Gradient Descent

Review

前面預測寶可夢cp值的例子裡，已經初步介紹了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(上標表示第幾組參數，下標表示這組參數中的第幾個參數)

假設θ是參數的集合：Suppose that θ has two variables $\{\theta_1, \theta_2\}$

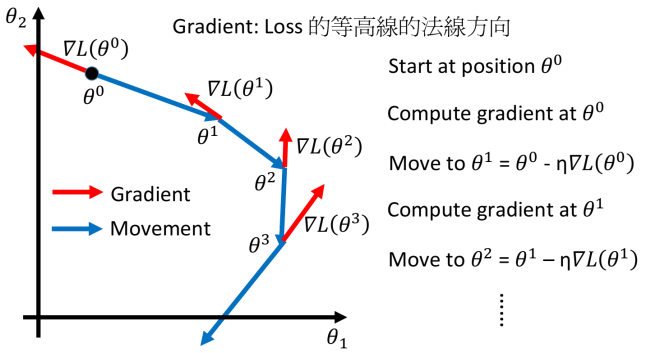
隨機選取一組起始的參數：Randomly start at $\theta^0 = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix}$

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

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

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

下圖是將gradient descent在投影到二維坐標系中可視化的樣子，圖上的每一個點都是$(\theta_1, \theta_2, loss)$在該平面的投影



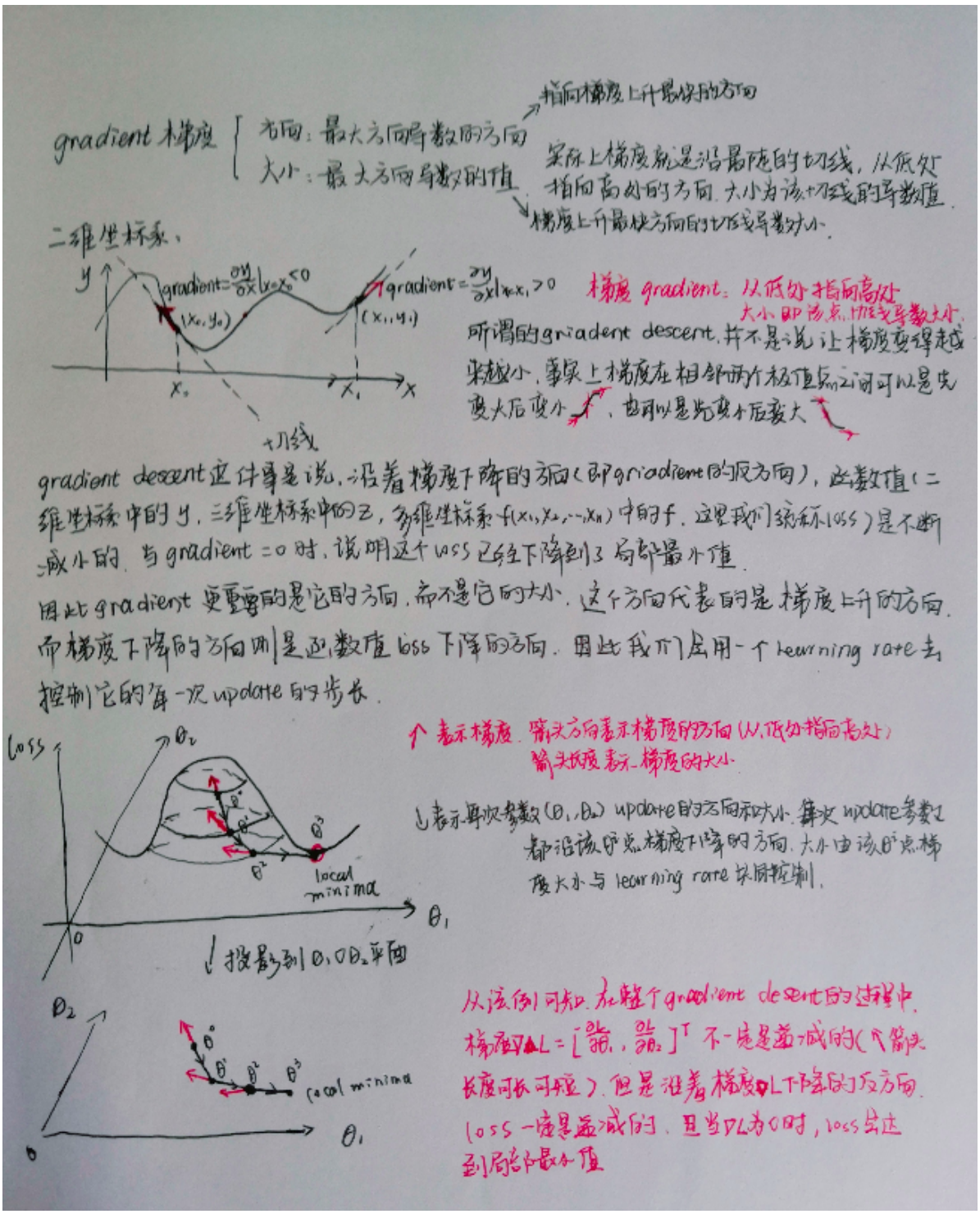
紅色箭頭是指在(θ_1, θ_2)這點的梯度，梯度方向即箭頭方向(從低處指向高處)，梯度大小即箭頭長度(表示在θ^i點處最陡的那條切線的導數大小，該方向也是梯度上升最快的方向)

