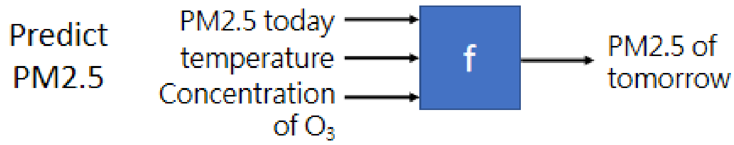
01-機器學習基本概念簡介

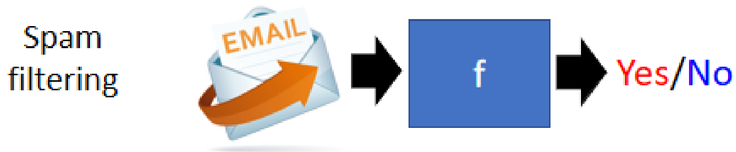
1. 機器學習的不同類型

機器學習就是讓機器具備找一個函式的能力

Regression: The function outputs a scalar.



Classification: Given options (**classes**), the function outputs the correct one.



1.1 Regression

要找的函式，其輸出是一個數值

1.2 Classification

函式的輸出，就是從設定好的選項裡選擇一個當作輸出

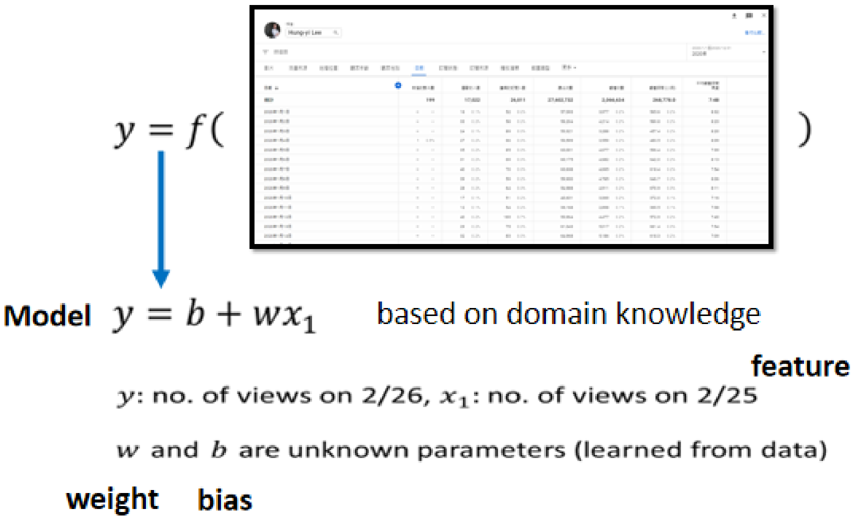
1.3 Structured Learning

機器產生有結構的東西的問題，學會創造

2. Case Study：預測頻道流量

2.1 訓練三步驟

Step 1：Function with Unknown Parameters



- y **是要預測的東西**
- x_1 **是頻道前一天總觀看人數，**跟 y 一樣都是數值
- b 跟 w 是**未知的參數，**透過準備資料找出最適合的參數

  猜測：

  未來點閱次數的函式 f，是前一天的點閱次數乘上 w 再加上 b。猜測往往是對問題本質上的了解，是 **domain knowledge**

名詞定義：

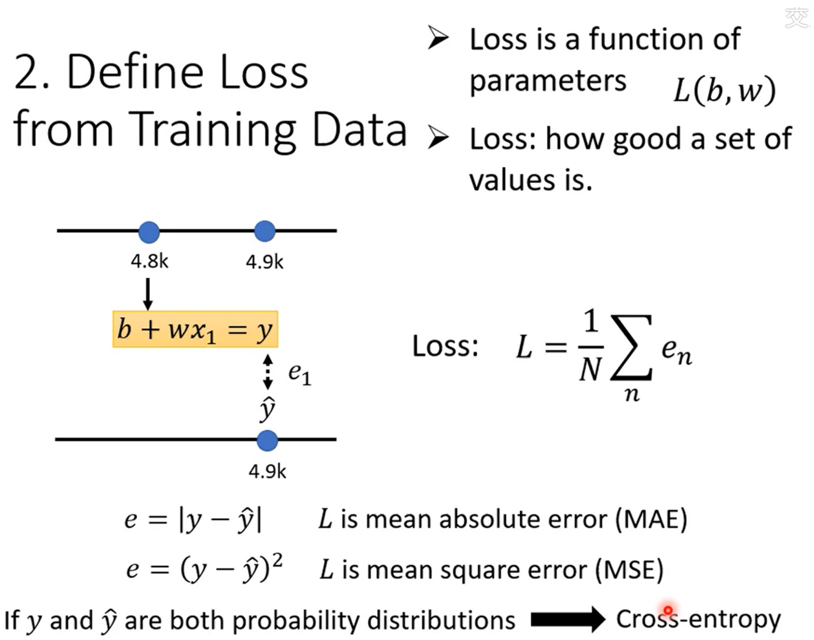
- **Feature：**function 中已知的訊息（x_1）
- **Weight：**未知參數，與 feature 直接相乘
- **Bias：**未知參數，直接相加

