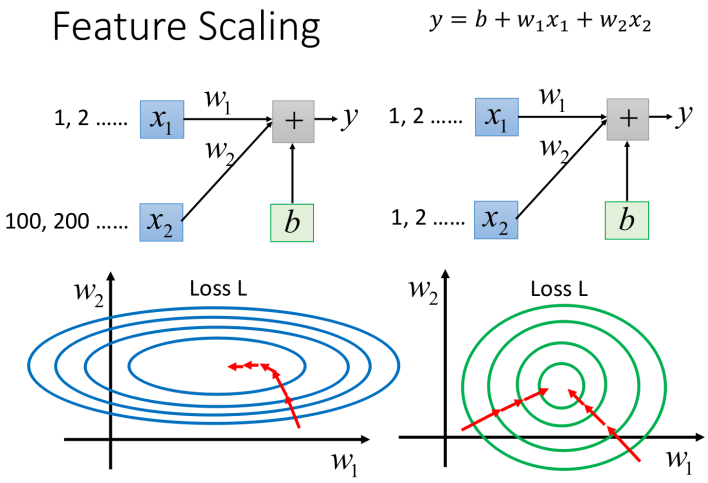


Make different features have the same scaling

原理解释

$y = b + w_1 x_1 + w_2 x_2$，假设x1的值都是很小的，比如1,2...；x2的值都是很大的，比如100,200...

此时去画出loss的error surface，如果对w1和w2都做一个同样的变动Δw，那么w1的变化对y的影响是比较小的，而w2的变化对y的影响是比较大的



左边的error surface表示，w1对y的影响比较小，所以w1对loss是有比较小的偏微分的，因此在w1的方向上图像是比较平滑的；w2对y的影响比较大，所以w2对loss的影响比较大，因此在w2的方向上图像是比较sharp的

如果x1和x2的值，它们的scale是接近的，那么w1和w2对loss就会有差不多的影响力，loss的图像接近于圆形，那这样做对gradient descent有什么好处呢？

对gradient decent的帮助

之前我们做的demo已经表明了，对于这种长椭圆形的error surface，如果不使用Adagrad之类的方法，是很难搞定它的，因为在像w1和w2这样不同的参数方向上，会需要不同的learning rate，用相同的lr很难达到最低点

如果有scale的话，loss在参数w1、w2平面上的投影就是一个正圆形，update参数会比较容易

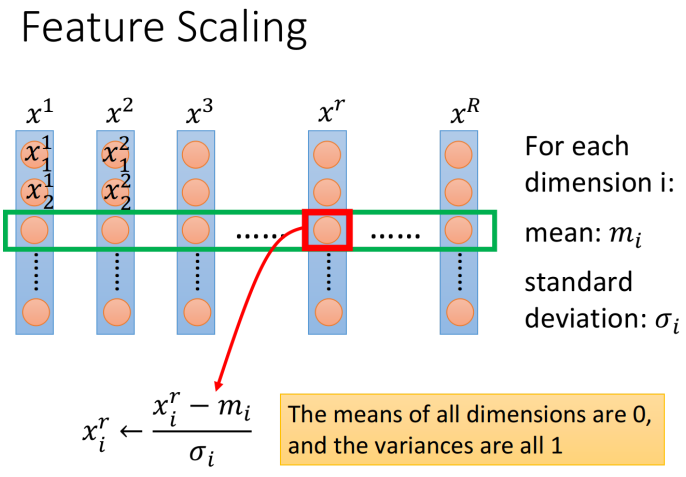
而且gradient descent的每次update并不都是向着最低点走的，每次update的方向是顺着等高线的方向(梯度gradient下降的方向)，而不是径直走向最低点；但是当经过对input的scale使loss的投影是一个正圆的话，不管在这个区域的哪一个点，它都会向着圆心走。因此feature scaling对参数update的效率是有帮助的

如何做feature scaling

假设有R个example(上标i表示第i个样本点)，$x^1, x^2, x^3, \ldots, x^r, \ldots x^R$，每一笔example，它里面都有一组feature(下标j表示该样本点的第j个特征)

