

# 機器 深度 學習筆記

呂芳元 [dan59314@gmail.com](mailto:dan59314@gmail.com)

2018 / 1 / 17

# 內容

決策樹 Decision Tree .....	3
神經元 Neuron .....	4
神經網路 Neural Network .....	5
梯度下降 Gradient Decend: .....	6
二次 Cost 和 Error ( Quadratic Cost) .....	6
Cost 公式解說 .....	7
交叉熵 Cost 和 Error (CrossEntropyCost) .....	9
反向傳播 BackPropagation .....	10
二次 Cost (QuadraticCost) .....	10
交叉熵 Cost (CrossEntropyCost) .....	10
訓練停滯 Overfitting .....	11
矯正參數 Regularization : .....	11
L1 / L2 Regualrization: .....	11
DropOut Regularization: .....	12
神經網路參數影響結果 .....	13
Cost Function 的影響 .....	13
Regularization ( lmbda) 的影響 .....	13
神經網路層數的影響 .....	14
權重初始化方式的影響 .....	15
提高準確度 Accuracy 的方法 : .....	17
增加參數與函數 : .....	17
神經網路參數(hyper-parameters)的選取 : .....	17
自動產生更多訓練資料 .....	18
權重偏向初始化 .....	19
消失的梯度 Vanish Gradient .....	21
公式 .....	21
解決方法 .....	22
CNN 網路(Convolution Neural Network) .....	23
多 Channel, 多 Filter 卷積神經網路(CNN) .....	24
準備工作 .....	25
數學教學 .....	26

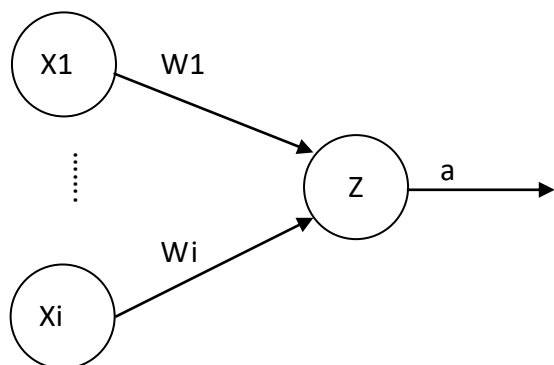
# 決策樹 Decision Tree

# 神經元 Neuron

神經元 是一個有多個輸入和輸出的單元，類似人腦的神經元作用。

電腦的神經元 是以多元一次線性方程式表示。

經由輸入  $x$  乘以權重  $w$ ，加總所有  $x_i * w_i$  後加上偏差值  $bias$ ，計算出該神經元的數值。



$$Z = w_1 * x_1 + w_2 * x_2 + w_3 * x_3 + \dots + w_i * x_i + b$$

$$= \text{Sum} ( w_i * x_i ) + b$$

$$a = \text{Sigmoid} ( Z )$$

$x_i$ : 輸入值。

$w_i$ : 輸入值  $x_i$  的權重。

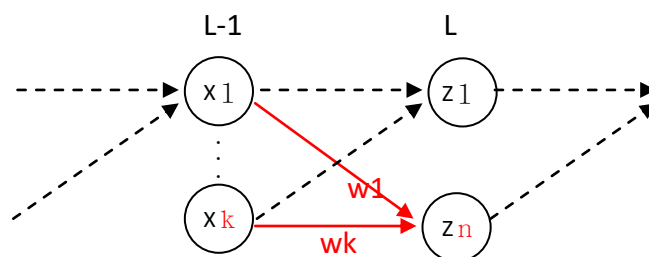
$Z$ : 每個神經元由前一層輸入  $\text{Sum}(w_i * x_i)$  加上偏向  $b$  的和。

