ML Lecture 3-1: Gradient Descent

> 臺灣大學人工智慧中心 科技部人工智慧技術暨全幅健康照護聯合研究中心 http://ai.ntu.edu.tw

Review

- **使用時機**： Machine Lraening 的第二步驟，我們定義了 loss function, L；而到第三步驟， 我們希望找到一個參數 θ，最小化 loss function 的 output，這時我們就可以用 Gradient Descent
- **實作**：
  - 假設 θ 有兩個參數，記成 $\{\theta_1, \theta_2\}$
  - 首先，隨機選一組初始值， $\theta^0 = \{\theta_1^0, \theta_2^0\}$
  - 再來計算下一個時間點的參數 θ^1 ，等於 θ^0 減掉 learning rate 乘上 θ^0 對 loss function 的偏微分，即 $\theta^1 = \theta^0 - \eta\nabla L(\theta^0)$
  - 不斷反覆進行上一步驟，這就是 Gradient Descent

- **視覺化**：
  - 將上述步驟視覺化的呈現，即在圖中隨機選一初始位置 θ^0 ，計算這個參數對 L 的 Gradient，及紅色箭頭
  - 再把 Gradient 乘上 η，取負號，就是藍色的箭頭，再加上 θ^0 ，就得到 θ^1
  - 上述步驟反覆進行下去，就是 Gradient Descent

Tips 1: Tuning your learning rate

- **視覺化「參數的變化 v.s. loss 的變化」**
  - learning rate 調**太小** (**藍**)：速度太慢，但終究會走到 local minimum
  - learning rate 調**太大** (**綠**)：步伐太大，永遠走不到 local minimum 的地方
  - **合適**的 learning rate (**紅**)：順利的走到 local minimum

    所以，在做 Gradient Descent 前，可以先視覺化「參數的變化 v.s. loss 的變化」，如右下圖，避免結果爛掉

- **調整 learning rate 的原則**
  - 隨著參數 update，learning rate 應該越來越小

    舉例：learning rate η 是 time dependent 的參數，$\eta^t = \eta/\sqrt{t+1}$
  - 每個不同的參數都應該給不一樣的 learning rate，需要因材施教

- **Adagrad**：調整 learning rate 一個容易實作的方法
  - 作法：將每一個參數的 learning rate，除上之前算出微分值的 root mean square
  - 舉例： w 是某一個參數，原本做 Gradient Descent 的時候，只 depend on 時間的值 $\Rightarrow$ $w^{t+1} \leftarrow w^t - \eta^t g^t$ 而 Adagrad 中，會把 η^t 再除以 σ^t (過去所有微分值的 root mean

square)，這個值對每一個參數都是不一樣的； 故，不同的參數， learning rate 都是不一樣的。

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

- 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

- 實作（如下圖所示） 初始參數 w^0、g^0，用 Adagrad 反覆 update 到第 t+1 次可以推導出，第 t+1 次，$w^{t+1} \leftarrow w^t - \frac{\eta^t}{\sigma^t} g^t$ 、$\sigma^t = \sqrt{\frac{1}{(t+1)} \Sigma_{i=0}^{t} (g^i)^2}$

Adagrad

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

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

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

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

$$\vdots$$

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

- Adagrad 中的**矛盾** 在一般的 Gradient Descent，<u>gradient 越大，參數 update 的越快</u> 但是，在 Adagrad 中，<u>gradient 越大，**分子 update 的步伐越大，分母 update 的步伐卻越小**</u>，兩者互相矛盾

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

Vanilla Gradient descent

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

Larger gradient, larger step

Adagrad

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

Larger gradient, larger step

Larger gradient, smaller step

- 論文中對**矛盾**的一些解釋
    - 直觀想法：Adagrad 中想要考慮的是 gradient 的**「反差」**，也就是遇到某一個特別大或特別小的 gradient 時的反差有多大這樣