Step 2：Define Loss from Training Data

loss 也是一個 function，它的輸入是 model 中的參數（w, b）

- **物理意義：function 輸出的值代表，如果把這一組未知的參數設定某一個數值時，這筆數值好還是不好**

  L 越大，代表一組參數越不好，這個大 L 越小，代表現在這一組參數越好
- **計算方法：求取估測的值跟實際的值（label） 之間的差距**
  - MAE（mean absolute error）
  - MSE（mean square error）
  - Cross-entropy：計算**機率分布**之間的差距



Error Surface：

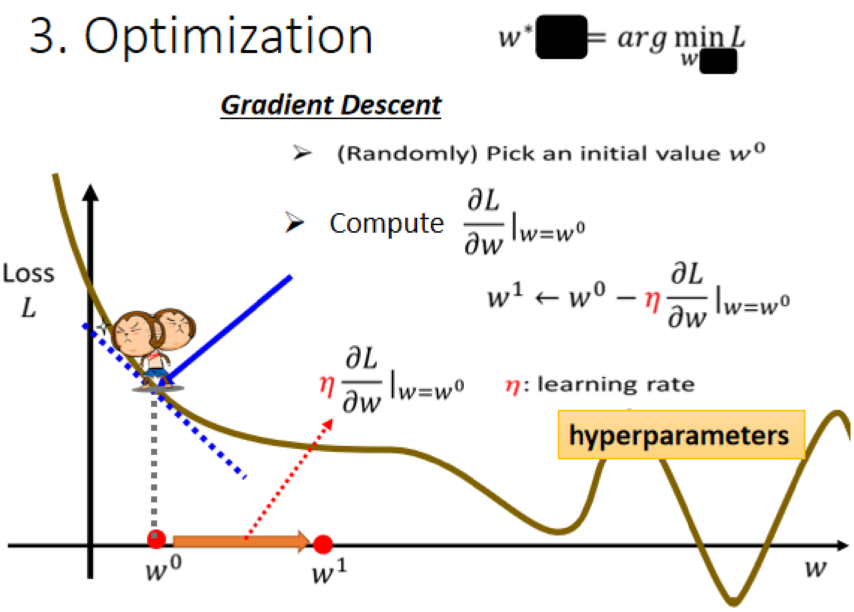
試**不同的參數，**然後**計算 loss 所畫出來的等高線圖**

Step 3：Optimization

找到能讓損失函數值最小的參數

方法：

Gradient Descent（梯度下降）



步驟：

1. 隨機選取初始值 w_0
2. 計算 $w = w_0$ 時，w 對 loss 的微分是多少
3. 根據微分（梯度）的方向，改變參數的值

   改變的大小取決於：

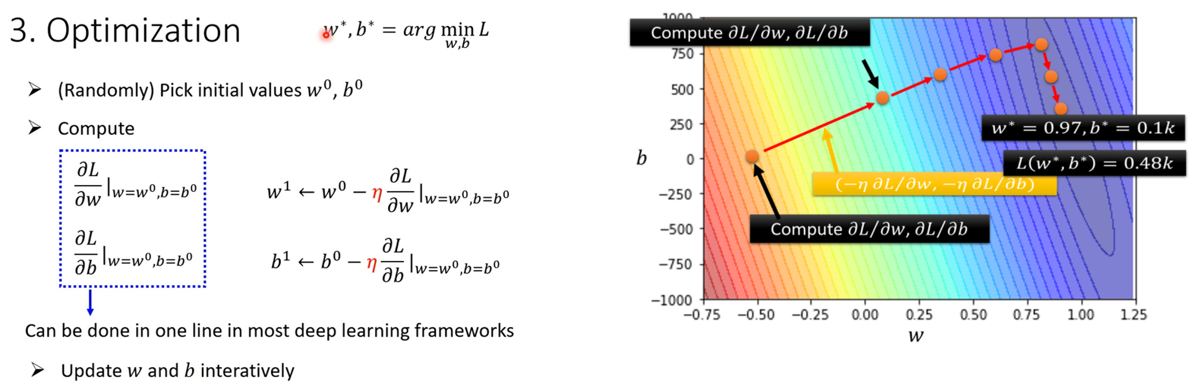
   - 斜率的大小
   - 學習率的大小**（超參數）**

4. 什麼時候停下來？

   a. 自己設置上限**（超參數）**

   b. 理想情況：微分值為 0（極小值點），不會再更新 ⇒ **有可能陷入局部最小值**，不能找到全局最小值
   事實上：局部最小值不是真正的問題！！！

推廣到多個參數：



2.2 Linear Model

根據周期性修改模型，考慮**前7天，甚至更多天**的值

$$y = b + wx_1$$