藍色曲線代表實際情況下參數θ_1和θ_2的更新過程圖，每次更新沿著藍色箭頭方向loss會減小，藍色箭頭方向與紅色箭頭方向剛好相反，代表著梯度下降的方向

因此，在整個gradient descent的過程中，梯度不一定是遞減的(紅色箭頭的長度可以長短不一)，但是沿著梯度下降的方向，函數值loss一定是遞減的，且當gradient=0時，loss下降到了局部最小值，總結：梯度下降法指的是函數值loss隨梯度下降的方向減小

初始隨機在三維坐標系中選取一個點，這個三維坐標系的三個變量分別為$(\theta_1, \theta_2, loss)$，我們的目標是找到最小的那個loss也就是三維坐標系中高度最低的那個點，而gradient梯度可以理解為高度上升最快的那個方向，它的反方向就是梯度下降最快的那個方向，於是每次update沿著梯度反方向，update的步長由梯度大小和learning rate共同決定，當某次update完成後，該點的gradient=0，說明到達了局部最小值

下面是關於gradient descent的一點思考：

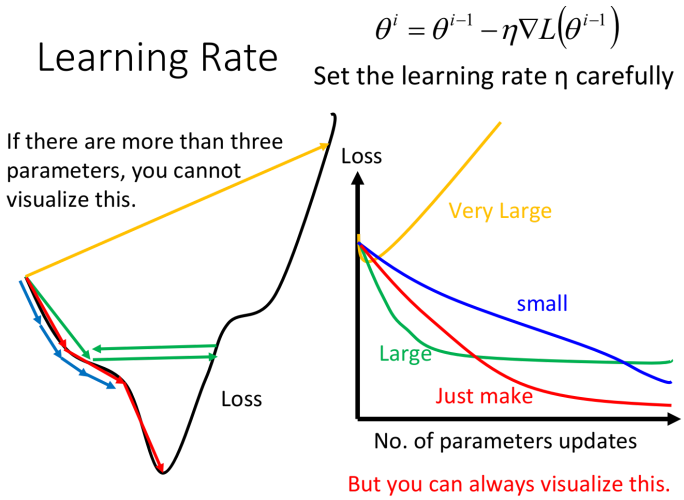


Learning rate存在的問題

gradient descent過程中，影響結果的一個很關鍵的因素就是learning rate的大小

- 如果learning rate剛剛好，就可以像下圖中紅色線段一樣順利地到達到loss的最小值

- 如果learning rate太小的話，像下圖中的藍色線段，雖然最後能夠走到local minimal的地方，但是它可能會走得非常慢，以至於你無法接受
- 如果learning rate太大，像下圖中的綠色線段，它的步伐太大了，它永遠沒有辦法走到特別低的地方，可能永遠在這個"山谷"的口上振盪而無法走下去
- 如果learning rate非常大，就會像下圖中的黃色線段，一瞬間就飛出去了，結果會造成update參數以後，loss反而會越來越大(這一點在上次的demo中有體會到，當lr過大的時候，每次更新loss反而會變大)