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

- 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	……

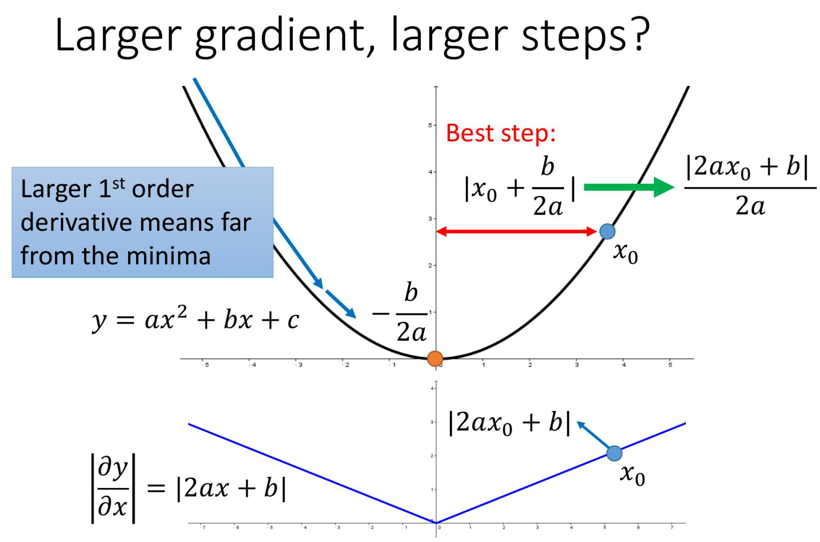
g^0	g^1	g^2	g^3	g^4	……
10.8	20.9	31.7	12.1	0.1	……

特別小

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

造成反差的效果

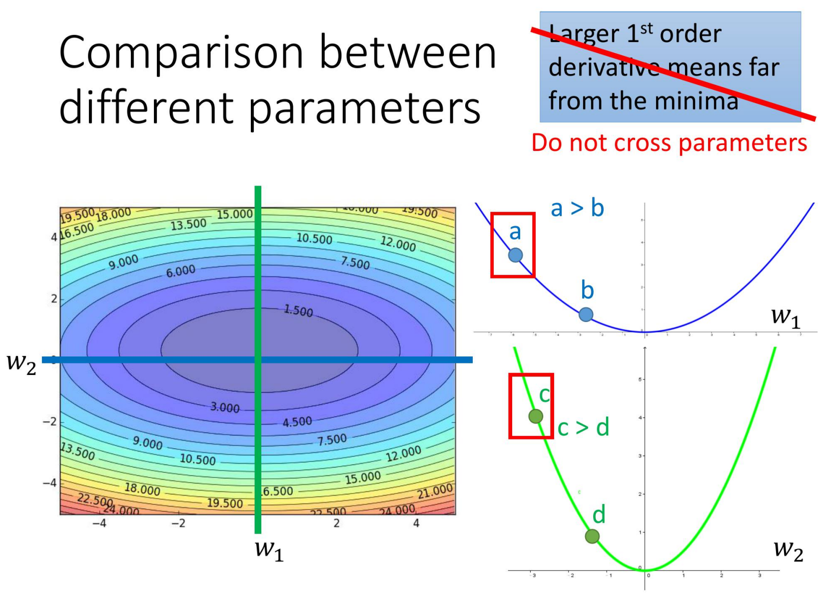
- 正式想法（只有一個參數時）
    1. 考慮一個二次函數，及其絕對值的圖形，最低點就是 $-\frac{b}{2a}$
    2. 隨機找一個點 (x_0) 做 Gradient Descent ，那最好的步伐就是這個點與最低點的距離，也就是 $|x_0 + \frac{b}{2a}|$； 整理下，又可以寫成 $\frac{|2ax_0+b|}{2a}$ ，而 $|2ax_0 + b|$ 就是x_0 這一點的微分
    3. 所以，如果今天算出來的微分值越大，代表離原點越遠，踏出去最好的步伐跟微分大小成正比 但是！這件事只在考慮「一個參數」的時候才成立，在多個參數的時候，不一定成立。