2017 - 2020 $L = 0.48k$ 2021 $L' = 0.58k$

$$y = b + \sum_{j=1}^{7} w_j x_j$$

2017 - 2020 $L = 0.38k$ 2021 $L' = 0.49k$

b	w_1^*	w_2^*	w_3^*	w_4^*	w_5^*	w_6^*	w_7^*
0.05k	0.79	-0.31	0.12	-0.01	-0.10	0.30	0.18

$$y = b + \sum_{j=1}^{28} w_j x_j$$

2017 - 2020 $L = 0.33k$ 2021 $L' = 0.46k$

$$y = b + \sum_{j=1}^{56} w_j x_j$$

2017 - 2020 $L = 0.32k$ 2021 $L' = 0.46k$

Linear models

2.2.1 Model Bias

問題：

模型遇到無法模擬或描述真實情況的狀況

解決：

需要一個更複雜的、更有彈性的、有未知參數的 function

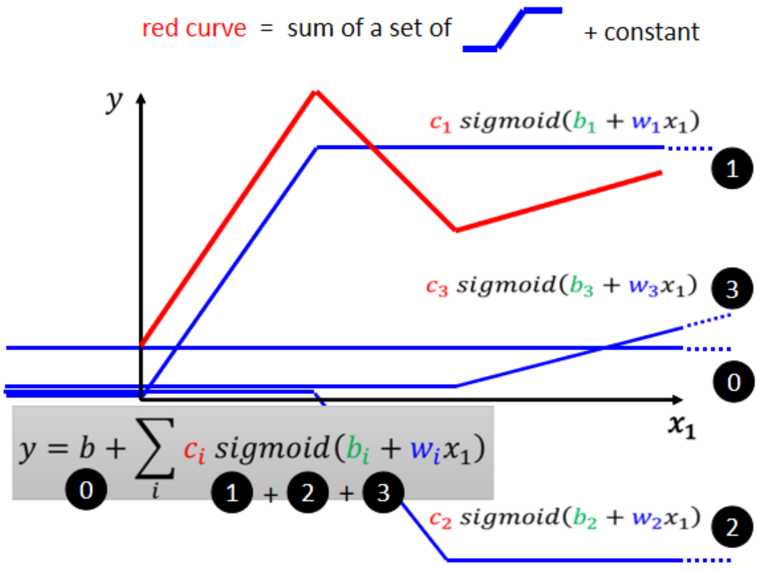
2.3 Piecewise Linear Curves（Sigmoid）

2.3.1 模型定義

定義：

由多段鋸齒狀的線段所組成的線，可以看作是一個常數，再加上若干個藍色的 function（**Hard Sigmoid**）

$$y = b + \sum_i sigmoid(b_i + w_i x_i)$$



用一條曲線來近似描述這條藍色的曲線：**Sigmoid 函數（S 型的 function）**

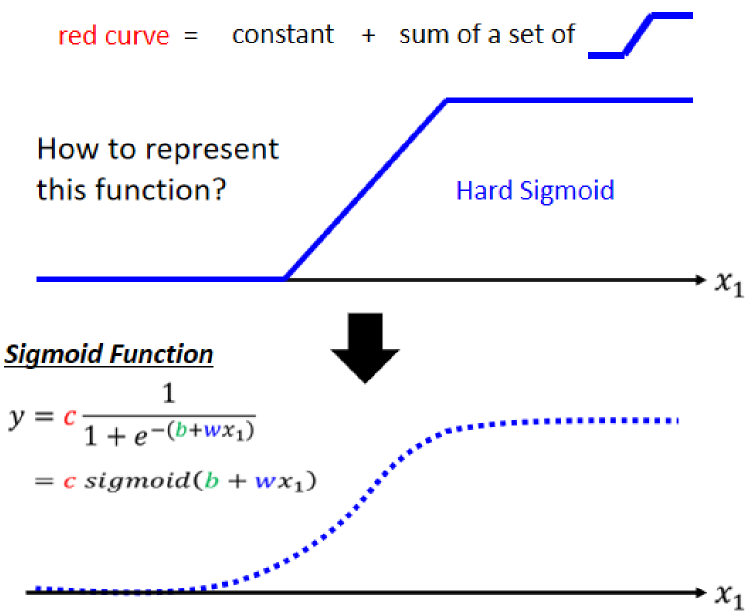
事實上，sigmoid 的個數就是神經網絡中的一層的 neuron 節點數（使用幾個 sigmoid 是**超參數**）

