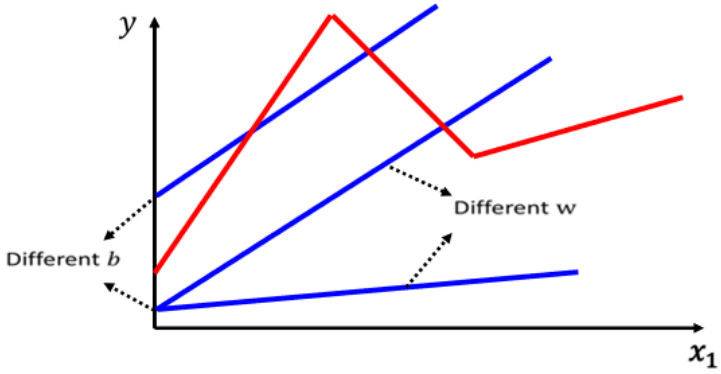
Regression P2

Piecewise Linear Curves

Linear 的 Model,也许太过简单了,我们可以想像说 x1 跟 y,也许它中间有比较复杂的关係,**对 Linear 的 Model 来说,x1 跟 y 的关係就是一条直线**,随著 x1 越来越高,y 就应该越来越大,你可以设定不同的 w,改变这条线的斜率,你可以设定不同的 b,改变这一条蓝色的直线,跟 y 轴的交叉点,但是无论你怎麼改 w 跟 b,它永远都是一条直线,永远都是 x1 越大,y 就越大,前一天观看的人数越多,隔天的观看人数就越多

Linear models are too simple ... we need more sophisticated modes.



Linear models have severe limitation. ***Model Bias***

We need a more flexible model!

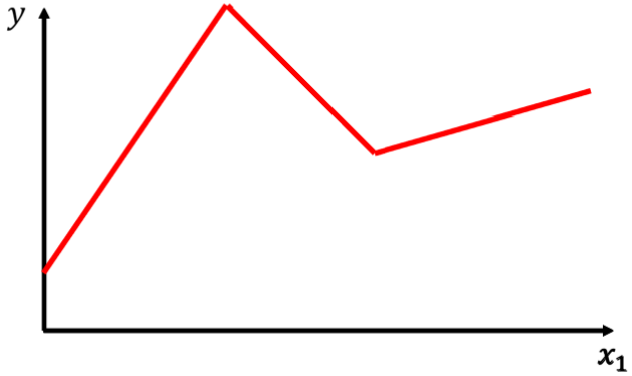
但也许现实并不是这个样子

- 也许在 x1 小於某一个数值的时候,前一天的观看人数跟隔天的观看人数是成正比,
- 也许当 x1 大於一个数值的时候,这个物极必反,过了一个假设 x1 太大,前天观看的人数太高,那隔天观看人数就会变少,也说不定
- 也许 x1 跟 y 中间,有一个比较复杂的,像这个红色线一样的关係

但你不管怎麼摆弄w 跟 b,你永远製造不出红色那一条线,你永远无法用 Linear 的 Model,製造红色这一条线,显然 Linear 的 Model 有很大的限制,**这一种来自於 Model 的限制,叫做 Model 的 Bias**,那其实我们刚才在课堂一开始的时候也叫做,也说 b 叫做 Bias,那这个地方有一点,在用词上有一点 Ambiguous,所以特别强调说,这个东西叫做 **Model 的 Bias,跟 b 的这个 Bias 不太一样**,它指的意思是说,没有办法模拟真实的状况

所以我们需要写一个更复杂的,更有弹性的,有未知参数的 Function,

red curve = constant + sum of a set of 



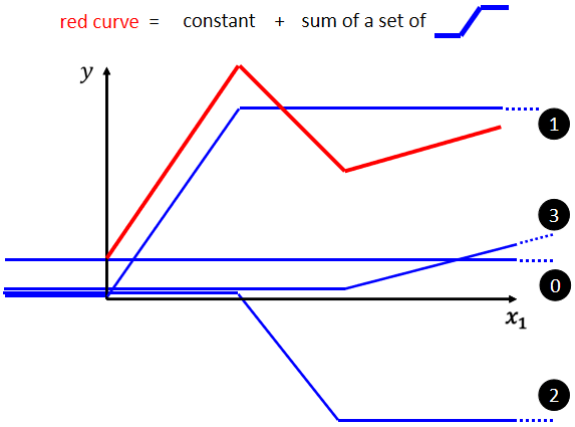
我们可以观察一下红色的这一条曲线,它可以看作是**一个常数,再加上一群蓝色的这样子的 Function**,.

这个蓝色的 Function,它的特性是

- 当输入的值,当 x 轴的值小於某一个这个 Flash Hold 的时候,它是某一个定值,
- 大於另外一个 Flash Hold 的时候,又是另外一个定值,
- 中间有一个斜坡

所以它是先水平的,然后再斜坡,然后再水平的,那它其实有名字,它的名字我们等一下再讲,这边我们因為它是蓝色的 Function,我们就先叫它蓝方吧 这样子,好 那所以呢 这个红色的线啊,它可以看作是一个常数项加一大堆的蓝方,好 那这个**常数项**,它的值应该要有多大呢,你就看这一条**红色的线啊,它跟 x 轴的交点在哪裡**,好 那这个常数项呢,就设跟 x 轴的交点一样大

那怎麼加上这个蓝色的 Function 以后,变成红色的这一条线?

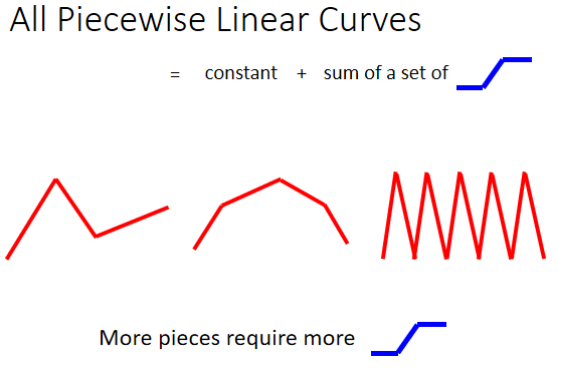


蓝线"1"Function 斜坡的**起点**,设在**红色 Function** 的**起始**的地方,然后第二个,**斜坡的终点**设在第一个转角处,你刻意让这边这个蓝色 Function 的斜坡,跟这个红色 Function 的斜坡,它们的**斜率是一样**的,这个时候如果你把 0 加上 1,你就可以得到红色曲线

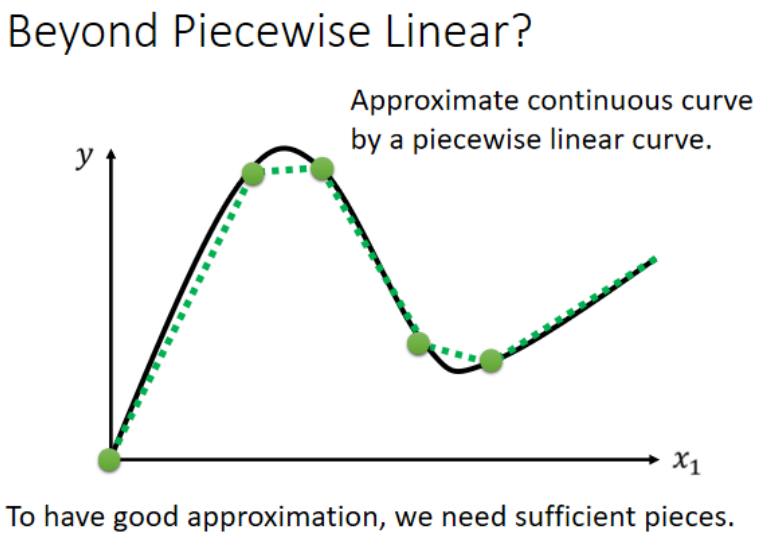
然后接下来,再加第二个蓝色的 Function,你就看红色这个线,**第二个转折点**出现在哪裡, 所以第二个**蓝色 Function**,它的斜坡就在红色 Function 的**第一个转折点,到第二个转折点之间**,你刻意让这边的**斜率跟这边的斜率一样**,这个时候你把 0加 1+2,你就可以得到两个转折点这边的线段,就可以得到红色的这一条线这边的部分

然后接下来第三个部分,第二个转折点之后的部分,你就加第三个蓝色的 Function,第三个蓝色的 Function,它这个坡度的起始点,故意设的跟这个**转折点一样**,这边的斜率,故意设的跟这边的**斜率一样**,好 接下来你把 0加 1+2+3 全部加起来,你就得到红色的这个线。