- 正式想法（多個參數時）
  1. 只考慮 w_1 (藍色切線) 時，可以看到 error surface 上「a 的微分值 > b」，a 也的確離原點較遠

     只考慮 w_2 (綠色切線) 時，可以看到 error surface 上「c 的微分值 > d」，c 也的確離原點較遠
  2. 但跨參數，同時比較「a 對 w_1 的微分」及「c 對 w_2 的微分」，可以發現<u>後者較大，但是後者離低點比近</u>

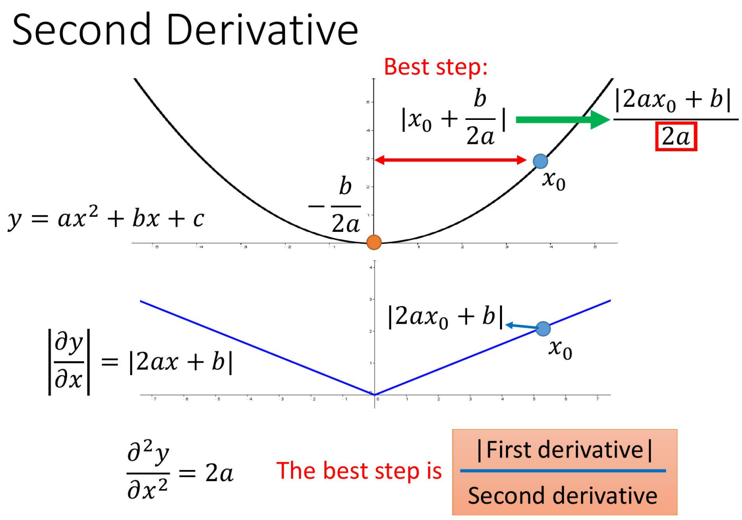
  ⇒ 所以，update參數選擇和微分值大小成正比是在「單一參數」的條件下才成立。



- 多參數時的 Adagrad

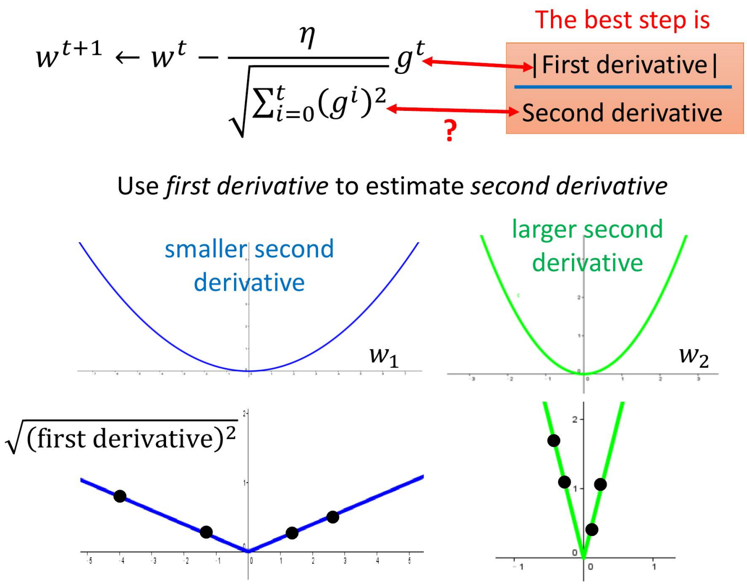
  我們回過頭看上一個圖，任一點 (x_0) 最好的步伐 $\frac{|2ax_0+b|}{2a}$ 中，分母 $2a$ 這項其實是 y 的二次微分。