$a$ : 每個神經元的最終激活值，Sigmoid Value。

因此，每個擁有多個輸入  $x, w$  的神經元值可以下列公式表示。

$$Z (L,n) = \text{Sum} ( w (n,k) * a (L-1,k) + b (L,n) )$$

$$a (L,n) = \text{Sigmoid} ( Z (L,n) )$$



$w (n,k)$  : 當前層  $L$  第  $n$  個神經元的第  $k$  個輸入(前一層的神經元)的權重。

$a (L-1,k)$  : 當前層  $L$  第  $n$  個神經元的第  $k$  個輸入(前一層的神經元)的激活值。

$b (L,n)$  : 當前層  $L$  第  $n$  個神經元的偏向值。

$L$ : 當前層， $L-1$  前一層。

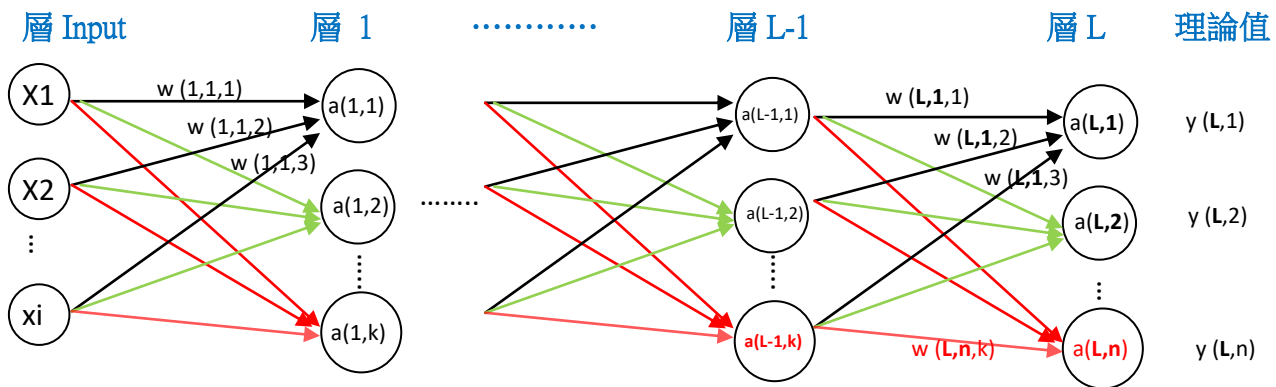
$n$ : 當前層的第  $n$  個神經元

$k$ : 前一層的第  $k$  個神經元

$$\text{Sigmoid} ( Z ) = 1 / ( 1 + e^{-Z} )$$

# 神經網路 Neural Network

由多個輸入，交叉連結多個輸出神經元，一層一層的交錯連結形成網路。



層 L / 神經元 n :

總和:  $Z(L,n) = \text{Sum} ( w(L,n,k) * a(L-1,k) + b(L,n) )$

$$\sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

激活值:  $a(L,n) = \text{Sigmoid} ( Z(L,n) )$

$$a_j^l = \sigma \left( \sum_k w_{jk}^l a_k^{l-1} + b_j^l \right)$$

$w(n,k)$  : 當前層 L 第 n 個神經元的第 k 個輸入(前一層的神經元)的權重。

$a(L-1,k)$  : 當前層 L 第 n 個神經元的第 k 個輸入(前一層的神經元)的激活值。

$b(L,n)$  : 當前層 L 第 n 個神經元的偏向值。

L: 當前層, L-1 前一層。

n: 當前層的第 n 個神經元

k: 前一層的第 k 個神經元

Cost:  $c(L,n) = (y(L,n) - a(L,n))^2$ , cost 值越小, 表示越接近理論值。

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

例如:

層 1 / 神經元 1:

總和值:  $z(1,1) = w(1,1,1)*x_1 + w(1,1,2)*x_2 + \dots w(1,1,i)*x_i + b(1,1)$

激活值:  $a(1,1) = \text{Sigmod}( z(1,1) )$

Cost:  $c(1,1) = ( a(1,1) - y(1,1) )^2$

# 梯度下降 Gradient Decend:

## 二次 Cost 和 Error ( Quadratic Cost)

輸出層 Error 定義：

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L).$$

因為 Cost 會隨著 activation 變化而改變，因此我們定義 Error 方程為 cost 對 a 的變化量(微分)。  $Err(L) = d(cost(L)) / d(a(L,n))$ 。

