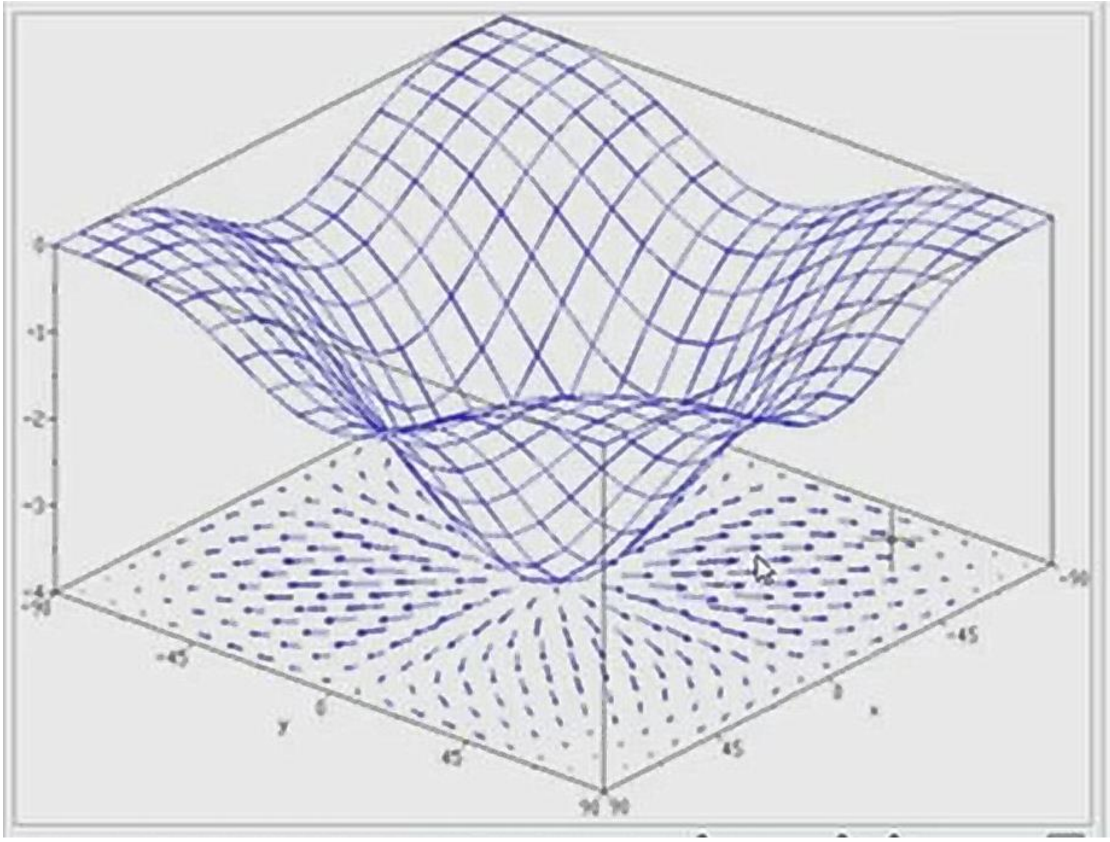
梯度：它就是將每個變數的偏微分集合在一起，將它們變成一個向量，若以圖像顯示的話，箭號越長它所代表的斜率越大。