當參數有很多個的時候(>3)，其實我們很難做到將loss隨每個參數的變化可視化出來(因為最多只能可視化出三維的圖像，也就只能可視化三維參數)，但是我們可以把update的次數作為唯一的一個參數，將loss隨著update的增加而變化的趨勢給可視化出來(上圖右半部分)

所以做gradient descent一個很重要的事情是，要把不同的learning rate下，loss隨update次數的變化曲線給可視化出來，它可以提醒你該如何調整當前的learning rate的大小，直到出現穩定下降的曲線

Adaptive Learning rates

顯然這樣手動地去調整learning rates很麻煩，因此我們需要有一些自動調整learning rates的方法

最基本、最簡單的大原則是：learning rate通常是隨著參數的update越來越小的

因為在起始點的時候，通常是離最低點是比較遠的，這時候步伐就要跨大一點；而經過幾次update以後，會比較靠近目標，這時候就應該減小learning rate，讓它能夠收斂在最低點的地方

舉例：假設到了第t次update，此時$\eta^t = \eta/\sqrt{t+1}$

這種方法使所有參數以同樣的方式同樣的learning rate進行update，而最好的狀況是每個參數都給他不同的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就是將不同參數的learning rate分開考慮的一種算法(adagrad算法update到後面速度會越來越慢，當然這只是adaptive算法中最簡單的一種)

Adagrad $\eta^t = \frac{\eta}{\sqrt{t+1}}$ $g^t = \frac{\partial L(\theta^t)}{\partial w}$

- Divide the learning rate of each parameter by the ***root mean square of its previous derivatives***

Vanilla Gradient descent

$$w^{t+1} \leftarrow w^t - \eta^t g^t$$

w is one parameters

Adagrad

$$w^{t+1} \leftarrow w^t - \frac{\eta^t}{\sigma^t} g^t$$

σ^t: ***root mean square*** of the previous derivatives of parameter w

Parameter dependent

這裡的w是function中的某個參數，t表示第t次update，g^t表示Loss對w的偏微分，而σ^t是之前所有Loss對w偏微分的方均根(根號下的平方均值)，這個值對每一個參數來說都是不一樣的

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

由於η^t和σ^t中都有一個$\sqrt{\frac{1}{1+t}}$的因子，兩者相消，即可得到adagrad的最終表達式：

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

Adagrad的contradiction解釋

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

我們在做gradient descent的時候，希望的是當梯度值即微分值g^t越大的時候(此時斜率越大，還沒有接近最低點)更新的步伐要更大一些，但是Adagrad的表達式中，分母表示梯度越大步伐越小，分子卻表示梯度越大步伐越大，兩者似乎相互矛盾

Vanilla Gradient descent

$$w^{t+1} \leftarrow w^t - \eta^t g^t$$

Larger gradient, larger step

Adagrad

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

Larger gradient, larger step

Larger gradient, smaller step

在一些paper裡是這樣解釋的：Adagrad要考慮的是，這個gradient有多surprise，即反差有多大，假設t=4的時候g^4與前面的gradient反差特別大，那麼g^t與$\sqrt{\frac{1}{t+1}\sum_{i=0}^{t}(g^i)^2}$之間的大小反差就會比較大，它們的商就會把這一反差效果體現出來

- How surprise it is 反差

特別大

g^0	g^1	g^2	g^3	g^4	……
0.001	0.001	0.003	0.002	0.1	……
g^0	g^1	g^2	g^3	g^4	……
10.8	20.9	31.7	12.1	0.1	……