結論：

1. 可以用 **Piecewise Linear 的 Curves**，去**逼近任何的連續的曲線**
2. 每一個 **Piecewise Linear 的 Curves**，都可以用**一大堆藍色的 Function 加上一個常量組合**起來得到
3. 只要有**足夠的藍色 Function** 把它加起來，就可以變成任何連續的曲線

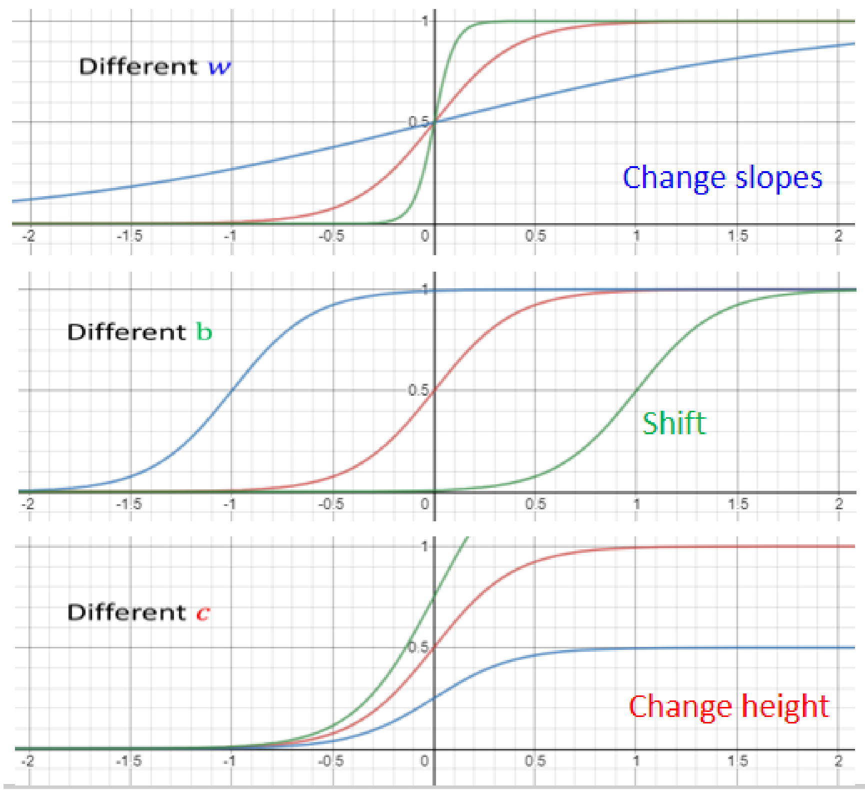
2.3.2 Sigmoid 函數

$$Sigmoid：y = c\frac{1}{1+e^{-(b+wx_1)}}$$



- x_1 趨近於正無窮大 ⇒ 收斂於高度 c
- x_1 負非常大，分母就會非常大 ⇒ y 值趨近於 0

調整 w, b, c，**可以得到各種不同的 sigmiod 來逼近”藍色function“，通過求和，最終近似各種不同的 continuous function**