梯度下降法：朝著斜率最大的方向走；朝著斜率大的方向<正梯度>走，會愈走愈高；相反的，就為<逆梯度>，就會愈走愈低。

梯度下降法的缺點：在變數多的時候，梯度的計算速度會非常慢，因為它每計算一次都要計算一次偏微分。

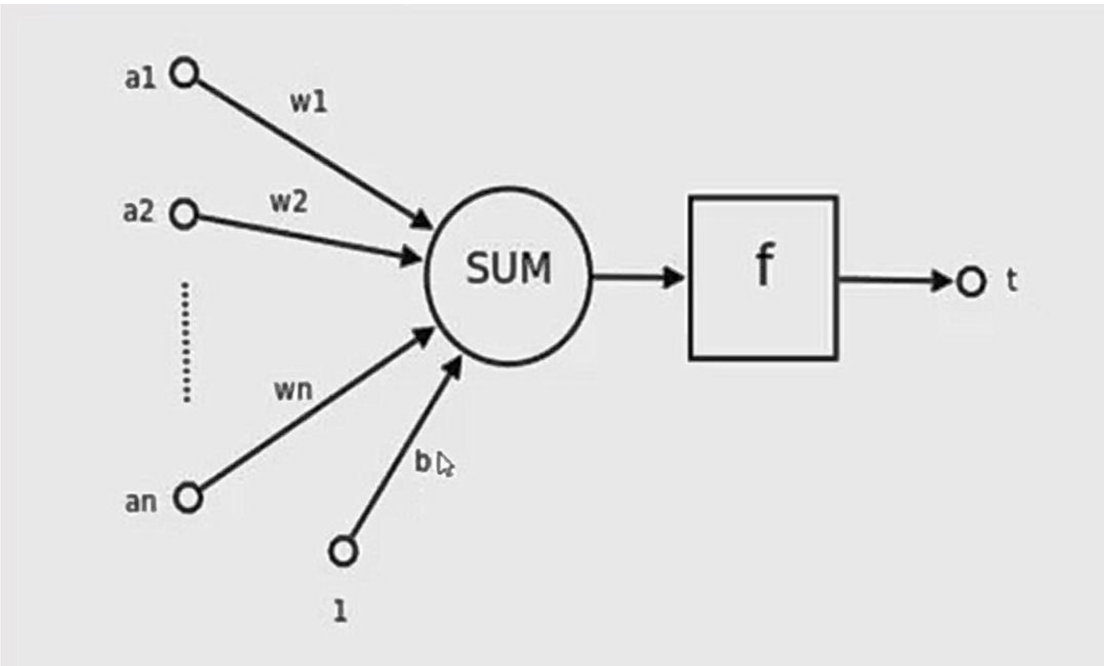
梯度下降法與神經網路的關係：以 AND 閘為

例，先將 AND 的真值表列出來，再列出未知數減真值表的算式，再利用梯度下降法來求出它的最小值，最後即可算出值出來。

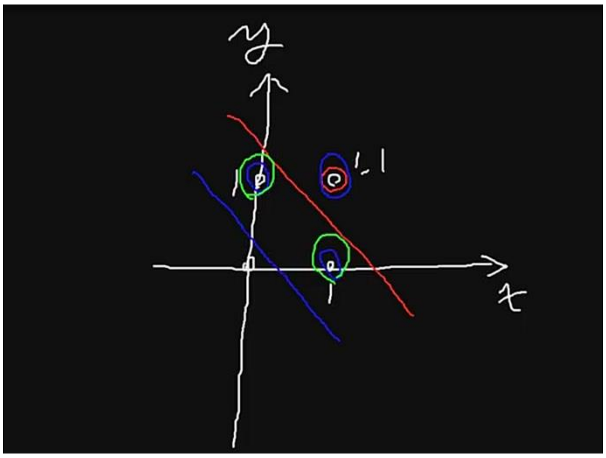
```
AND gate
x   y | o = and(x,y)
------|---
0   0 | 0     x0 y0 o0
0   1 | 0     x1 y1 o1
1   0 | 0     x2 y2 o2
1   1 | 1     x3 y3 o3

w1*x + w2*y + b = o
w1*0 + w2*0 + b = o0   => 0
w1*0 + w2*1 + b = o1   => 0
w1*1 + w2*0 + b = o2   => 0
w1*1 + w2*1 + b = o3   => 1

f(x,y,w) = (o0-0)^2 + (o1-0)^2 + (o2-0)^2 + (o3-1)^2
```



(單一神經元圖例)



(簡易圖解 為何 XOR 無法用單層神經網路做出來)

反傳遞演算法：上面提到梯度下降法的缺點就是計算非常慢，因此就需要反傳遞演算法的幫助了，它的規則基本上就是反向的偏微分，首先先設為代號，然後再用等候右邊的值向左偏微分回去，最後就可以得到反傳遞的值，所有的偏微分和起來就是它的梯度。