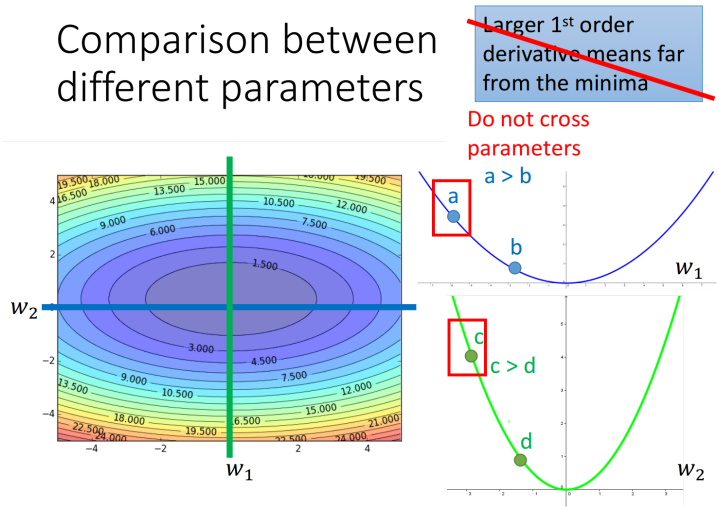
特別小

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

造成反差的效果

gradient越大，離最低點越遠這件事情在有多個參數的情況下是不一定成立的

如下圖所示，w1和w2分別是loss function的兩個參數，loss的值投影到該平面中以顏色深度表示大小，分別在w2和w1處垂直切一刀(這樣就只有另一個參數的gradient會變化)，對應的情況為右邊的兩條曲線，可以看出，比起a點，c點距離最低點更近，但是它的gradient卻越大

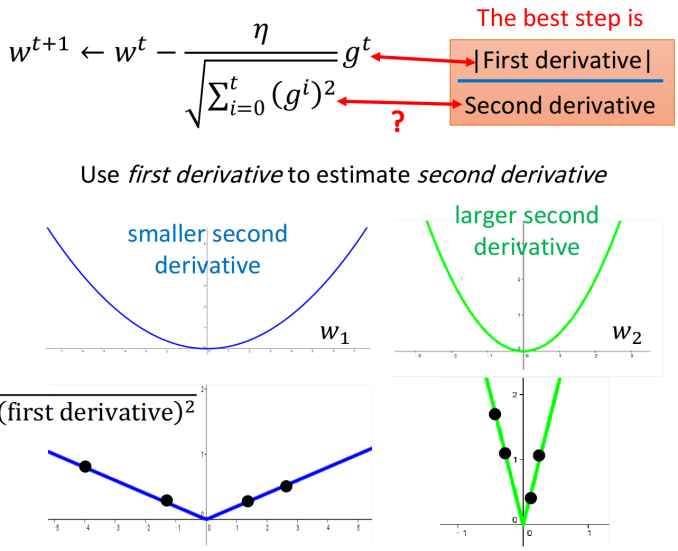


實際上，對於一個二次函數$y = ax^2 + bx + c$來說，最小值點的$x = -\frac{b}{2a}$，而對於任意一點x_0，它邁出最好的步伐長度是$|x_0 + \frac{b}{2a}| = |\frac{2ax_0+b}{2a}|$(這樣就一步邁到最小值點了)，聯繫該函數的一階和二階導數$y' = 2ax + b$、$y'' = 2a$，可以發現the best step is $|\frac{y'}{y''}|$，也就是說他不僅跟一階導數(gradient)有關，還跟二階導師有關，因此我們可以通過這種方法重新比較上

面的a和c點，就可以得到比較正確的答案

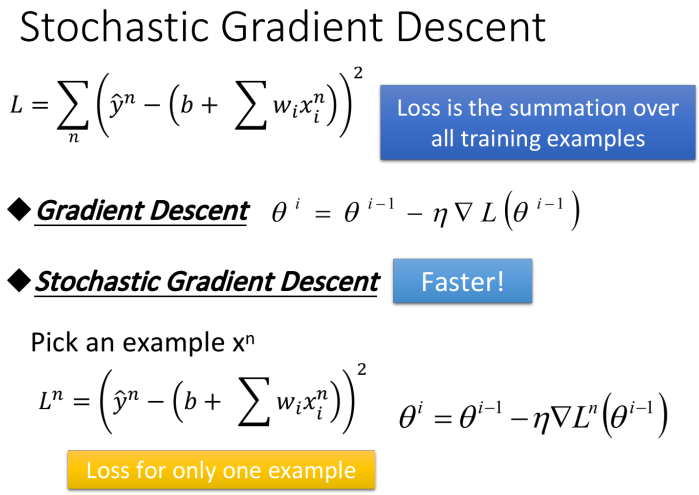
再來回顧Adagrad的表達式：$w^{t+1} = w^t - \frac{\eta}{\sum_{i=0}^{t}(g^i)^2} \cdot g^t$

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，在不增加任何額外運算的前提下，想辦法去估測二次微分的值