所以**红色这个线,可以看作是一个常数,再加上一堆蓝色的 Function**



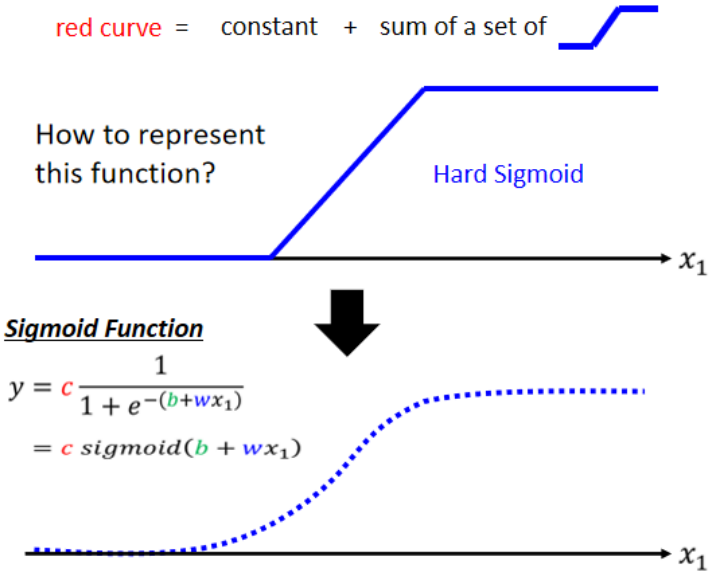
你现在这个 Curves 啊,它是有很多线段所组成的,它是有很多锯齿状的线段所组成的,这个叫做**Piecewise Linear 的 Curves**,那你会发现说这些 Piecewise Linear 的 Curves,你有办法用常数项,加一大堆的蓝色 Function 组合出来,只是他们用的蓝色 Function 不见得一样,你要有很多不一样的蓝色 Function,加上一个常数以后,你就可以组出这些 Piecewise Linear 的 Curves。那如果你今天 Piecewise Linear 的 Curves 越复杂,也就是这个转折的点越多啊,那你需要的这个蓝色的 Function 就越多



讲到这边有人可能会说,那也许我们今天要考虑的 x 跟 y 的关係**不是 Piecewise Linear 的 Curves** ,也许它是这样子的曲线,那就算是这样的曲线,也无所谓,我们可以在这样的曲线上面,先取一些点,再把这些点点起来,变成一个 Piecewise Linear 的 Curves,而这个 Piecewise Linear 的 Curves 跟原来的曲线,它会非常接近,如果你今天点取的够多,或你点取的位置适当的话,这个 Piecewise Linear 的 Curves,就可以逼近这一个,连续的这一个曲线,就可以逼近这一个不是 Piecewise Linear,它是有角度的 有弧度的这一条曲线。

所以我们今天知道一件事情,你可以用 Piecewise Linear 的 Curves,去逼近任何的连续的曲线,而每一个 Piecewise Linear 的 Curves,又都可以用一大堆蓝色的 Function 组合起来,也就是说,我只要有足够的蓝色 Function 把它加起来,我也许就可以变成任何连续的曲线

所以今天,假设我们的 **x 跟 y 的关係,它也许非常地复杂**,那也没关係,我们就想办法写一个带有未知数的 Function,这个带有未知数的 Function 它表示的,就是一堆蓝色的 Function,加上一个 Constant,那我们接下来要问的问题就是,这一个蓝色 Function,它的式子应该要怎麼把它写出来呢?



也许你要直接写出它 没有那麼容易,但是你可以用一条曲线来理解它,用一个 Sigmoid 的 Function,来逼近这一个蓝色的 Function,那 Sigmoid Function,它的式子长的是这个样子的,

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

它的横轴输入是 x1,输出是 y,输入的 x1,我们先乘上一个 w,再加上一个 b,再取一个负号,再取 Exponential,再加 1,这一串被放在分放在,放在分母的地方

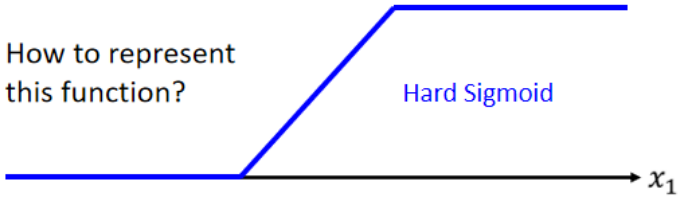
把 1 除以 1 加上 Exponential -b+wx1,前面 你可以乘上一个 Constant 叫做 c,好 那

- 如果你今天输入的这个 x1 的值,趋近於无穷大的时候,那 $e^{-(b+wx_1)}$这一项就会消失,那当 x1 非常大的时候,这一条这边就会收敛在这个高度是 c 的地方
- 那如果今天 x1 负的非常大的时候,分母的地方就会非常大,那 y 的值就会趋近於 0.

所以你可以用这样子的一个 Function逼近这一个蓝色的 Function,那这个东西它的名字叫做 Sigmoid,Sigmoid,如果你要 硬要翻成中文的话,可以翻成 S 型的,所以 Sigmoid Function 就是 S 型的 Function,因為它长得是 有点像是 S 型的哦,所以叫它 Sigmoid Function,那这边我们之后都懒得把 Exponential 写出来,我们就直接写成这个样子

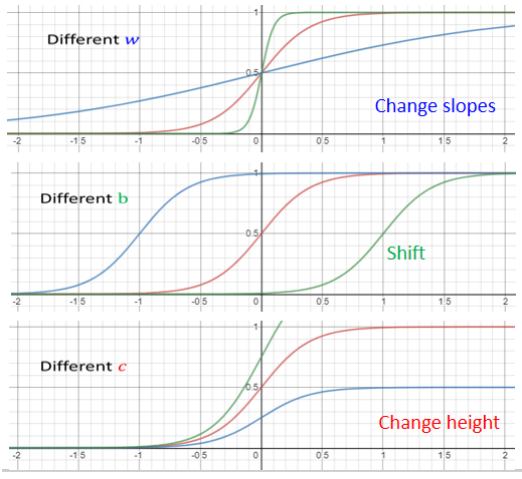
$$y = c * sigmoid(b + wx_1)$$

就是 y 等於 c 倍的 Sigmoid,然后这个括号裡面放$b + wx_1$,然后这个 $b + wx_1$,实际上做的事情,就是把它放在 Exponential 的指数下,前面加一个负号,然后 1+Exponential 的$-(b + wx_1)$ 放在分母的地方,然后前面乘上 c,就等於 y



所以我们可以用这个 Sigmoid Function,去逼近一个蓝色的 Function,那其实这个蓝色的 Function,比较常见的名字就叫做,Hard 的 Sigmoid 啦,只是我本来是想说一开始,我们是先介绍蓝色的 Function,才介绍 Sigmoid,所以一开始说它叫做 Hard Sigmoid,有一点奇怪,所以我们先告诉你说,有一个 Sigmoid Function,它可以逼近这个蓝色的 Function,那这个蓝色的 Function,其实通常就叫做 Hard 的 Sigmoid

那我们今天我们需要各式各样不同的,蓝色的 Function,还记得吗,我们要**组出各种不同的曲线,那我们就需要各式各样合适的蓝色的 Function**,而这个合适的蓝色的 Function 怎麼製造出来呢

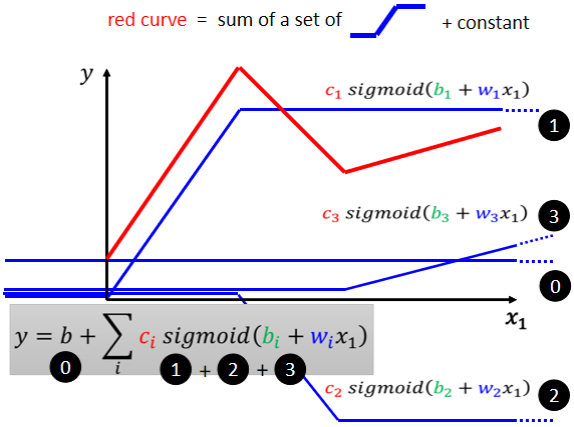


$$y = c\frac{1}{1+e^{-(b+wx_1)}}$$

我们就需要调整这裡的 b 跟 w 跟 c,你就可以製造各种不同形状的 Sigmoid Function,用各种不同形状的 Sigmoid Function,去逼近这个蓝色的 Function.

- 如果你今天改 w 你就会改变**斜率**你就会改变斜坡的坡度
- 如果你动了 b 你就可以把这一个 Sigmoid Function 左右移动
- 如果你改 c 你就可以改变它的高度

所以你只要有**不同的 w 不同的 b 不同的 c,你就可以製造出不同的 Sigmoid Function**,把**不同的 Sigmoid Function 疊起来以后,你就可以去逼近各种不同的,Piecewise Linear 的 Function**,然后 Piecewise Linear 的 Function,可以拿来**近似各种不同的 Continuous 的 Function**



所以假设我们要把红色的这条线,它的函数写出来的话,那可能长什麼样子呢?