這樣我們就可以寫出下列兩組關係式：

$$g_f^z = g_f^q * g_q^z$$

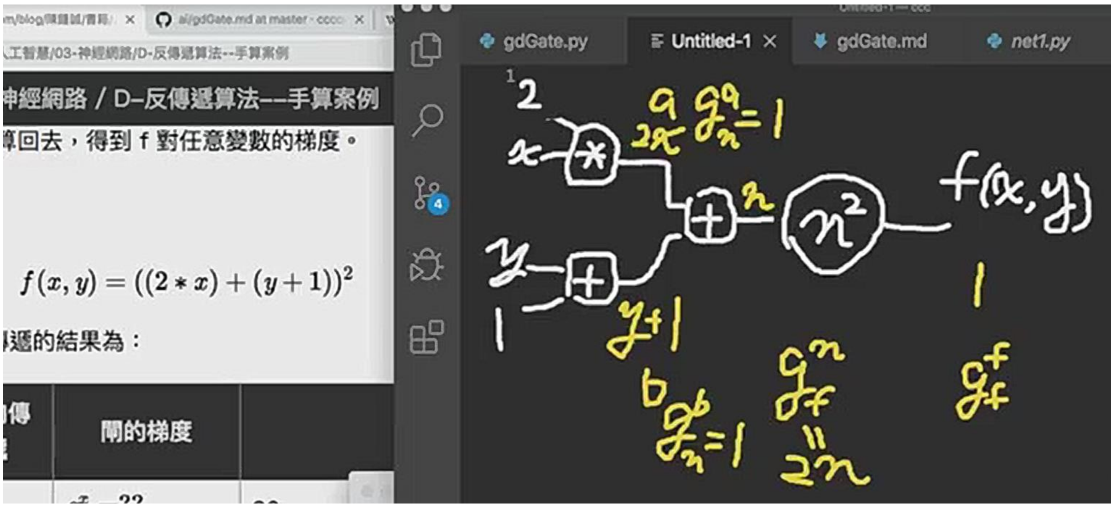
$$g_f^y = g_f^q * g_q^y$$

由於 f=q*z, q=x+y ，因此我們可以計算出下列算式：

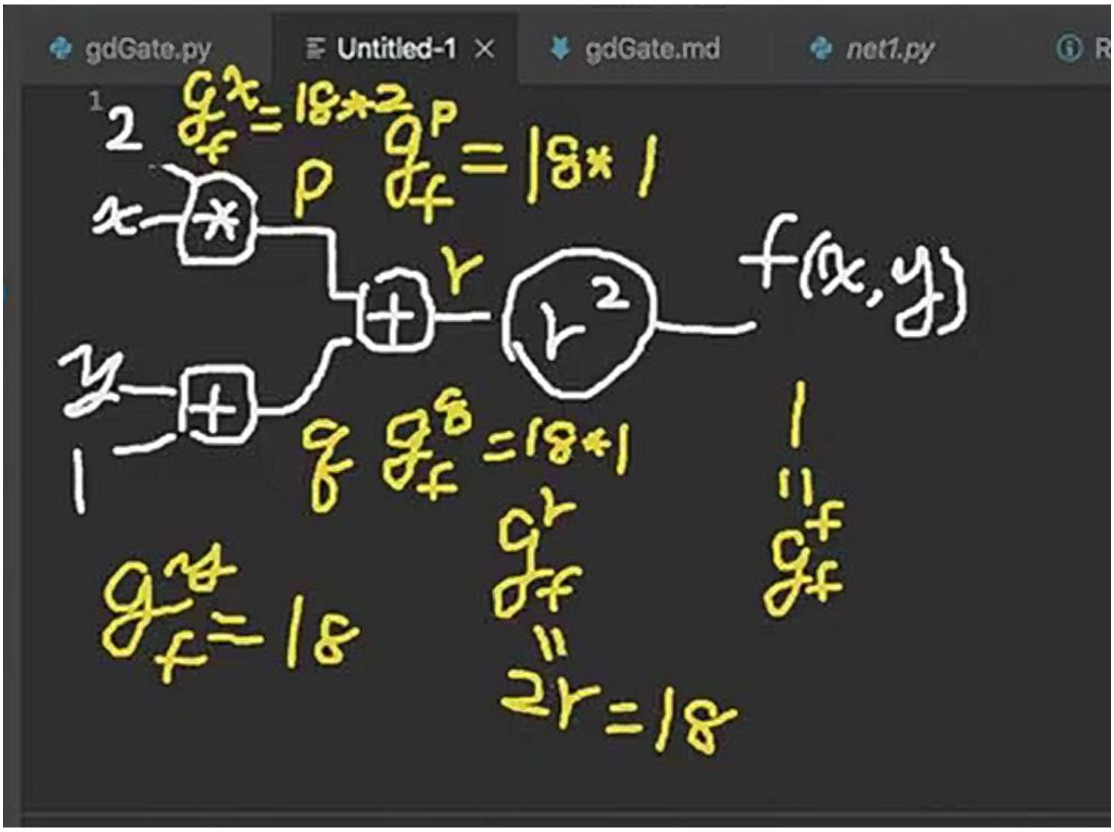
$$g_f^q = z$$

$$g_q^z = 1$$

$$g_q^y = 1$$



運算式	正向傳遞	閘的梯度	反向傳遞
x = 3	x=3	$g^x_f = ??$	36
y = 2	y=2	$g^y_f = ??$	18
p = 2x	p=6	$g^x_p = 2$	$g^x_f = g^p_f * g^x_p = 18 * 2 = 36$
q = y+1	q=3	$g^y_q = 1$	$g^y_f = g^q_f * g^y_q = 18 * 1 = 18$
r = p+q = 2x+y+1	r=9	$g^q_r = 1$; $g^p_r = 1$	$g^q_f = g^r_f * g^q_r = 18 * 1$; $g^p_f = g^r_f * g^p_r = 18 * 1$
$f = r * r = (2x + y + 1)^2$	f=9*9	$g^r_f = 2r = 18$	$g^r_f = g^r_f * g^f_f = 18$
f = f	f=81		$g^f_f = 1$



深度優先搜尋法(Depth-First Search)：一條路會一直走下去，不管旁邊有沒有其他路，優點是速度快，但不易找到最佳解。

```
'''
dfs:1 => 2 => 3 => 4 => 5 => 6 =>
stack: 存在函數呼叫自動產生的堆疊中，並沒有一個外顯變數存放堆疊。
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
'''
```

廣度優先搜尋 (Breath-First Search)：優先把附近的路先走過，雖然速度較慢，但一定會找到最佳路徑。

```
bfs:1 => 2 => 5 => 3 => 4 => 6 =>
queue:
1
1 2 5
2 5 3 4
5 3 4 6
3 4 6
4 6
6
```

資料來源：都是來自上課的影片跟老師的程式還有老師的網站