对每一个demension i，都去算出它的平均值mean=m_i，以及标准差standard deviation=σ_i

对第r个example的第i个component，减掉均值，除以标准差，即$x_i^r = \frac{x_i^r - m_i}{\sigma_i}$



说了那么多，实际上就是将每一个参数都归一化成标准正态分布，即$f(x_i)=\frac{1}{\sqrt{2\pi}}e^{-\frac{x_i^2}{2}}$，其中$x_i$表示第i个参数

Gradient Descent的理论基础

Taylor Series

泰勒表达式：$h(x)=\sum_{k=0}^{\infty}\frac{h^{(k)}(x_0)}{k!}(x-x_0)^k=h(x_0)+h'(x_0)(x-x_0)+\frac{h''(x_0)}{2!}(x-x_0)^2+\dots$

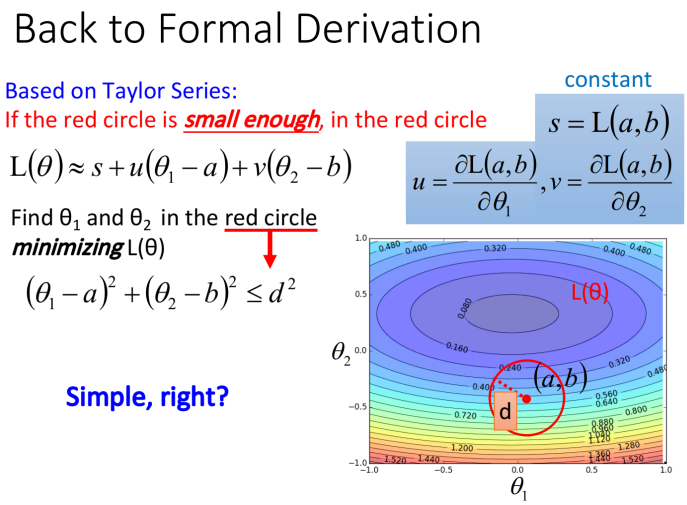
When x is close to x_0：$h(x)\approx h(x_0)+h'(x_0)(x-x_0)$

同理，对于二元函数，when x and y is close to x_0 and y_0：

$h(x,y)\approx h(x_0,y_0)+\frac{\partial h(x_0,y_0)}{\partial x}(x-x_0)+\frac{\partial h(x_0,y_0)}{\partial y}(y-y_0)$

从泰勒展开式推导出gradient descent

对于loss图像上的某一个点(a,b)，如果我们想要找这个点附近loss最小的点，就可以用泰勒展开的思想



假设用一个red circle限定点的范围，这个圆足够小以满足泰勒展开的精度，那么此时我们的loss function就可以化简为：

$L(\theta)\approx L(a,b)+\frac{\partial L(a,b)}{\partial\theta_1}(\theta_1-a)+\frac{\partial L(a,b)}{\partial\theta_2}(\theta_2-b)$

令$s=L(a,b)$，$u=\frac{\partial L(a,b)}{\partial\theta_1}$，$v=\frac{\partial L(a,b)}{\partial\theta_2}$

则$L(\theta)\approx s+u\cdot(\theta_1-a)+v\cdot(\theta_2-b)$

假定red circle的半径为d，则有限制条件：$(\theta_1-a)^2+(\theta_2-b)^2\le d^2$

此时去求$L(\theta)_{min}$，这里有个小技巧，把$L(\theta)$转化为两个向量的乘积：
$u\cdot(\theta_1-a)+v\cdot(\theta_2-b)=(u,v)\cdot(\theta_1-a,\theta_2-b)=(u,v)\cdot(\Delta\theta_1,\Delta\theta_2)$

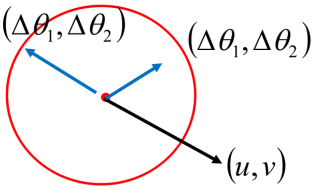
观察图形可知，当向量(θ_1-a,θ_2-b)与向量(u,v)反向，且刚好到达red circle的边缘时(用η去控制向量的长度)，$L(\theta)$最小