我们知道说**红色这条线** 就是 0加 1+2+3,而这个 123 啊,它们都是蓝色的 Function,所以它们的函式 就是有一个固定的样子,它们都写做 $(b + wx_1)$,去做 Sigmoid 再乘上 c1,只是 1 跟 2 跟 3,它们的 w 不一样,它们的 b 不一样,它们的 c 不一样,如果是第一个蓝色 Function,它就是 w1 b1 c1,第二个蓝色 Function,我们就说它的,它用的是 w2 b2 c2,第三个蓝色 Function,我们就说它用的是 w3 b3 c3,好 那我们接下来呢,就是把 0 跟 123 全部加起来以后,我们得到的函式,就长这一个样子

$$y = b + \sum_i c_i sigmoid(b_i + w_i x_1)$$

我们把 1+2+3 加起来,然后 Summation 裡面呢,就是 ci 乘上 Sigmoid,bi+wi 乘上 x1,所以这边**每一个式子,都代表了一个不同蓝色的 Function,Summation 的意思,就是把不同的蓝色的 Function 给它加起来**,就是这边 Summation 的意思.

别忘了加一个 Constant,这边用 b 来表示这个 Constant

所以我们今天就写出了一个这样子的 Function,如果我们假设裡面的 b 跟 w 跟 c,它是未知的,它是我们未知的参数,那我们就可以设定不同的 b 跟 w 跟 c,设定不同的 b 跟 w 跟 c,我们就可以製造不同的蓝色的 Function,製造不同的蓝色的 Function 疊起来以后,就可以製造出不同的红色的 Curves,製造出不同的红色的 Curves,就可以製造出不同的 Piecewise Linear 的 Curves,就可以去逼近,各式各样不同的 Continuous 的 Function

$$y = b + wx_1$$
$$\downarrow$$
$$y = b + \sum_i c_i\ sigmoid(b_i + w_ix_1)$$

所以我们其实有办法写出一个,这个非常有弹性的,有未知参数的 Function,它长这个样子就是 Summation 一堆 Sigmoid,但它们有不同的 c 不同的 b 不同的 w,好 那所以本来我们是 Linear 的 Model,y 等於 b+w 乘上 x1,它有非常大的限制,这个限制叫做 Model 的 Bias,那我们要**如何减少 Model 的 Bias** 呢

我们可以写一个更有弹性的,有未知参数的 Function,它叫做 $y = b + \sum_i c_i sigmoid(b_i + w_i x_1)$ 本来这边是 $b + wx_1$,这边变成$b_i + w_i x_1$,然后我们有很多不同的 bi,有很多不同的 wi,它们都通过 Sigmoid 都乘上 ci,把它统统加起来再加 b 等於 y,我们只要带入不同的 c 不同的 b 不同的 w,我们就可以变出各式各样,就可以组合出各式各样不同的 Function

New Model: More Features

$$y = b + wx_1$$

$$y = b + \sum_i c_i \, sigmoid(b_i + w_i x_1)$$

$$y = b + \sum_j w_j x_j$$

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

那我们刚才其实已经进化到,不是只用一个 Feature,即X_1,我们可以**用多个 Feature**

我们这边用 **j 来代表 Feature 的编号**

举例来说刚才如果要考虑前 28 天的话,j 就是 1 到 28,考虑前 56 天的话,j 就是 1 到 56,那如果把这个 Function,再扩展成我们刚才讲的上面这个,比较有弹性的 Function 的话那也很简单,我们就把 Sigmoid 裡面的东西换掉,本来这边是

$$y = b + \sum_j w_i x_j$$

那这边呢,就把这一项放到这个括号裡面,改成

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

把本来放在这边的东西放到 Sigmoid 裡面,然后呢这个每一个 Sigmoid 的 Function 裡面呢,都有不同的 bi 不同的 w_{ij},然后取 Sigmoid 以后乘上 ci 就全部加起来,再加上 b 就得到 y,我们只要这边 ci bi 跟 w_{ij} 在放不同的值,就可以变成不同的 Function.

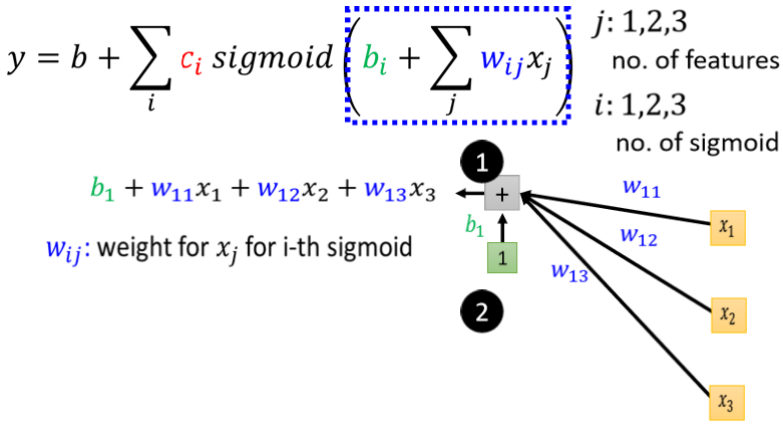
那如果讲到这边你还是觉得有点抽象的话,如果你看这个式子觉得有点头痛的话,那我们用比较直观的方式,把这个式子实际上做的事把它画出来,

我们先考虑一下 j 就是 1 2 3 的状况,就是我们**只考虑三个 Feature**

举例来说 我们只考虑前一天 前两天,跟前三天的 Case,

- 所以 j 等於 1 2 3,那所以输入就是 **x1 代表前一天的观看人数,x2 两天前观看人数,x3 三天前的观看人数**
- 每一个 **i 就代表了一个蓝色的 Function**,只是我们现在每一个蓝色的 Function,都用一个 Sigmoid Function 来比近似它

那这边呢,这个 1 2 3 就代表我们有三个 Sigmoid Function,那我们先来看一下,这个**括号裡面**做的事情是什麼

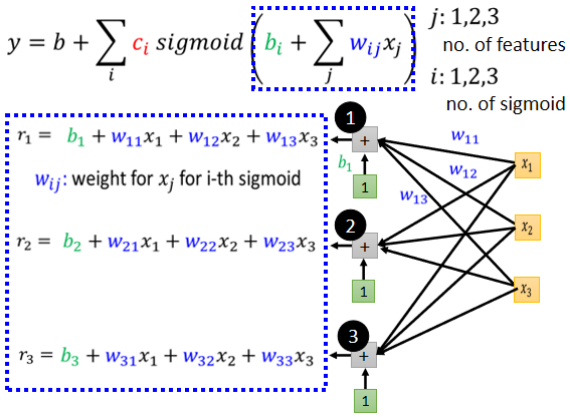


每一个 Sigmoid 都有一个括号,第一个 Sigmoid i 等於 1 的 Case ,就是把

- x1 乘上一个 Weight 叫 w_{11}
- x2 乘上另外一个 Weight 叫 w_{12}
- x3 再乘上一个 Weight 叫做 w_{13}
- 全部把它**加起来**,不要忘了再加一个 b

$$b_1 + w_{11}x_1 + w_{12}x_2 + w_{13}x_3$$

这个得到的式子就是这个样子,所以这边我们**用 w_{ij} ,来代表在第 i 个 Sigmoid 裡面,乘给第 j 个 Feature 的 Weight**,第一个 Feature 它就是 w11,第二个 Features 就是乘 w12,第三个 Feature 都是乘 w13,所以三个 Features1 2 3,这个 w 的第二个下标就是 123,w 的第一个下标代表是,现在在考虑的是第一个 Sigmoid Function,那我们有三个 Sigmoid Function,



第二个 Sigmoid Function,它在括号裡面做的事情就是把 x1 乘上 w21,把 x2 x2 乘上 w22,把 x3 x3 乘上 w23,统统加起来再加 b2

第三个 Sigmoid 呢,第三个 Sigmoid 在括号裡面做的事情,就是把 x1 x2 x3,分别乘上 w31 w32 跟 w33 再加上 b3

我们现在為了**简化**起见,我们把括弧裡面的数字,用一个比较简单的符号来表示,所以这一串东西我们当作 r1,这一串东西我们当作 r2,这一串东西我们叫它 r3.

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

这个 x1 x2 跟 x3 和 r1 r2 r3,中间的关係是什麼呢,你可以用矩阵跟向量相乘的方法,写一个比较简单的简洁的写法.我们刚才已经知道说 r1 r2 r3,也就是括弧裡面算完的结果啊,三个 Sigmoid 括弧裡面算完的结果,r1 r2 r3 跟输入的三个 Feature x1 x2 x3,它们中间的关係就是这样,把 x1 x2 x3 乘上不同的 Weight,加上不同的 Bias,也就是不同的 b 会得到不同的 r.