  所以，最好的步伐要<u>正比於一次微分</u>且與<u>二次微分的大小成反比</u>。



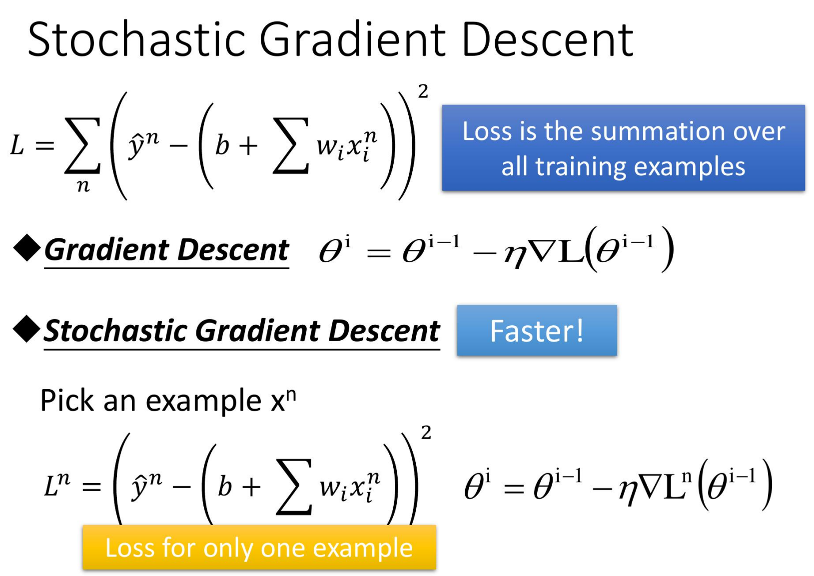
當參數量大、資料量多的時候，我們可以從一次微分去估算二次微分（如下圖）

藍色是比較平滑的峽谷（二次微分較小）、綠色是比較陡峭的峽谷（二次微分較大）

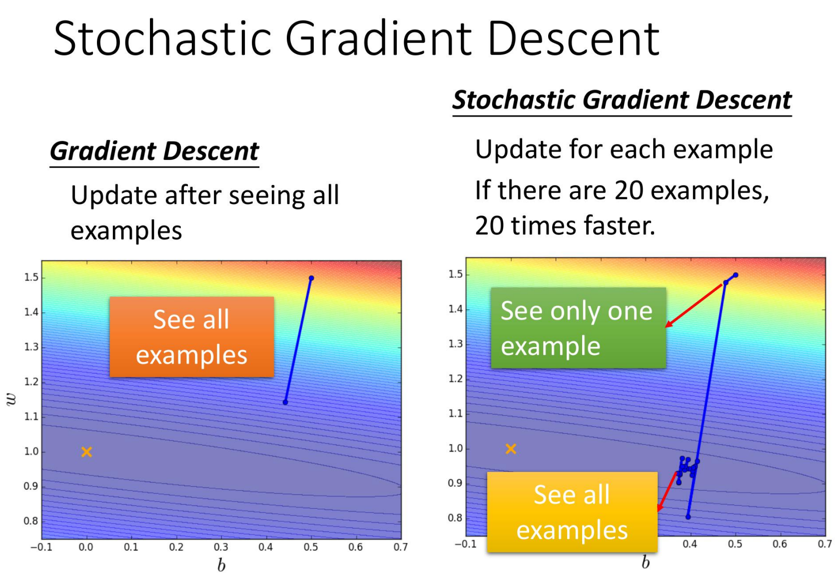


Tips 2: Stochastic Gradient Descent

- **想法比較**
  - Regression 的 Loss function：$L = \Sigma_n(\hat{y}^n - (b + \Sigma w_i x_i^n))^2$，是 summation over <u>所有 training data</u>
  - Stochastic Gradient Descent：每次取一個 x^n (可隨機也可照順序)，<u>只考慮這個 example</u> 的 Loss function，寫作 L^n 而在 update 參數的時候，就只看這個 example 的 loss 更新參數；每看完一個參數，就更新一次。