因為  $cost = (a-y)^2$ ，其中  $a = \text{Sigmoid}(z)$ ，因此 cost 對 a 微分  $\Rightarrow Err(L) = (a-y) * \text{sigmoid}'(z)$ ，其中  $\text{sigmoid}'(z)$  為 a 相對於 z 的變化。

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L). \quad \Rightarrow \quad \delta^L = (a^L - y) \odot \sigma'(z^L). \quad \Rightarrow \quad \delta^L = \nabla_a C \odot \sigma'(z^L).$$

非輸出層 Error 定義:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l),$$

非輸出層的誤差，是下一層的誤差  $err(L+1)$ ，反向乘以  $wei[L+1,k]$ ，得到的值。

$$Err(L,k) = Err(L+1,n) * Weis(L+1,n,k)$$

$$Err(L) = (a-y) * \text{sigmoid}'(z) * Weis(L+1,n,k) = Weis(L+1,n,k) * (a-y) * \text{sigmoid}'(z)$$

因為 Error 是 Cost 對 activation 的變化量，所以 Error 為 cost 對 a 微分。

又因為 a 是由神經元的 weights 和 bias 得到 ( $a = \text{Sigmoid}(wi*xi + b)$ )，所以 Error 也可以是 cost 對 weight 的變化量、cost 對 bias 的變化量。

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l.$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l.$$

計算 w, b 的漸變量，更新前一層的 w 和 b，慢慢逼近最佳值。

w 漸變量  $\rightarrow w0(L,n,k) = d(c(L,n)) / d(w(L,n,k))$  # 將 cost 對 weight 微分

b 漸變量  $\rightarrow b0(L,n) = d(c(L,n)) / d(b(L,n))$  # 將 cost 對 bias 微分

新的 w, b 就等於

$$w(L,n,k) = w(L,n,k) + w0(L,n,k)$$

$$b(L,n) = b(L,n) + b0(L,n)$$

## Cost 公式解說

L : 當前層的編號，

L-1 : 前一層編號

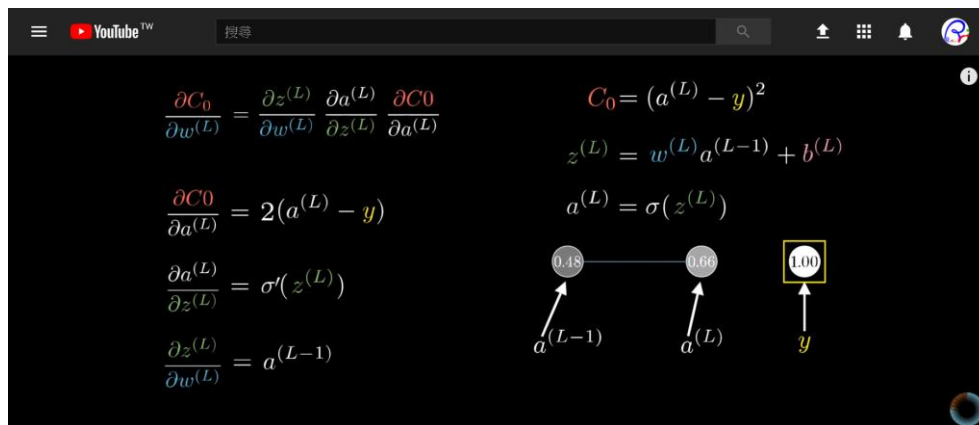
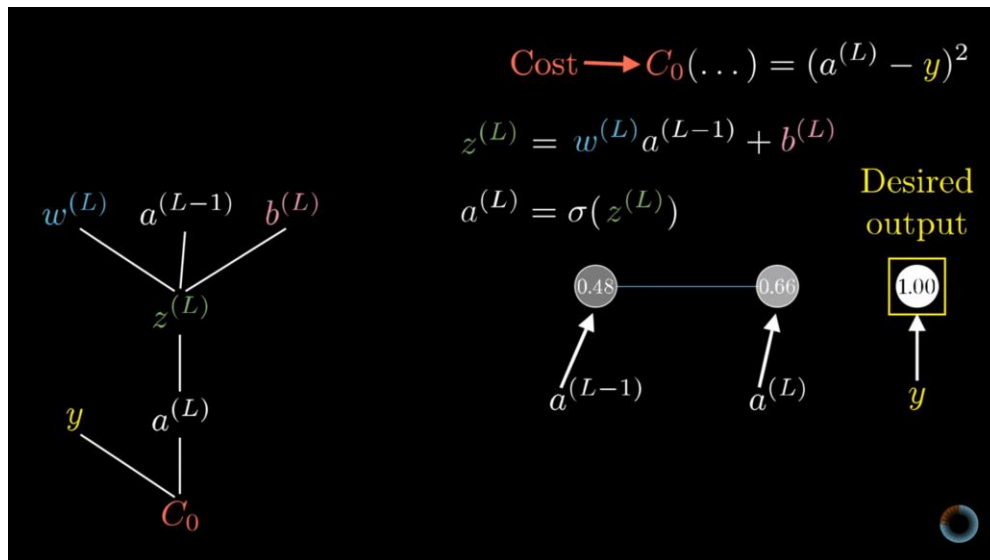
k : 前一層的神經元個數