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

如果你熟悉线性代数的话,简化成矩阵跟向量的相乘,把 x1 x2 x3 拼在一起变成一个向量,把这边所有的 w 统统放在一起变成一个矩阵,把 b1 b2 b3 拼起来变成一个向量,把 r1 r2 r3 拼起来变成一个向量,那这是三个式子,你就可以简写成,有一个向量叫做 x,这个 x 乘 1个矩阵叫做 w,这个 w 裡面有 9 个数值就是这边的 9 个 w,就是这边的 9 个 Weight,x 先乘上 w 以后再加上 b 就得到 r 这个向量,那**这边做的事情跟上边做的事情是一模一样**的,没有半毛钱的不同,只是表示的方式不一样而已,

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

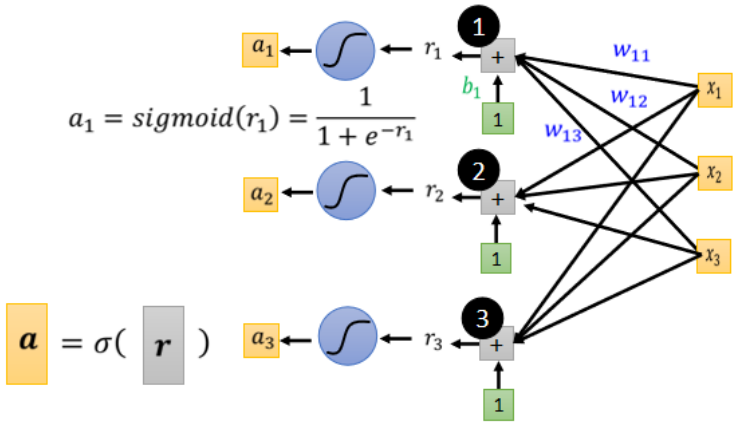
$$\begin{aligned} r_1 &= b_1 + w_{11}x_1 + w_{12}x_2 + w_{13}x_3 \\ r_2 &= b_2 + w_{21}x_1 + w_{22}x_2 + w_{23}x_3 \\ r_3 &= b_3 + w_{31}x_1 + w_{32}x_2 + w_{33}x_3 \end{aligned}$$

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

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

那把它改成线性代数比较常用的表示方式,x 乘上矩阵 w 再加上向量 b,会得到一个向量叫做 r

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

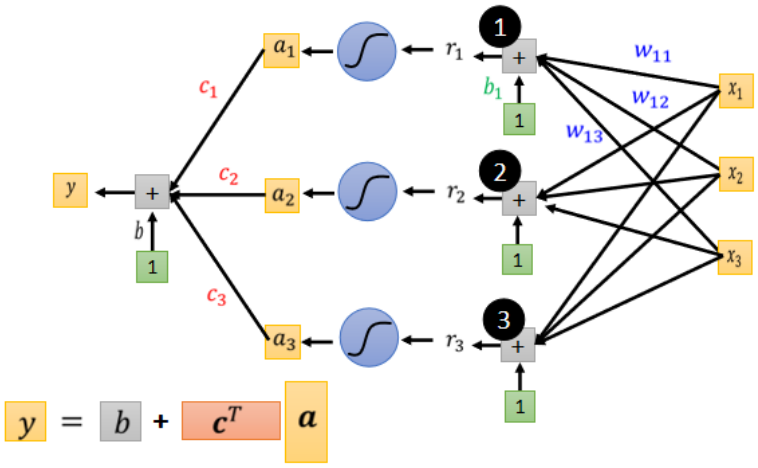


在这个括号裡面做的事情就是这麼一回事,把 x 乘上 w 加上 b 等於 r,r 呢就是这边的 r1 r2 r3,这是 r1 r2 r3,好 那接下来这个 r1 r2 r3 哪,就要分别通过 Sigmoid Function,好 分别通过 Sigmoid Function,因為我们实际上做的值就是,做的事情就是把 r1 取一个负号,再乘 再做 Exponential 再加 1,然后把它放到分母的地方,1 除以 1+Exponential 负 r1 等於 a1,然后同样的方法由 r2 去得到 a2,把 r3 透过 Sigmoid Function 得到 a3,所以这边这个蓝色的虚线框框裡面做的事情,就是从 x1 x2 x3 得到了 a1 a2 a3,

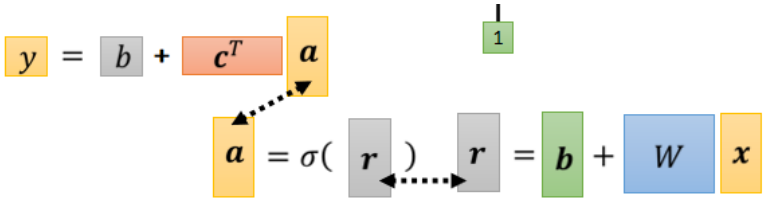
$$\boldsymbol{a} = \sigma(\boldsymbol{r})$$

接下来呢,我们这边呢有一个简洁的表示方法,是我们用 r 通过一个,叫做这个 Sigmoid 的 Function,我们用这个东西,我们这边呢用这个符号呢,来代表通过这个 Sigmoid 的 Function,然后呢 所以我们得到了 a 这个向量,就把 r1 r2 r3 分别通过 Sigmoid Function,但我们直接用这个符号来表示它,然后得到 a1 a2 a3

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



接下来我们这个 Sigmoid 的输出,还要乘上 ci 然后还要再加上 b,如果你要用向量来表示的话,a1 a2 a3 拼起来叫这个向量 a,c1 c2 c3 拼起来叫一个向量 c,那我们可以把这个 c 呢,作 **Transpose**,好 那 a 呢 乘上 c 的 Transpose 再加上 b,好再加上 b 我们就得到了 y



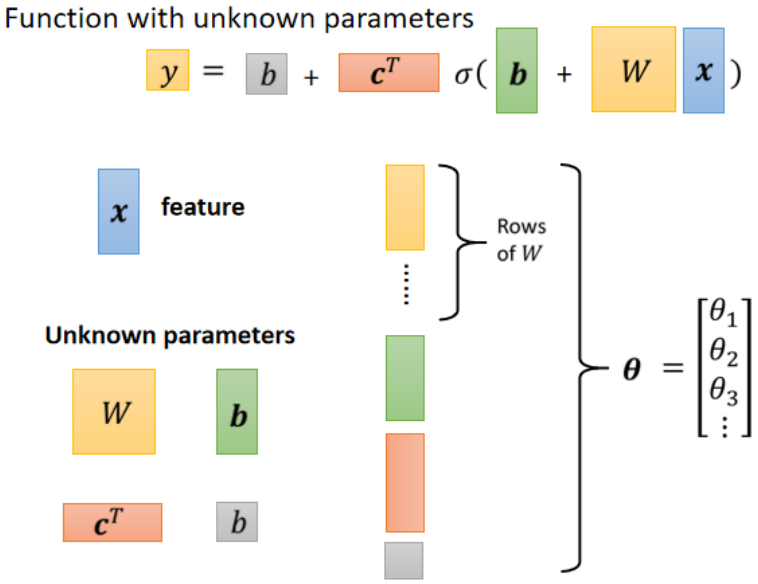
它整体而言做的事情就是 x 输入是 x,我们的 Feature 是 x 这个向量,x 乘上矩阵 w 加上向量 b 得到向量 r,再把向量 r 透过 Sigmoid Function得 到向量 a,再把向量 a 跟乘上 c 的 Transpose 加上 b 就得到 y。

所以这是上面这件事情,如果你想要用**线性代数**的方法来表示它,用向量矩阵相乘方法来表示它,欸 就长得一副这个样子,那这边的这个 r 就是这边的 r,这边的 a 就这边的 a,所以我们可以把这一串东西,放到这个括号裡面,再把这个 a 呢 放到这裡来,所以把相同的东西併起来以后,整体而言就是长这个样子,上面这一串东西,我们觉得比较这个,比较有弹性的这个 Function,如果你要线性代数来表示它的话,就是下面这个式子啦

$$y = b + c^T \sigma(b + Wx)$$

x 乘上 w 再加上 b 通过 Sigmoid Function,乘上 c 的 Transpose 加 b 就得到 y

接下来，怎麼把这些未知的参数找出来之前,我们先再稍微**重新定义一下我们的符号**,



这边的这个 x 是 Feature,这边的 W b c 跟 b,这边有两个 b 啊,但是**这两个 b 是不一样**的,**绿色**这一个是一个**向量**,**灰色**这个是一个**数值**,显示它们是不一样的东西

我们把这个黄色的这个 w,把这个 b 把这个 c 把这个 b 统统拿出来,集合在这边,它们就是我们的 Unknow 的 Parameters,就是我们的未知的参数