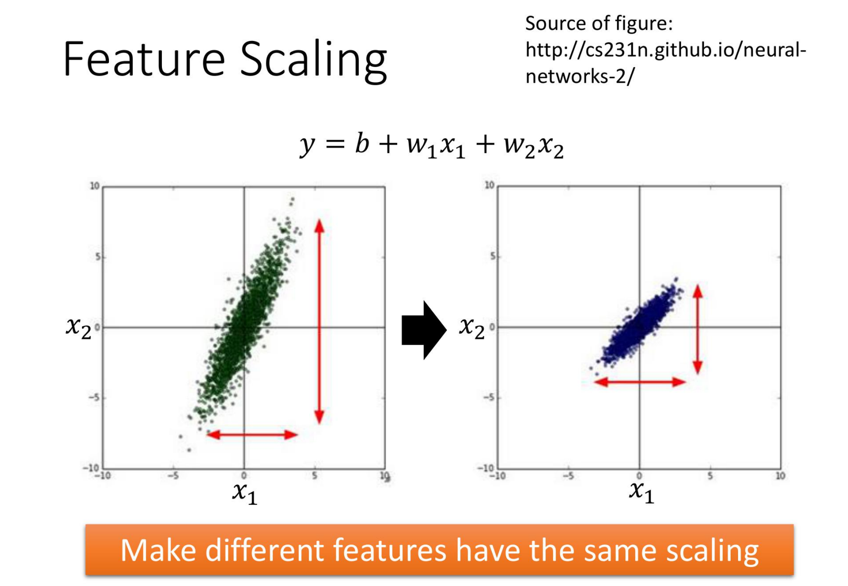


- **優點**
    - Gradient Descent：會看完全部 example，再 update 參數，**較穩定**。
    - Stochastic：每看完一個 example，就更新一次，假設有 20 個 example 的話，更新速度就是上者的 **20 倍**。



Tip 3: Feature Scaling

- **目的**：希望讓不同 feature 有相同的 scaling（讓他們的 range 分佈變成一樣）



- **舉例**：$y = b + w_1 x_1 + w_2 x_2$
  - 圖形：

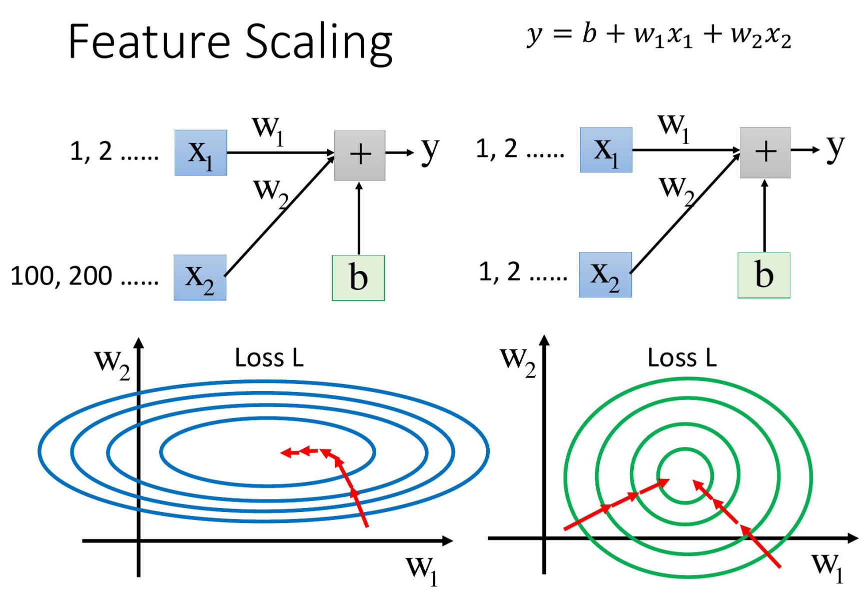
    左圖：x_1 的值都是比較小的，x_2 的值都是較大的；當 w_1 跟 w_2 做一樣的更動時，w_1 的變化對 y 較小，w_2 的變化對 y 較大

    右圖：x_1 跟 x_2 的 scale 是接近的，w_1、w_2 對 loss 有差不多的影響力，畫出來的圖形接近圓形
  - 執行 Gradient Descent：

    左圖：長橢圓的 error surface 需要不同的 learning rate，也就是要用 adaptive learning

    右圖：正圓形的 error surface，不論從哪個點開始，都會向著圓心走

    $\Rightarrow$ 有做 feature scaling，則在參數的 update 上較有效率。



- **常見作法**：

對 r 個 example 的每一個 dimension i 做 **normalization**（$x_i^r \leftarrow \frac{x_i^r - m_i}{\sigma_i}$）使得每個 dimension 的 **mean = 0**, **variance = 1**