z : 當前層 L 的神經元的 Sum 值,  $z = w_1*a_1 + \dots + w_k*a_k + b$

a : 當前層 L 的神經元的 Sigmoid 值,  $a = \text{Sigmoid}(z)$

y : 當前層 L 的神經元的理論值

C0 : 當前層 L 所有神經元 Sigmoid 值對理論值的差值平方和  $c = (a-y)^2$



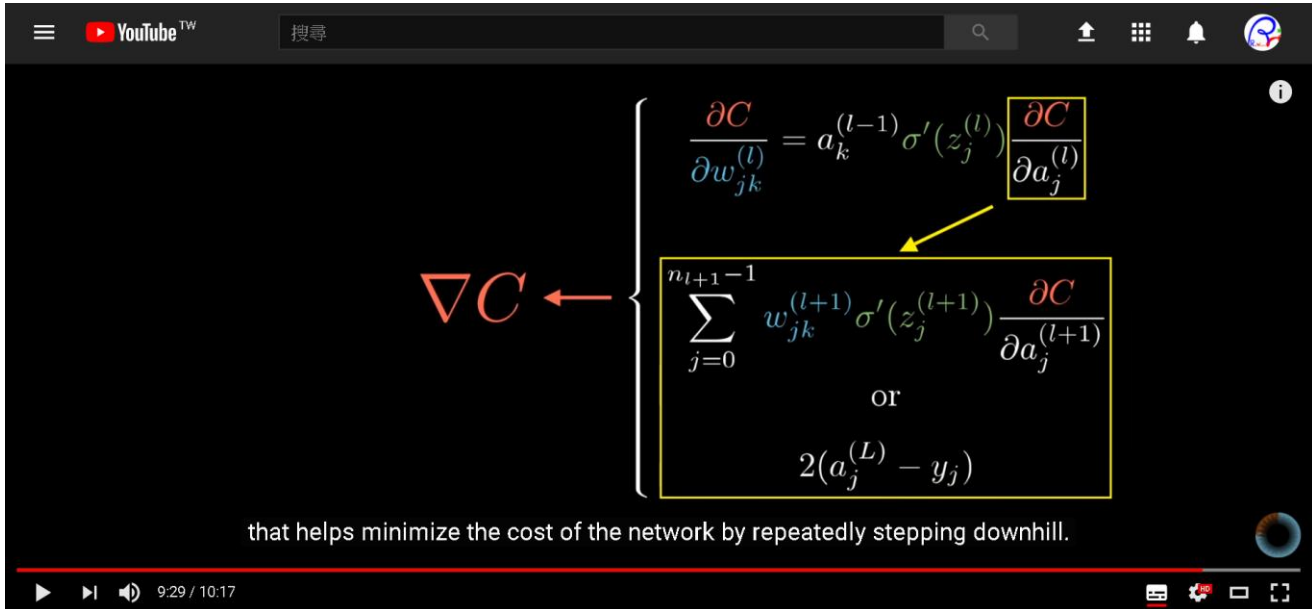
說明:

<https://www.youtube.com/watch?v=llg3gGewQ5U>

數學:

<https://www.youtube.com/watch?v=tIeHLnjs5U8>

- $L$  : 當前層的編號，  
 $L-1$  : 前一層編號  
 $j$  : 當前層  $L$  的神經元編號  
 $k$  : 前一層  $L-1$  的神經元編號  
 $z_j$  : 當前層  $L$  的第  $j$  個神經元的 Sum 值,  $z_j = w_{j0} * a_0 + w_{j1} * a_1 + \dots + w_{jk} * a_k + b_j$   
 $a_j$  : 當前層  $L$  的第  $j$  個神經元的 activation 值,  $a_j = \text{Sigmoid}(z_j)$   
 $C_0$  : 當前層  $L$  所有神經元 Sigmoid 值對對理論值的差值平方和



The video player shows a mathematical derivation for the backpropagation of error gradients through a neural network layer. The main equation is:

$$\nabla C \leftarrow \left\{ \begin{array}{l} \frac{\partial C}{\partial w_{jk}^{(l)}} = a_k^{(l-1)} \sigma'(z_j^{(l)}) \boxed{\frac{\partial C}{\partial a_j^{(l)}}} \\ \sum_{j=0}^{n_{l+1}-1} w_{jk}^{(l+1)} \sigma'(z_j^{(l+1)}) \frac{\partial C}{\partial a_j^{(l+1)}} \\ \text{or} \\ 2(a_j^{(L)} - y_j) \end{array} \right.$$

that helps minimize the cost of the network by repeatedly stepping downhill.



## 交叉熵 Cost 和 Error (CrossEntropyCost)

### Step 3: Gradient Descent

- Formulation of  $\partial L / \partial w$  and  $\partial L / \partial b$

$$L(w, b) = \sum_{n=1}^{10} \left( \hat{y}^n - (b + w \cdot x_{cp}^n) \right)^2$$

$$\frac{\partial L}{\partial w} = ? \sum_{n=1}^{10} 2 \left( \hat{y}^n - (b + w \cdot x_{cp}^n) \right) (-x_{cp}^n)$$

$$\frac{\partial L}{\partial b} = ? \sum_{n=1}^{10} 2 \left( \hat{y}^n - (b + w \cdot x_{cp}^n) \right) (-1)$$

## 反向傳播 **BackPropagation**

二次 Cost (**QuadraticCost**)

交叉熵 Cost (**CrossEntropyCost**)

# 訓練停滯 Overfitting

說明：神經網路反向傳播訓練一定次數後，雖然 cost 不斷降低，但是準確率 accuracy 卻沒有顯著提升，稱為 Overfitting。

解決方法：

1. 增加訓練樣本數。
2. 減小神經網路規模，隱藏層越多，越容易造成 Overfitting。  
注意：某些高端的深度學習需大規模的神經網路，不可減少。
3. 在 Cost 方程內加上規範參數 Regularization，用來削弱權重變動太大的 noise 造成的影響。

## 矯正參數 Regularization：

### L1 / L2 Regularization:

對 Cost 函數增加一項變數值，不改變 Cost 函數原來的架構。

所有 weight 平方總和，乘以  $\lambda$  再除以  $2n$ 。  
 $\frac{\lambda}{2n} \sum_w w^2$

Quadratic Cost：

$$C = \frac{1}{2n} \sum_x \|y - a^L\|^2 + \frac{\lambda}{2n} \sum_w w^2$$

CrossEntropy Cost:

$$C = -\frac{1}{n} \sum_{xj} [y_j \ln a_j^L + (1 - y_j) \ln(1 - a_j^L)] + \frac{\lambda}{2n} \sum_w w^2$$