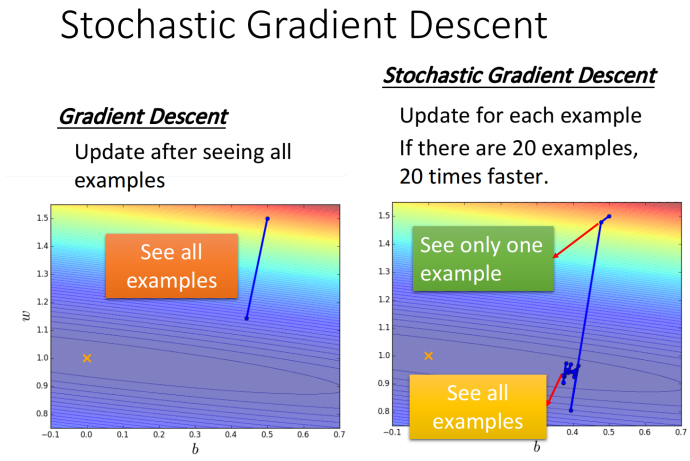


Stochastic Gradicent Descent

隨機梯度下降的方法可以讓訓練更快速，傳統的gradient descent的思路是看完所有的樣本點之後再構建loss function，然後去update參數；而stochastic gradient descent的做法是，看到一個樣本點就update一次，因此它的loss function不是所有樣本點的error平方和，而是這個隨機樣本點的error平方



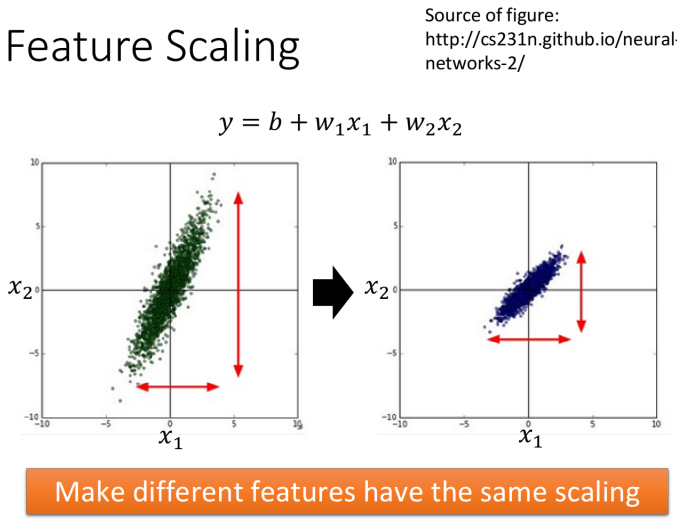
stochastic gradient descent與傳統gradient descent的效果對比如下：



Feature Scaling

概念介紹

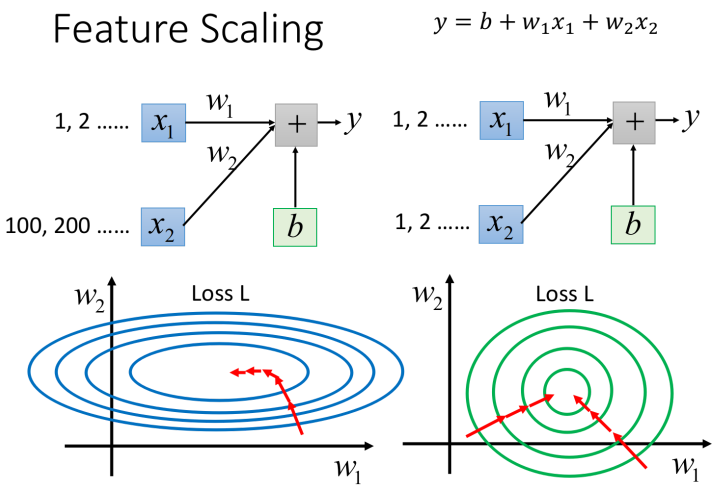
特徵縮放，當多個特徵的分佈範圍很不一樣時，最好將這些不同feature的範圍縮放成一樣



原理解釋

$y = b + w_1x_1 + w_2x_2$，假設x1的值都是很小的，比如1,2...；x2的值都是很大的，比如100,200...