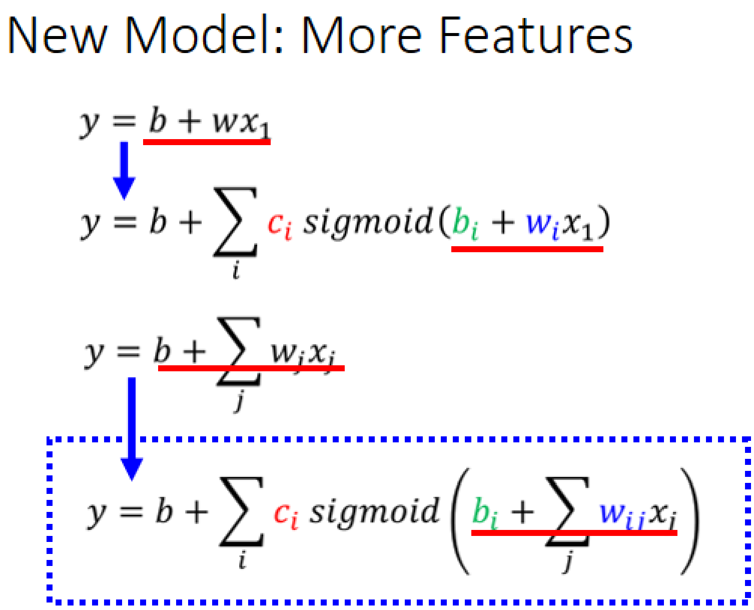
- 如果改 w，就會改變**斜率**，改變斜坡的坡度

- 如果改 b，就可以把 sigmoid function 左右移動
- 如果改 c，就可以改變它的高度

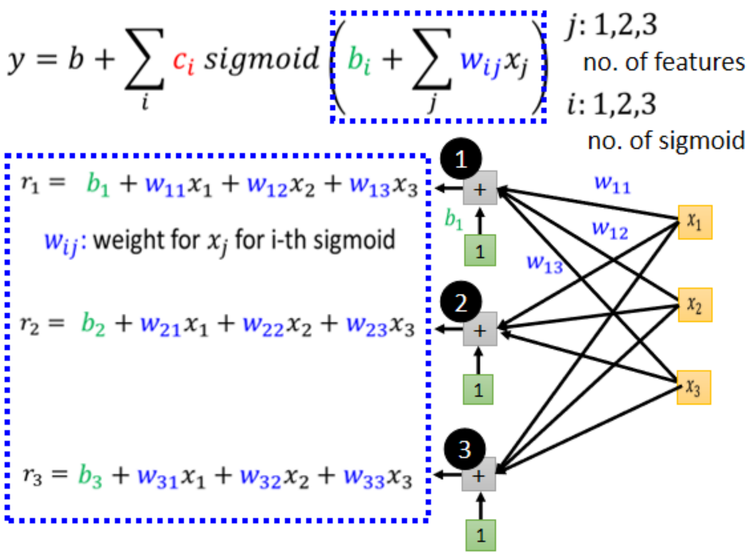
總結：

利用若干個具有不同 w, b, c 的 Sigmoid 函數與一個常數參數的組合，可以模擬任何一個連續的曲線（非線性函數）

擴展到多個特徵：



$$y = b + \sum_i sigmoid(b_i + \sum_j (w_{ij} x_j))$$



- j 等於 1 2 3，x_1 **代表前一天的觀看人數，x_2 兩天前觀看人數，x_3 三天前的觀看人數**
- 每一個 i **就代表了一個藍色的 function**，現在每一個藍色的 function 都用一個 sigmoid function 來近似它
- w_{ij} 第 i 個 sigmoid 給第 j 個 feature 的權重

轉化為矩陣計算 + 激勵函數形式：

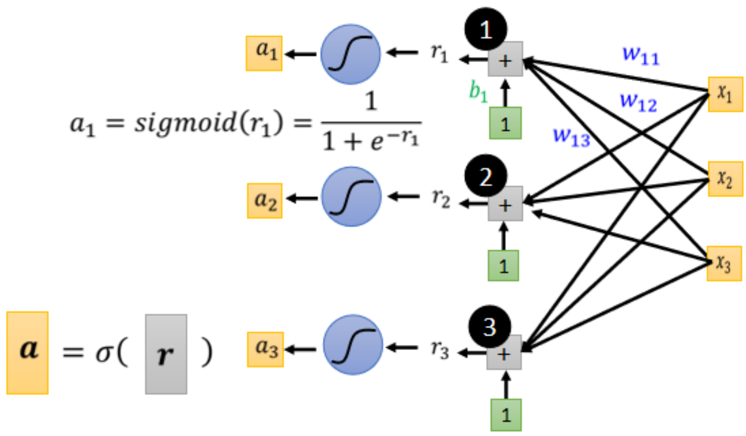
$$y = b + \sum_i c_i \, sigmoid\left(b_i + \sum_j w_{ij} x_j\right) \qquad \begin{matrix} i: 1,2,3 \\ j: 1,2,3 \end{matrix}$$

$$\boxed{\begin{aligned} r_1 &= b_1 + w_{11}x_1 + w_{12}x_2 + w_{13}x_3 \\ r_2 &= b_2 + w_{21}x_1 + w_{22}x_2 + w_{23}x_3 \\ r_3 &= b_3 + w_{31}x_1 + w_{32}x_2 + w_{33}x_3 \end{aligned}}$$

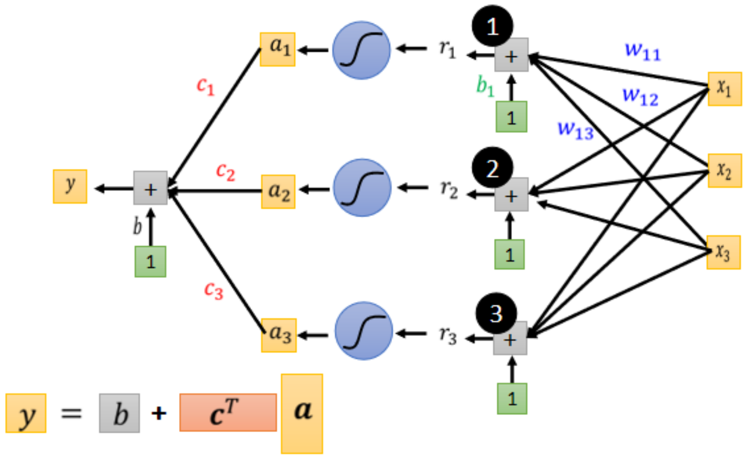
$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} + \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