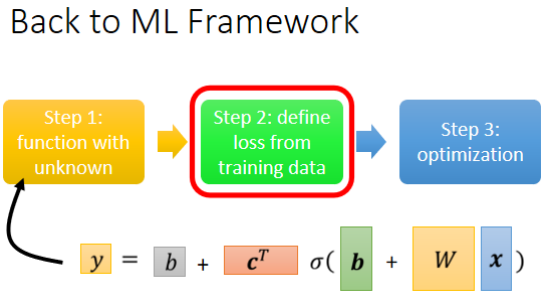
那我们把**这些东西通通拉直**,**拼成一个很长的向量**,我们把 w 的每一个 Row,或者是每一个 Column 拿出来,今天不管你是拿过 Row 或拿 Column 都可以,你就把 w 的每一个 Column 或每一个 Row 拿出来,拼成一个长的向量,把 b 拼上来 把 c 拼上来 把 b 拼上来,这个长的向量,我们直接用一个符号叫做 θ 来表示它

θ 是一个很长的向量,裡面的第一个数值我们叫 $\theta 1$,第二个叫 $\theta 2$ 这个叫 $\theta 3$,那 θ裡面,这个向量裡面有一些数值是来自於这个**矩阵**,有些数值是来自於 b,有些数值来自於 c,有些数值来自於这边这个 b,那我们就不分了,反正θ**它统称我们所有的未知的参数,我们就一律统称** θ

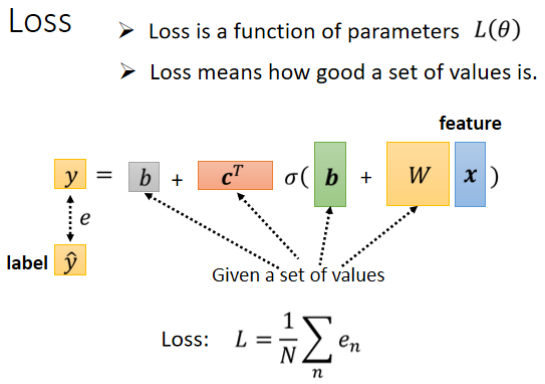
Q&A

1. 我试著回答看看,我猜他的问题是说,我们其实要做 Optimization 这件事,找一个可以让 Loss 最小的参数,有一个最**暴力的方法**就是,爆收所有可能的未知参数的值对不对,像我们刚才在只有 w 跟 b 两个参数的前提之下,我根本就可以爆收所有可能的 w 跟 b 的值嘛,所以在**参数很少**的情况下,你不 甚至你有可能不用 Gradient Descent,不需要什麼 Optimization 的技巧,但是我们今天参数很快就会变得非常多,像在这个例子裡面参数有一大把,有 w b 有 c 跟 b 串起来,变成一个很长的向量叫 θ,那这个时候你就不能够用爆收的方法了,你需要 Gradient Descent 这样的方法,来找出可以让 Loss 最低的参数
2. 这位同学的问题是说,刚才的例子裡面有三个 Sigmoid,那為什麼是三个呢,能不能够四个 五个 六个呢,可以 Sigmoid 的数目是你自己决定的,而且 Sigmoid 的数目越多,你可以產生出来的,Piecewise Linear 的 Function 就越复杂,就是假设你只有三个 Sigmoid,意味著你只能產生三个线段,但是假设你有越多 Sigmoid,你就可以產生有越多段线的,Piecewise Linear 的 Function,你就可以逼近越复杂的 Function,但是至於要几个 Sigmoid,这个又是另外一个 Hyper Parameter,这个你要自己决定,我们在刚才例子裡面举三个,那只是一个例子,也许我以后不应该举三个,因為这样会让你误以為说,Input Feature 是三个,Sigmoid也是三个,不是就是说,Sigmoid 几个可以自己决定
3. Hard 的 Sigmoid,首先它的 Function 你写出来可能会比较复杂,你一下子写不出它的 Function,但如果你可以写得出它的 Function 的话,你其实也可以用 Hard Sigmoid,你想要用也可以,所以不是一定只能够用,刚才那个 Sigmoid 去逼近那个 Hard Sigmoid,完全有别的做法,等一下我们就会讲别的做法

Back to ML_Step 2 :define loss from training data



那接下来进入第二步了,我们要定 Loss,有了新的这个 Model 以后,我们 Loss 没有什麼不同,定义的方法是一样的,只是我们的符号改了一下,之前是 L (w 跟 b),因為 w 跟 b 是未知的,那我们现在接下来的未知的参数很多了,你再把它一个一个列出来,太累了,所以我们直接**用 θ 来统设所有的参数**,所以我们现在的 Loss Function 就变成 $L(\theta)$

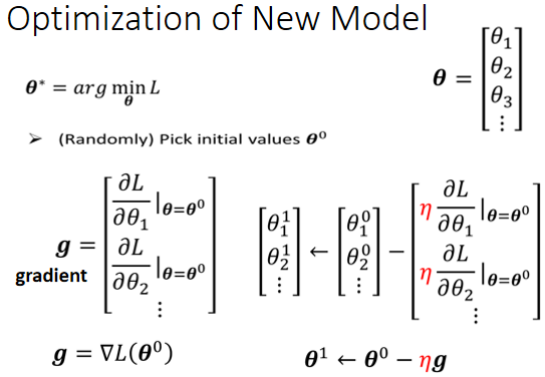


这个 Loss Function 要问的就是,这个 θ 如果它是某一组数值的话,会有多不好或有多好,那计算的方法,**跟刚才只有两个参数的时候,其实是一模一样的**

- 先给定某一组 W b c^T 跟 b 的值,你先给定某一组 θ的值,假设你知道 w 的值是多少,把 w 的值写进去 b 的值写进去,c 的值写进去 ,b 的值写进去
- 然后把一种 Feature x 带进去,然后看看你估测出来的 y 是多少
- 再计算一下跟真实的 Label 之间的差距,你得到一个 e
- 把所有的误差通通加起来,你就得到你的 Loss

Back to ML_Step 3: Optimization

接下来下一步就是 Optimization,Optimization跟前面讲的没有什麼不同 还是一样的,所以就算我们换了一个新的模型,这个 Optimization 的步骤.



我们现在的 **θ 它是一个很长的向量**,我们把它表示成 θ1 θ2 θ3 等等等,我们现在就是要**找一组 θ,这个 θ 可以让我们的 Loss 越小越好**,可以让 Loss 最小的那一组 θ,我们叫做 θ 的 Start θ^*

- 我们一开始要**随机选一个初始的数值,这边叫做** θ_0 你可以随机选,那之后也可能会讲,也会讲到更好的找初始值的方法,我们现在先随机选就好
- 接下来呢你要**计算微分**,你要对每一个未知的参数,这边用 θ1 θ2 θ3 来表示,你要**為每一个未知的参数,都去计算它对 L 的微分**,那把每一个参数都拿去计算对 L 的微分以后,集合起来它就是一个向量,那

个向量我们用 g 来表示它,这边假设有 1000 个参数,这个向量的长度就是 1000,这个向量裡面就有 1000 个数字,这个东西有一个名字,,这个向量有一个名字叫做 Gradient,那很多时候你会看到,Gradient 的表示方法是这个样子的,你把 L 前面放了一个**倒三角形**,这个就代表了 Gradient,这是一个 Gradient 的简写的方法,那其实我要表示的就是这个向量,L 前面放一个倒三角形的意思就是,把所有的参数 θ1 θ2 θ3,通通拿去对 L 作微分.那后面放 θ0 的意思是说,我们这个算微分的位置,是在 θ 等於 θ0 的地方,在 θ 等於 θ0 的地方,我们算出这个 Gradient

- 算出这个 g 以后,接下来呢我们 **Update 参数**,更新的方法,跟刚才只有两个参数的状况是一模一样的,只是从更新两个参数,可能换成更新成 1000 个参数,但更新的方法是一样的,本来有一个参数叫 θ1,**上标 0 代表它是一个起始的值**,它是一个随机选的起始的值,把这个 θ_1^0 **减掉 η 乘上微分的值,得到** θ_1^1**,代表 θ1 更新过一次的结果**,θ_2^0 减掉微分乘以,减掉 η 乘上微分的值,得到θ_2^1,以此类推,你就可以把那 1000 个参数统统都更新了.