Gradient descent – two variables

Red Circle: (If the radius is small)

$$L(\theta) \approx s + u(\theta_1 - a) + v(\theta_2 - b)$$

$$\Delta\theta_1 \qquad \Delta\theta_2$$



Find θ_1 and θ_2 in the red circle

minimizing L(θ)

$$(\theta_1 - a)^2 + (\theta_2 - b)^2 \le d^2$$

$$\Delta\theta_1 \qquad \Delta\theta_2$$

To minimize L(θ)

$$\begin{bmatrix} \Delta\theta_1 \\ \Delta\theta_2 \end{bmatrix} = -\eta \begin{bmatrix} u \\ v \end{bmatrix} \Rightarrow \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} - \eta \begin{bmatrix} u \\ v \end{bmatrix}$$

$(\theta_1 - a, \theta_2 - b)$实际上就是$(\Delta\theta_1, \Delta\theta_2)$，于是$L(\theta)$局部最小值对应的参数为中心点减去gradient的加权

$$\begin{bmatrix} \Delta\theta_1 \\ \Delta\theta_2 \end{bmatrix} = -\eta \begin{bmatrix} u \\ v \end{bmatrix} => \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} - \eta \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial L(a,b)}{\partial \theta_1} \\ \frac{\partial L(a,b)}{\partial \theta_2} \end{bmatrix}$$

这就是gradient descent在数学上的推导，注意它的重要前提是，给定的那个红色圈圈的范围要足够小，这样泰勒展开给我们的近似才会更精确，而η的值是与圆的半径成正比的，因此理论上learning rate要无穷小才能够保证每次gradient descent在update参数之后的loss会越来越小，于是当learning rate没有设置好，泰勒近似不成立，就有可能使gradient descent过程中的loss没有越来越小

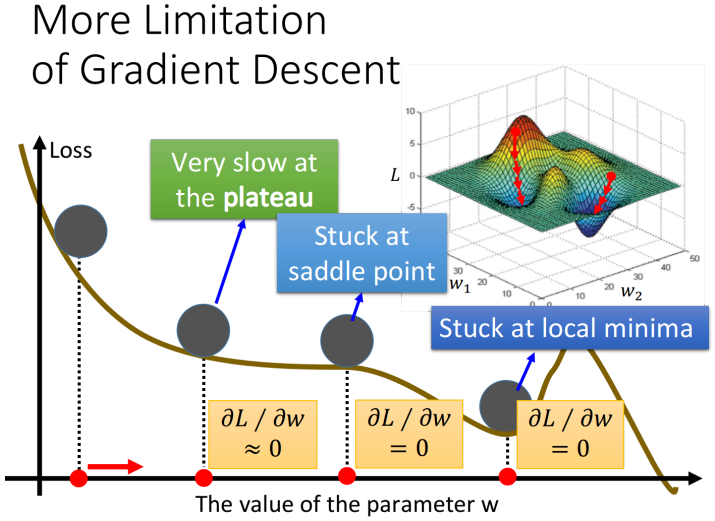
当然泰勒展开可以使用二阶、三阶乃至更高阶的展开，但这样会使得运算量大大增加，反而降低了运行效率

Gradient Descent的限制

之前已经讨论过，gradient descent有一个问题是它会停在local minima的地方就停止update了

事实上还有一个问题是，微分值是0的地方并不是只有local minima，settle point的微分值也是0

以上都是理论上的探讨，到了实践的时候，其实当gradient的值接近于0的时候，我们就已经把它停下来了，但是微分值很小，不见得就是很接近local minima，也有可能像下图一样在一个高原的地方



综上，**gradient descent的限制是，它在gradient即微分值接近于0的地方就会停下来，而这个地方不一定是global minima，它可能是local minima，可能是saddle point鞍点，甚至可能是一个loss很高的plateau平缓高原**