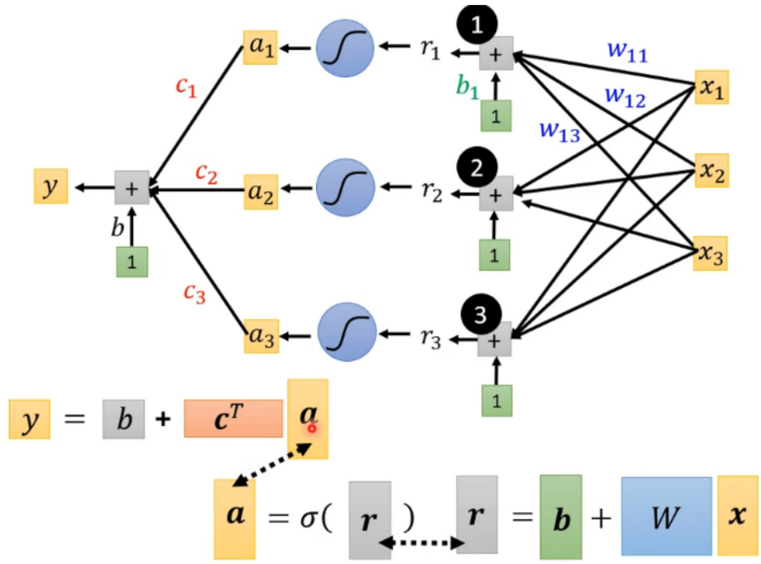
$$\boldsymbol{r} = \boldsymbol{b} + W \boldsymbol{x}$$

$$y = b + \sum_i c_i \, \boxed{sigmoid\left(b_i + \sum_j w_{ij} x_j\right)} \qquad \begin{matrix} i: 1,2,3 \\ j: 1,2,3 \end{matrix}$$

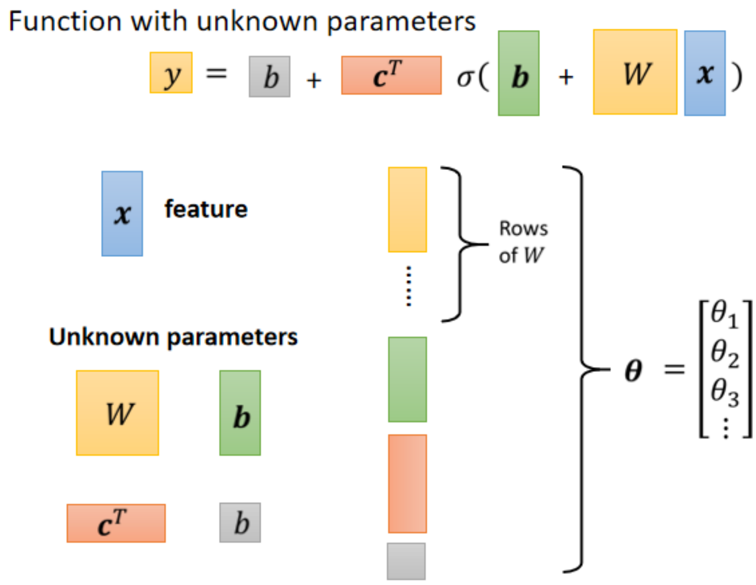


$$y = \boxed{b + \sum_i c_i \, sigmoid\left(b_i + \sum_j w_{ij} x_j\right)} \qquad \begin{matrix} i: 1,2,3 \\ j: 1,2,3 \end{matrix}$$





總之：



一般來說，將所有參數統稱為 θ（包含 $W, \vec{b}, b$...)