此時去畫出loss的error surface，如果對w1和w2都做一個同樣的變動Δw，那麼w1的變化對y的影響是比較小的，而w2的變化對y的影響是比較大的



左邊的error surface表示，w1對y的影響比較小，所以w1對loss是有比較小的偏微分的，因此在w1的方向上圖像是比較平滑的；w2對y的影響比較大，所以w2對loss的影響比較大，因此在w2的方向上圖像是比較sharp的

如果x1和x2的值，它們的scale是接近的，那麼w1和w2對loss就會有差不多的影響力，loss的圖像接近於圓形，那這樣做對gradient descent有什麼好處呢？

對gradient decent的幫助

之前我們做的demo已經表明了，對於這種長橢圓形的error surface，如果不使用Adagrad之類的方法，是很難搞定它的，因為在像w1和w2這樣不同的參數方向上，會需要不同的learning rate，用相同的lr很難達到最低點

如果有scale的話，loss在參數w1、w2平面上的投影就是一個正圓形，update參數會比較容易

而且gradient descent的每次update並不都是向著最低點走的，每次update的方向是順著等高線的方向(梯度gradient下降的方向)，而不是徑直走向最低點；但是當經過對input的scale使loss的投影是一個正圓的話，不管在這個區域的哪一個點，它都會向著圓心走。因此feature scaling對參數update的效率是有幫助的

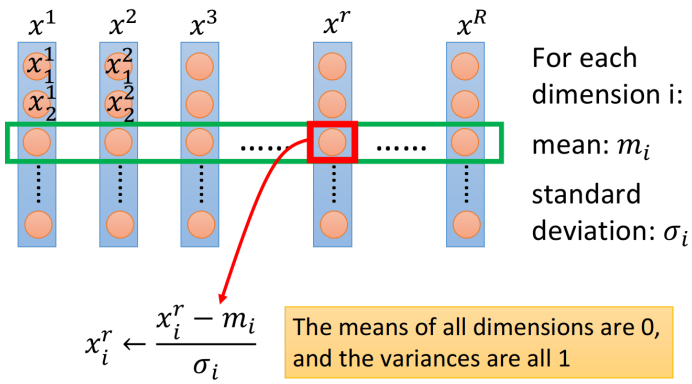
如何做feature scaling

假設有R個example(上標i表示第i個樣本點)，$x^1, x^2, x^3, \ldots, x^r, \ldots x^R$，每一筆example，它裡面都有一組feature(下標j表示該樣本點的第j個特徵)

對每一個demension i，都去算出它的平均值mean=m_i，以及標準差standard deviation=σ_i

對第r個example的第i個component，減掉均值，除以標準差，即$x_i^r \leftarrow \frac{x_i^r - m_i}{\sigma_i}$

Feature Scaling



說了那麼多，實際上就是將每一個參數都歸一化成標準正態分佈，即$f(x_i) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x_i^2}{2}}$，其中$x_i$表示第i個參數

Gradient Descent的理論基礎

Taylor Series

泰勒表達式：

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

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

同理，對於二元函數，when x and y is close to x_0 and y_0：

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

從泰勒展開式推導出gradient descent

對於loss圖像上的某一個點(a,b)，如果我們想要找這個點附近loss最小的點，就可以用泰勒展開的思想

Back to Formal Derivation

Based on Taylor Series:
If the red circle is ***small enough***, in the red circle

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

constant

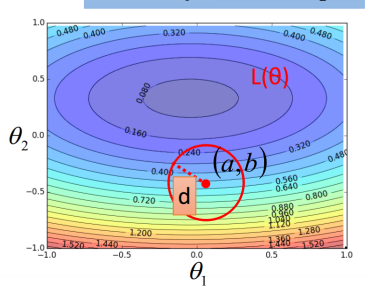
$$s = L(a,b)$$

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

Find θ_1 and θ_2 in the red circle
minimizing L(θ)

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

Simple, right?