简写把这边**所有的 θ 合起来当做一个向量,我们用** θ^0**来表示**,把 η 提出来,那剩下**每一个参数对 L 微分的部分**,叫做 **Gradient 叫做 g**,所以 θ0 减掉 η 乘上 g,就得到 θ1

$$\theta^1 \leftarrow \theta^0 - \eta g$$

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 $\boldsymbol{g} = \nabla L(\boldsymbol{\theta}^2)$

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

θ0 减掉 θ0 这个向量,减掉 η 乘上 g,g 也是一个向量会得到 θ1,那假设你这边参数有 1000 个,那 θ0 就是 1000 个数值,1000 微的向量,g 是1000 微的向量,θ1 也是 1000 微的向量

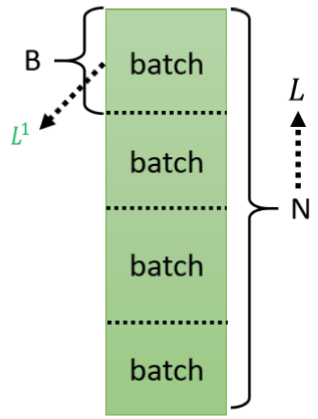
那整个操作就是这样了,就是由 θ0 算 Gradient,根据 Gradient 去把 θ0 更新成 θ1,然后呢再算一次 Gradient,然后呢根据 Gradient 把 θ1 再更新成 θ2,再算一次 Gradient 把 θ2 更新成 θ3,以此类推直到你不想做,或者是你算出来的这个 Gradient,是 0 向量 是 Zero Vector,导致你没有办法再更新参数為止,不过在实作上你几乎不太可能,作出 Gradient 是 0 向量的结果,通常你会停下来就是你不想做了.

Optimization of New Model

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

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

$$\boldsymbol{\theta}^1 \leftarrow \boldsymbol{\theta}^0 - \eta \boldsymbol{g}$$

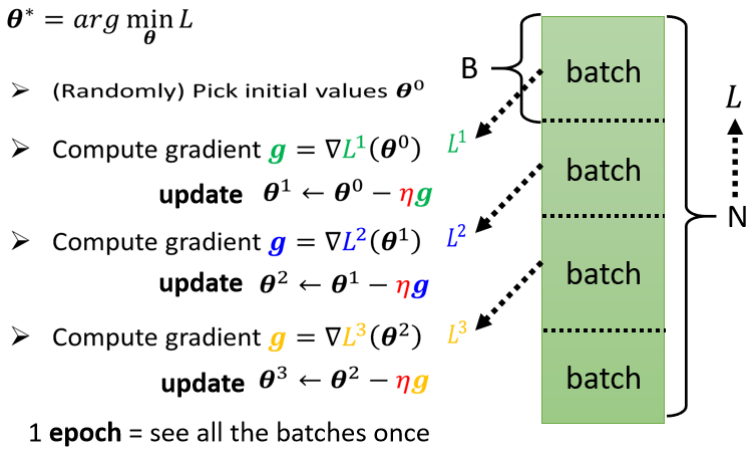


但是实作上,那这边是一个实作的 Detail 的 Issue,实际上我们在做 Gradient的时候,我们会这麼做

我们这边有大 N 笔资料,我们会把这大 N 笔资料分成一个一个的 Batch,就是一包一包的东西 一组一组的,怎麼分,随机分就好

所以每个 Batch 裡面有大 B 笔资料,所以本来全部有大 N 笔资料,现在大 B 笔资料一组,**一组叫做 Batch**

那**本来我们是把所有的 Data 拿出来算一个 Loss**,那现在我们不这麼做,我们**只拿一个 Batch 裡面的 Data出来算一个 Loss**,我们这边把它叫 L1,那跟这个 L 呢以示区别,因为你把全部的资料拿出来算 Loss,跟只拿一个 Batch 拿出来,的资料拿出来算 Loss,它不会一样嘛,所以这边用 L1 来表示它

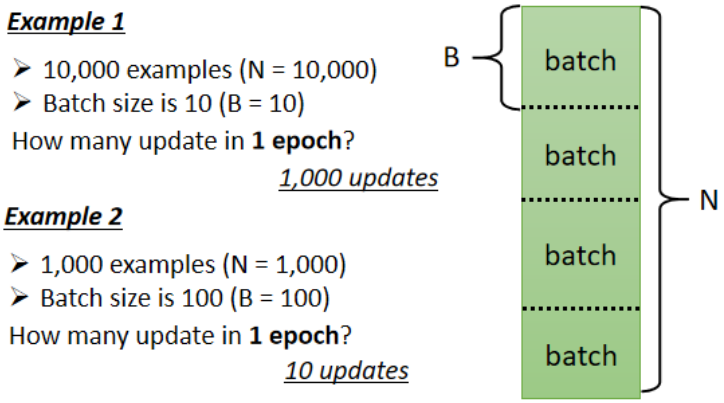


但是你可以想像说假设这个 B 够大,也许 L 跟 L1 会很接近 也说不定,所以实作上的时候,每次我们会先选一个 Batch,用这个 Batch 来算 L,**根据这个 L1 来算 Gradient,用这个 Gradient 来更新参数**,接下来再选下一个 Batch 算出 L2,根据 L2 算出 Gradient,然后再更新参数,再取下一个 Batch 算出 L3,根据 L3 算出 Gradient,再用 L3 算出来的 Gradient 来更新参数

所以我们并不是拿大 L 来算 Gradient,实际上我们是拿一个 Batch 算出来的 L1 L2 L3,来计算 Gradient,那把所有的 Batch 都看过一次,叫做一个 Epoch,**每一次更新参数叫做一次 Update**, Update 跟 Epoch 是不一样的东西

每次更新一次参数叫做一次 Update,把所有的 Batch 都看过一遍,叫做一个 Epoch

那至於为什麼要分一个一个 Batch,那这个我们下週再讲,但是为了让大家更清楚认识,Update 跟 Epoch 的差别,这边就举一个例子



假设我们有 10000 笔 Data,也就是**大 N 等於 10000**,假设我们的 **Batch 的大小是设 10**,也就大 B 等於 10

接下来问,我们在一个 Epoch 中,总共 Update 了几次参数?

那你就算一下这个大 N 个 Example,10000 笔 Example,总共形成了 10000 除以 10,也就是 **1000 个 Batch**,所以在一个 Epoch 裡面,你其实已经**更新了参数 1000 次**,所以一个 Epoch 并不是更新参数一次,在这个例子裡面一个 Epoch,已经更新了参数 1000 次了

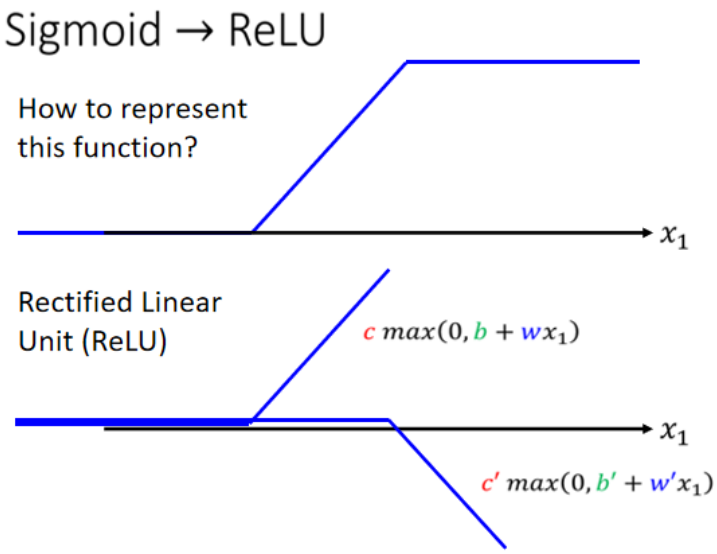
那第二个例子,就是假设有 1000 个资料,Batch Size 设 100,那其实 Batch Size 的大小也是你自己决定的,所以这边我们又多了一个 HyperParameter,所谓 HyperParameter 就是你自己决定的东西,人所设的东西不是.机器自己找出来的,叫做 HyperParameter,我们今天已经听到了,几个 Sigmoid 也是一个 HyperParameters,Batch Size 也是一个 HyperParameter,好 1000 个 Example,Batch Size 设 100,那1个 Epoch 总共更新几次参数呢,是 10 次

所以做了一个 Epoch 的训练,你其实不知道它更新了几次参数,有可能 1000 次,也有可能 10 次,**取决於它的 Batch Size 有多大**

模型变型

那我们其实还可以对模型做更多的变形,刚才有同学问到说,这个 Hard Sigmoid 不好吗,為什麼我们一定要把它换成 Soft 的 Sigmoid

你确实可以不一定要换成 Soft 的 Sigmoid,有其他的做法,举例来说这个 Hard 的 Sigmoid,我刚才说它的函式有点难写出来,其实也没有那麼难写出来,它可以看作是**两个 Rectified Linear Unit 的加总**,所谓 Rectified Linear Unit 它就是长这个样



它有一个水平的线,走到某个地方有一个转折的点,然后变成一个斜坡,那这种 Function 它的式子,写成

$$c * max(0, b + wx_1)$$

这个$max(0, b + wx_1)$的意思就是,看 0 跟 $b + wx_1$ 谁比较大,**比较大的那一个就会被当做输出**,所以如果 b + wx1 小於 0,那输出就是0,如果 $b + wx_1$ 大於 0,输出就是 $b + wx_1$

那总之这一条线,可以写成$c * max(0, b + wx_1)$,每条不同的 w 不同的 b 不同的 c,你就可以挪动它的位置,你就可以改变这条线的斜率,那这种线呢在机器学习裡面,我们叫做 **Rectified Linear Unit**,它的缩写叫做 ReLU,名字念起来蛮有趣的,它真的就唸ReLU

Sigmoid → ReLU

$$y = b + \sum_{i} c_i \, sigmoid\left(b_i + \sum_j w_{ij} x_j\right)$$

Activation function

$$y = b + \sum_{2i} c_i \, max\left(0, b_i + \sum_j w_{ij} x_j\right)$$

Which one is better?