2.3.3 寫出 Loss 函數

因為所有參數統稱為 θ，loss 表示為 $L(\theta)$

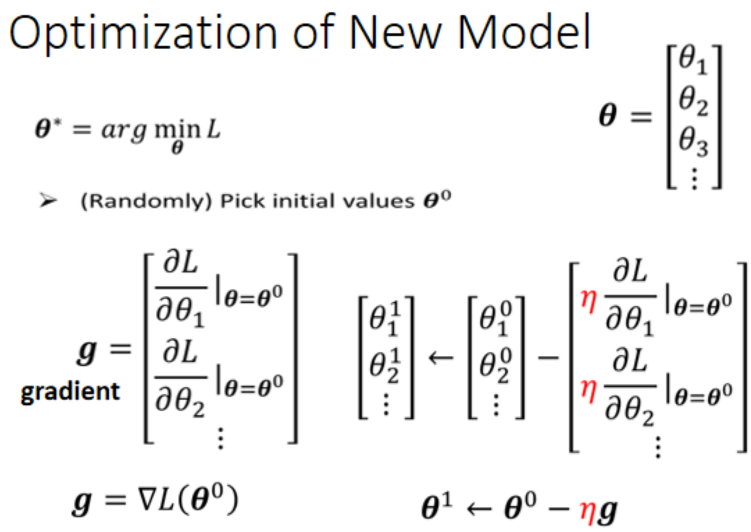
2.3.4 優化過程

仍是梯度下降

（1）選定**初始參數值**（向量） θ_0

（2）對每個參數求**偏微分**

（3）更新參數，直至設定的次數



Optimization of New Model

$$\boldsymbol{\theta}^* = arg \min_{\boldsymbol{\theta}} L$$

- (Randomly) Pick initial values $\boldsymbol{\theta}^0$
- Compute gradient $\boldsymbol{g} = \nabla L(\boldsymbol{\theta}^0)$

  $\boldsymbol{\theta}^1 \leftarrow \boldsymbol{\theta}^0 - \eta \boldsymbol{g}$
- Compute gradient $\boldsymbol{g} = \nabla L(\boldsymbol{\theta}^1)$

  $\boldsymbol{\theta}^2 \leftarrow \boldsymbol{\theta}^1 - \eta \boldsymbol{g}$
- Compute gradient $\boldsymbol{g} = \nabla L(\boldsymbol{\theta}^2)$

  $\boldsymbol{\theta}^3 \leftarrow \boldsymbol{\theta}^2 - \eta \boldsymbol{g}$

▲ Batch training

每次更新參數時，只使用 **1 個 batch** 裡的資料計算 loss，求取梯度，更新參數

> batch 大小也是**超參數**

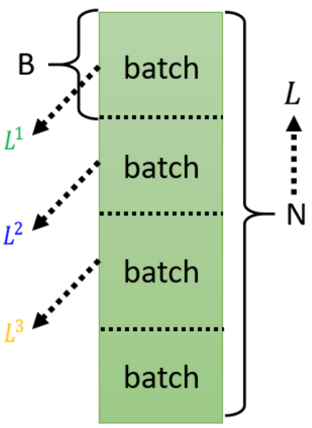
$$\boldsymbol{\theta}^* = arg \min_{\boldsymbol{\theta}} L$$

- (Randomly) Pick initial values $\boldsymbol{\theta}^0$
- Compute gradient $\boldsymbol{g} = \nabla L^1(\boldsymbol{\theta}^0)$

  update $\boldsymbol{\theta}^1 \leftarrow \boldsymbol{\theta}^0 - \eta \boldsymbol{g}$
- Compute gradient $\boldsymbol{g} = \nabla L^2(\boldsymbol{\theta}^1)$

  update $\boldsymbol{\theta}^2 \leftarrow \boldsymbol{\theta}^1 - \eta \boldsymbol{g}$
- Compute gradient $\boldsymbol{g} = \nabla L^3(\boldsymbol{\theta}^2)$

  update $\boldsymbol{\theta}^3 \leftarrow \boldsymbol{\theta}^2 - \eta \boldsymbol{g}$