假設用一個red circle限定點的範圍，這個圓足夠小以滿足泰勒展開的精度，那麼此時我們的loss function就可以化簡為：

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

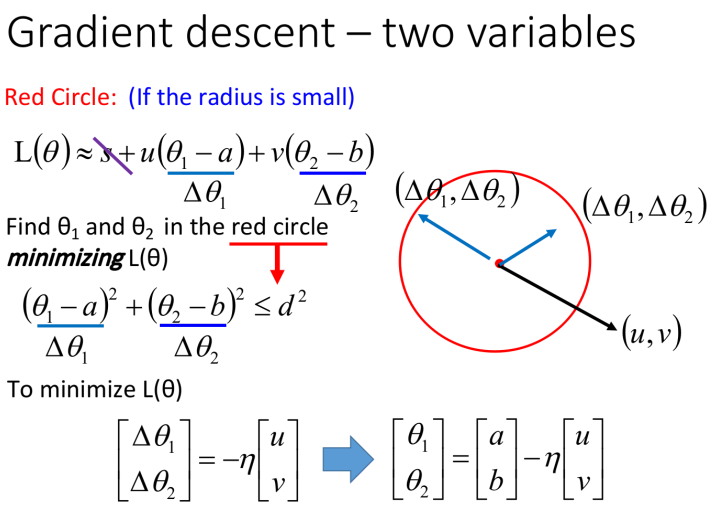
令$s = L(a,b)$，$u = \frac{\partial L(a,b)}{\partial \theta_1}$，$v = \frac{\partial L(a,b)}{\partial \theta_2}$

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

假定red circle的半徑為d，則有限制條件：$(\theta_1 - a)^2 + (\theta_2 - b)^2 \leq d^2$

此時去求$L(\theta)_{min}$，這裡有個小技巧，把$L(\theta)$轉化為兩個向量的乘積：
$u \cdot (\theta_1 - a) + v \cdot (\theta_2 - b) = (u,v) \cdot (\theta_1 - a, \theta_2 - b) = (u,v) \cdot (\Delta\theta_1, \Delta\theta_2)$

觀察圖形可知，當向量$(\theta_1 - a, \theta_2 - b)$與向量$(u,v)$反向，且剛好到達red circle的邊緣時(用$\eta$去控制向量的長度)，$L(\theta)$最小



$(\theta_1 - a, \theta_2 - b)$實際上就是$(\Delta\theta_1, \Delta\theta_2)$，於是$L(\theta)$局部最小值對應的參數為中心點減去gradient的加權

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

這就是gradient descent在數學上的推導，注意它的重要前提是，給定的那個紅色圈圈的範圍要足夠小，這樣泰勒展開給我們的近似才會更精確，而η的值是與圓的半徑成正比的，因此理論上learning rate要無窮小才能夠保證每次gradient descent在update參數之後的loss會越來越小，於是當learning rate沒有設置好，泰勒近似不成立，就有可能使gradient descent過程中的loss沒有越來越小

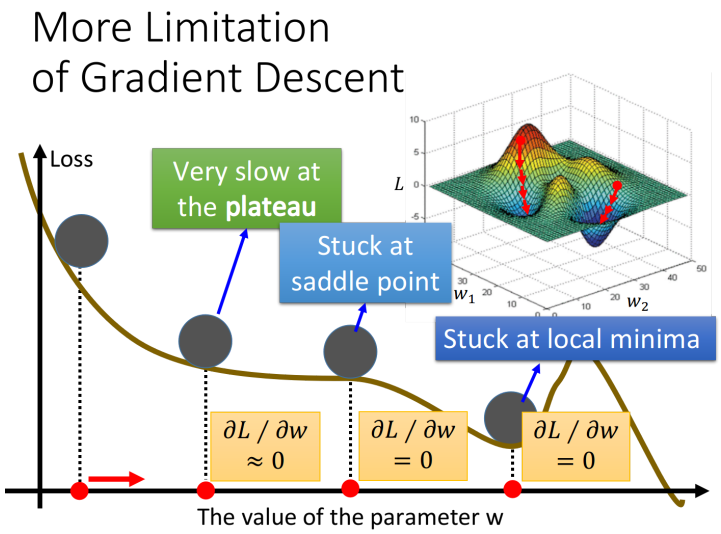
當然泰勒展開可以使用二階、三階乃至更高階的展開，但這樣會使得運算量大大增加，反而降低了運行效率

Gradient Descent的限制

之前已經討論過，gradient descent有一個問題是它會停在local minima的地方就停止update了

事實上還有一個問題是，微分值是0的地方並不是只有local minima，settle point的微分值也是0

以上都是理論上的探討，到了實踐的時候，其實當gradient的值接近於0的時候，我們就已經把它停下來了，但是微分值很小，不見得就是很接近local minima，也有可能像下圖一樣在一個高原的地方



綜上，**gradient descent的限制是，它在gradient即微分值接近於0的地方就會停下來，而這個地方不一定是global minima，它可能是local minima，可能是saddle point鞍點，甚至可能是一個loss很高的plateau平緩高原**