那你把两个 ReLU 叠起来,就可以变成 Hard 的 Sigmoid,你想要用 ReLU 的话,就把 Sigmoid 的地方,换成$max(0, b_i + w_{ij}x_j)$.

那本来这边只有 i 个 Sigmoid,你要 **2 个 ReLU,才能够合成一个 Hard Sigmoid** ,所以这边有 i 个 Sigmoid,那如果 ReLU 要做到一样的事情,那你可能需要 2 倍的 ReLU,因为 2 个 ReLU 合起来,才是一个 Hard Sigmoid,所以要 2 倍的 ReLU,所以我们把 Sigmoid 换成 ReLU,这边就是把一个式子换了,因为要表示一个 Hard 的 Sigmoid,表示那个蓝色的 Function 不是只有一种做法,你完全可以用其他的做法,好 那这个 Sigmoid 或是 ReLU,他们在机器学习裡面,我们就叫它 Activation Function,他们是有名字的,他们统称为 Activation Function.

当然还有其他常见的,还有其他的 Activation Function,但 Sigmoid 跟 ReLU,应该是今天最常见的 Activation Function,那哪一种比较好呢,这个我们下次再讲,哪一种比较好呢,我接下来的实验都选择用了 ReLU,显然 ReLU 比较好,至於它為什麼比较好,那就是下週的事情了.

接下来就真的做了这个实验,这个都是真实的数据.

Experimental Results

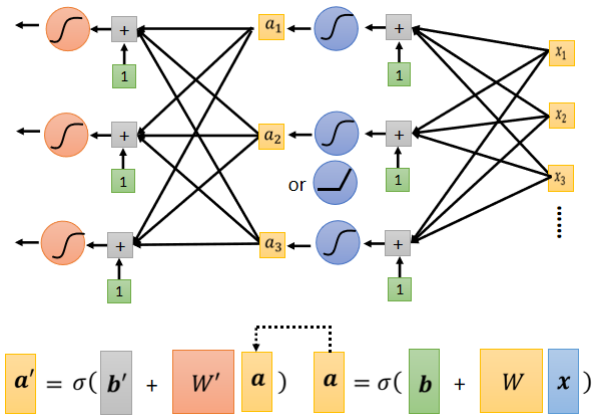
$$y = b + \sum_{2i} c_i \, max\left(0, b_i + \sum_j w_{ij} x_j\right)$$

	linear	10 ReLU	100 ReLU	1000 ReLU
2017 – 2020	0.32k	0.32k	0.28k	0.27k
2021	0.46k	0.45k	0.43k	0.43k

- 如果是 Linear 的 Model,我们现在考虑 56 天,训练资料上面的 Loss 是 0.32k,没看过的资料 2021 年资料是 0.46k
- 如果用 10 个 ReLU,好像没有进步太多,这边跟用 Linear 是差不多的,所以看起来 10 个 ReLU 不太够
- 100 个 ReLU 就有显著的差别了,100 个 ReLU 在训练资料上的 Loss,就可以从 0.32k 降到 0.28k,有 100 个 ReLU,我们就可以製造比较复杂的曲线,本来 Linear 就是一直线,但是 100 个 ReLU 我们就可以產生 100 个,有 100 个折线的Function,在测试资料上也好了一些.
- 接下来换 1000 个 ReLU,1000 个 ReLU,在训练资料上 Loss 更低了一些,但是在没看过的资料上,看起来也没有太大的进步

多做几次

接下来还可以做什麼呢,我们还可以继续改我们的模型,



举例来说,刚才我们说从 x 到 a 做的事情,是把 x 乘上 w 加 b,再通过 Sigmoid Function,不过我们现在已经知道说,不一定要通过 Sigmoid Function,通过 ReLU 也可以,然后得到 a。

我们可以把这个同样的事情,再反覆地多做几次,刚才我们把 w x 乘上 w 加 b,通过 Sigmoid Function 得到 a,我们可以把 a 再乘上另外一个 w',再加上另外一个 b',再通过 Sigmoid Function,或 RuLU Function,得到 a'

所以我们可以把 x,做这一连串的运算產生 a,接下来把 a 做这一连串的运算產生 a',那我们可以反覆地多做几次,**要做几次,这个又是另外一个 Hyper Parameter**,这是另外一个你要自己决定的事情,你要做两次吗 三次吗 四次吗 一百次吗,这个你自己决定,不过这边的 w 跟这边的 w',它们不是同一个参数喔,这个 b 跟这边的 b',它们不是同一个参数喔,是增加了更多的未知的参数

那就是接下来就真的做了实验了,我们就是每次都加 100 个 ReLU,那我们就是 Imput Features,就是 56 天前的资料

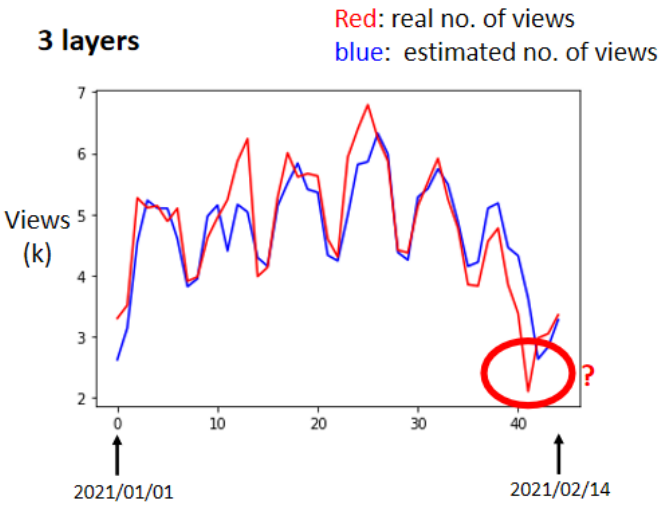
Experimental Results

- 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
2017 – 2020	0.28k	0.18k	0.14k
2021	0.43k	0.39k	0.38k

- 如果是只做一次 只做一次,就那个乘上 w 再加 b,再通过 ReLU 或 Sigmoid,这件事只做一次的话,这是我们刚才看到的结果
- 两次，这个 Loss 降低很多,0.28k 降到 0.18k,没看过的资料上也好了一些
- 三层，又有进步,从 0.18k 降到 0.14k,所以从一层到 从就是乘一次 w,到通过一次 ReLU,到通过三次 ReLU,我们可以从 0.28k 到 0.14k,在训练资料上,在没看过的资料上,从 0.43k 降到了 0.38k,看起来也是有一点进步的,

那这个是那个真实的实验结果啦,就我们来看一下,今天有做通过三次 ReLU 的时候,做出来的结果怎麼样



横轴就是时间,纵轴是观看的人次 是千人,**红色的线代表的是真实的数据,蓝色的线是预测出来的数据**

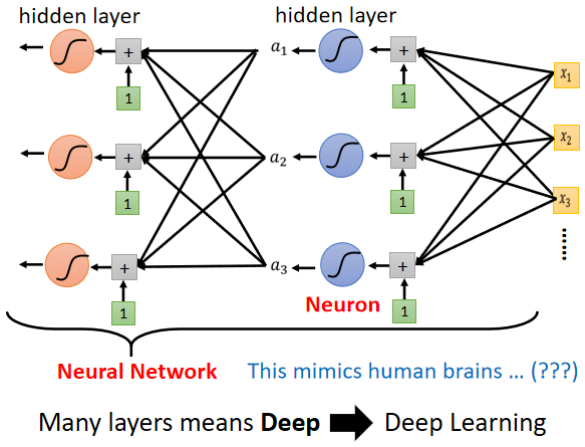
那你会发现说,在这种低点的地方啊,你看红色的数据是每隔一段时间,就会有两天的低点,在低点的地方,机器的预测还算是蛮準确的,

那这边有一个神奇的事情,这个机器高估了真实的观看人次,尤其是在这一天,这一天有一个很明显的低谷,但是机器没有预测到这一天有明显的低谷,它是晚一天才预测出低谷

这天最低点就是除夕啊,谁除夕还学机器学习 对不对,好 所以当然对机器来说,你不能怪它,它根本不知道除夕是什麼,它只知道看前 56 天的值,来预测下一天会发生什麼事,所以它不知道那一天是除夕,所以你不能怪它预测地不準,这一天就是除夕