1 **epoch** = see all the batches once

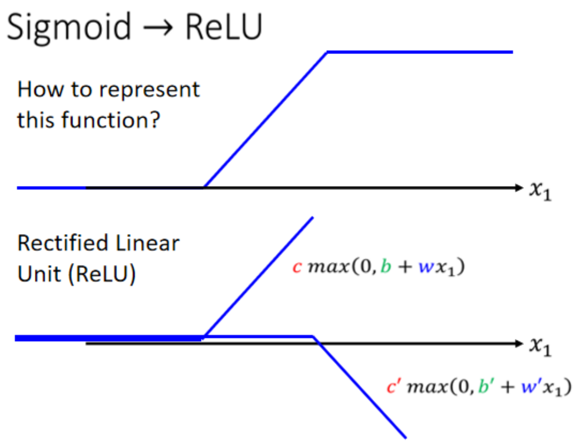
Update：每次更新一次參數叫做一次 Update

Epoch：把所有 batch 都看過一遍叫做一個 Epoch

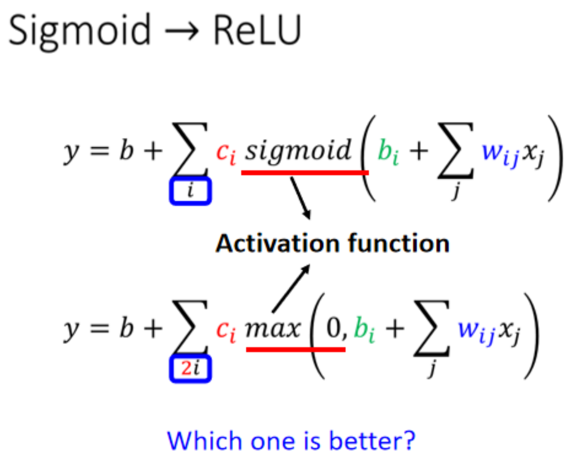
2.4 ReLU

把兩個 ReLU 疊起來就等於 **hard sigmoid**

$$y = b + \sum_i max(0, b_i + w_i x_i)$$

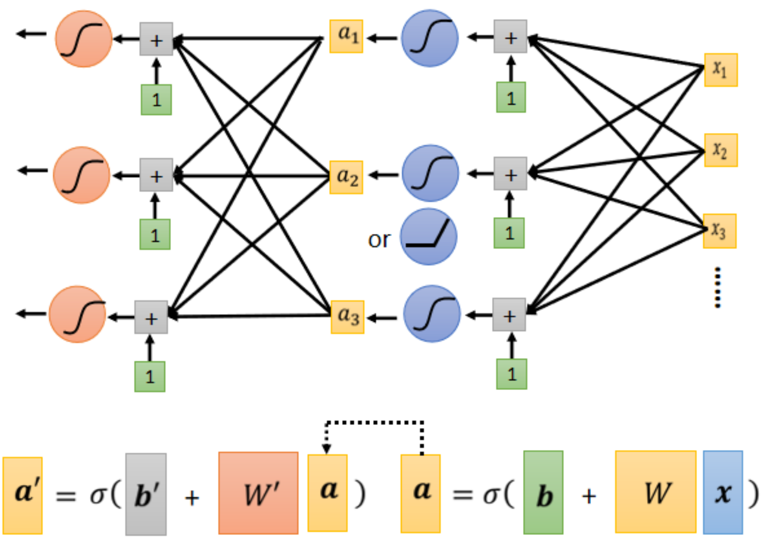


兩個 ReLU 可以合成一個 hard sigmoid



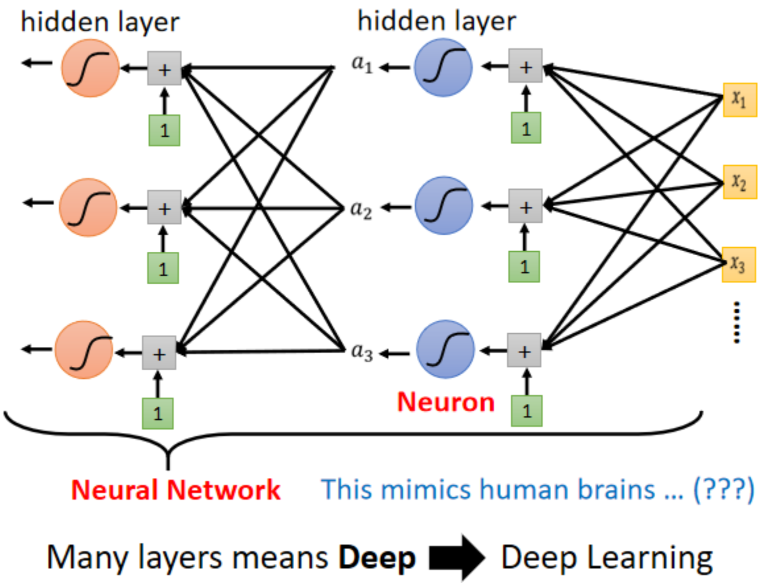
Sigmoid 和 ReLU 都屬於激勵函數（**Activation Function**）

▲ 模型變型 ⇒ 多加幾層



函數的函數的函數…的函數

3. 引入 Deep Learning



問題：

為什麼要“深”，而不“胖”？

Why don't we go deeper?

- Loss for multiple hidden layers
  - 100 ReLU for each layer
  - input features are the no. of views in the past 56 days

	1 layer	2 layer	3 layer	4 layer
2017 – 2020	0.28k	0.18k	0.14k	0.10k
2021	0.43k	0.39k	0.38k	0.44k

Better on training data, worse on unseen data

➡ Overfitting

Overfitting：在訓練資料上有變好，但是在沒看過的資料上沒有變好

4. Learn More

To learn more ……

Basic Introduction

https://youtu.be/Dr-WRlEFefw

Backpropagation

Computing gradients in an efficient way

https://youtu.be/ibJpTrp5mcE