好名字

到目前為止,我们讲了很多各式各样的模型,那我们现在还缺了一个东西,缺一个好名字,你知道这个外表啊是很重要的,一个死臭酸宅穿上西装以后就潮了起来,或者是隻鞋半缕的,说他是汉左将军宜城亭侯中山靖王之后,也就潮了起来 ,所以我们的模型也需要一个好名字,所以它叫做什麼名字呢,这些 Sigmoid 或 ReLU 啊,它们叫做 Neuron,我们这边有很多的 Neuron,很多的 Neuron 就叫做 Neural Network,Neuron 就是神经元,人脑中就是有很多神经元,很多神经元串起来就是一个神经网路,跟你的脑是一样的,接下来你就可以到处骗麻瓜说,看到没有 这个模型就是在模拟人们脑 知道吗,这个就是在模拟人脑,这个就是人工智慧,然后麻瓜就会吓得把钱掏出来



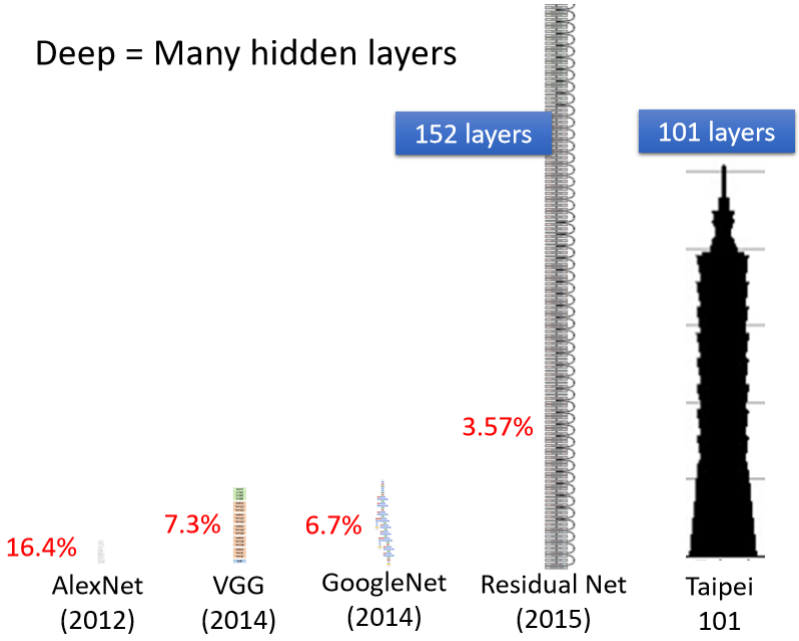
Many layers means **Deep** ➡ Deep Learning

但是啊 这个把戏在 80 90 年代的时候,已经玩过了这样,Neural Network 不是什麼新的技术,80 90 年代就已经用过了,当时已经把这个技术的名字搞到臭掉了,Neural Network 因為之前吹捧得太过浮夸,所以后来大家对 Neural Network 这个名字,都非常地感冒,它就像是个脏话一样,写在 Paper 上面都註定会被,就会註定害你的 Paper 被拒绝, 所以后来為了要重振 Neural Network 的雄风,所以怎麼办呢,需要新的名字,怎麼样新的名字呢,这边有很多的 Neural,每一排 Neural 我们就叫它一个 Layer,它们叫 Hidden Layer,

有很多的 Hidden Layer 就叫做 Deep,这整套技术就叫做 Deep Learning,好 我们就把 Deep Learning 讲完了,就是这麼回事。

Deep

就是这样来的,好 所以人们就开始,把类神经网路越疊越多 越疊越深,12 年的时候有一个 AlexNet,它有 8 层 它的错误率是 16.4%,两年之后 VGG 19层,错误率在影像辨识上进步到 7.3 %,这个都是在影像辨识上一个,这个基準的资料库上面的结果,后来 GoogleNet 有错误率降到 6.7%,有 22 层,但这些都不算是什麼



Residual Net 有 152 层啊,它比 101 还要高啊,但是这个 Residual Net 啊,其实要训练这麼深的 Network 是有诀窍的,这个我们之后再讲。

如果你仔细思考一下,我们一路的讲法的话,你有没有发现一个奇妙的违和的地方,我们一开始说,我们想要用 ReLU 或者是 Sigmoid,去逼近一个复杂的 Function,实际上只要够多的 ReLU 够多的 Sigmoid,就可以逼近任何的 连续的 Function ,我们只要有够多的 Sigmoid,就可以知道够复杂的线段,就可以逼近任何的 Continuous 的 Function,所以我们只要一排 ReLU 一排 Sigmoid,够多就足够了,**那深的意义到底何在呢**

把 ReLU Sigmoid Function 反覆用,到底有什麼好处呢,為什麼不把它们直接排一排呢,直接排一排也可以表示任何 Function 啊,所以把它反覆用没什麼道理啊,所以有人就说把 Deep Learning,把 ReLU Sigmoid 反覆用,不过是个噱头,你之所以喜欢 Deep Learning,只是因為 Deep 这个它名字好听啦,ReLU Sigmoid 排成一排,你只可以製造一个肥胖的 Network,Fat Neural Network,跟 Deep Neural Network 听起来,量级就不太一样,Deep 听起来就比较厉害啦,Fat Neural Network 还以為是死肥宅 Network,就不厉害这样子,那到底 Deep 的理由,為什麼我们不把 Network 变胖,只把 Network 变深呢,这个是我们日后要再讲的话题

那有人就说,那**怎麼不变得更深呢**,刚才只做到 3 层,应该要做得更深嘛,现在 Network 都是叠几百层的啊,没几百层都不好意思说,你在叫做 Deep Learning 所以要做更深,

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

所以确实做得更深 做 4 层,4 层在训练资料上,它的 Loss 是 0.1k,在没有看过 2021 年的资料上,是如何呢 是 0.44k,惨掉了。在训练资料上,3 层比 4 层差,4 层比 3 层好,但是在没看过的资料上,4 层比较差,3 层比较好,在有看过的资料上,在训练资料上,跟没看过的资料上,它的结果是不一致的,这种训练资料跟测试,这种训练资料跟没看过的资料,它的结果是不一致的状况,这个状况叫做 Overfitting,

机器学习会发生 **Overfitting** 的问题,指的就是**在训练资料上有变好,但是在没看过的资料上没有变好**这件事情,

但是做到目前为止,我们都还没有真的发挥这个模型的力量,你知道我们要发挥这个模型的力量,和 2021 的资料到 2 月 14 号之前的资料,我们也都已经手上有了

Let's predict no. of views today!

- If we want to select a model for predicting no. of views today, which one will you use?

	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

We will talk about model selection next time. ☺

所以我们要真正做的事情是什麼,我们要做的事情就是预测未知的资料,但是如果我们要预测未知的资料,我们应该选 3 层的 Network,还是 4 层的 Network 呢,举例来说 今天是 2 月 26 号,今天的观看人数我们还不知道,如果我们要用一个 Neural Network,用我们已经训练出来的 Neural Network,去预测今天的观看人数,

至於怎麼选模型,这个是下週会讲的问题,大家 但是大家都非常有 Sense,知道我们要选 3 层的,多数人都决定要选 3 层的,你可能会说 我怎麼不选 4 层呢,4 层在训练资料上的结果比较好啊,**可是我们并不在意训练资料的结果啊,我们在意的是没有看过的资料**,而 2 月 26 号是没有看过的资料,我们应该选一个在训练的时候,没有看过的资料上表现会好的模型,所以我们应该选 3 层的 Network

那你可能以為这门课就到这边结束了,其实不是 我们真的来预测一下,2 月 26 号应该要有的观看次数是多少,但是因為其实 YouTube 的统计,它没有那麼及时,所以它现在只统计到 2 月 24 号,没关係 我们先计算一下 2 月 25 号的,观看人数是多少,这个 3 层的 Network 告诉我说,2 月 25 号这个频道的总观看人次,应该是 5250 人,那我们先假设 2 月 25 号是对的,但实际上我还不知道 2 月 25 号对不对,因為 YouTube 后台统计的数据还没有出来啊,但我们先假设这一天都是对的,然后再给我们的模型去预测 2 月 26 号的数字,得到的结果是 3.96k 有 3960 次,那它為什麼这边特别低,因為模型知道说,这个礼拜五观看的人数,就是比较少啊,所以它预测特别低,听起来也是合理的

To learn more ……

Basic Introduction

https://youtu.be/Dr-WRlEFefw

Backpropagation
Computing gradients in an efficient way

https://youtu.be/ibJpTrp5mcE

好 那今天其实就讲了深度学习,那今天讲的不是一般的介绍方式,如果你想要听一般的介绍方式,过去的课程影片也是有的,我就把连结附在这边,然后深度学习的训练,会用到一个东西叫 Backpropagation,其实它就是比较有效率,算 Gradients 的方法,跟我们今天讲的东西没有什麼不同,但如果你真的很想知道,Backpropagation 是什麼的话,影片连结也附在这边.