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} w$$

$$\frac{\partial C}{\partial b} = \frac{\partial C_0}{\partial b}$$

L1 Regularization 減少一個常量

$$C = C_0 + \frac{\lambda}{n} \sum_w |w| \quad w \rightarrow w' = w - \frac{\eta \lambda}{n} \text{sgn}(w) - \eta \frac{\partial C_0}{\partial w}$$

L2 Regularization 減少一個權重的固定比例

$$w \rightarrow w' = w \left(1 - \frac{\eta \lambda}{n}\right) - \eta \frac{\partial C_0}{\partial w}$$

## DropOut Regularization:

對神經網路的結構做改變，除了可解決 **OverFitting** 外，也可減少對神經元的依賴，並提高準確率。

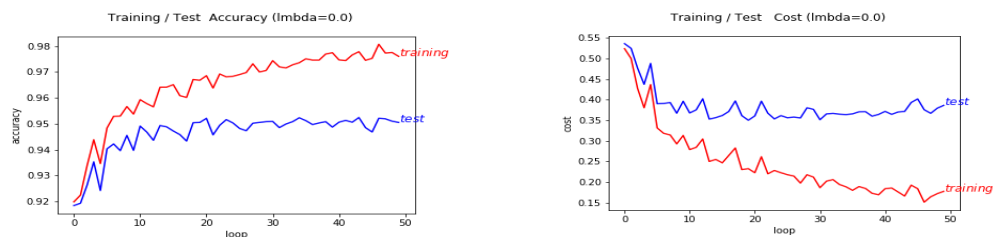
1. 一開始隨機刪掉隱藏層內一半的神經元，然後照之前的方式做運算。
2. 回復刪掉的那一半神經元，刪掉剛剛用到的一半神經元，再做運算。
3. 將兩次所得的輸入做平均。

# 神經網路參數影響結果

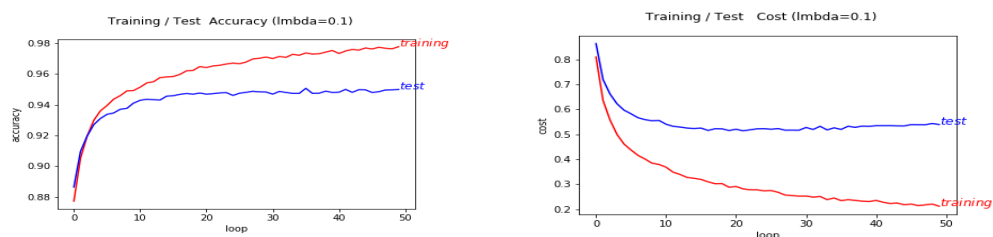
## Cost Function 的影響

## Regularization ( lmbda) 的影響

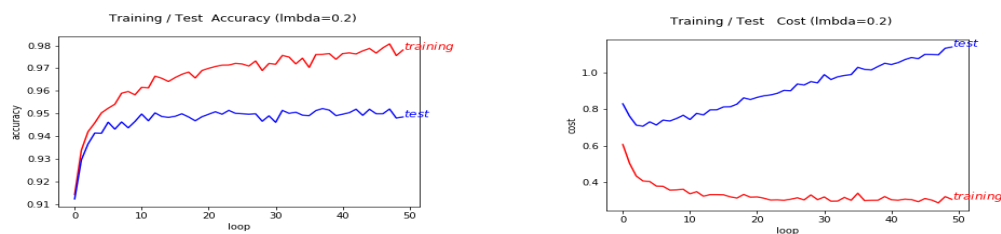
lmbda = **0.0**, learnRate=**3.0**, LayerNeurons=[784, 30, 10]



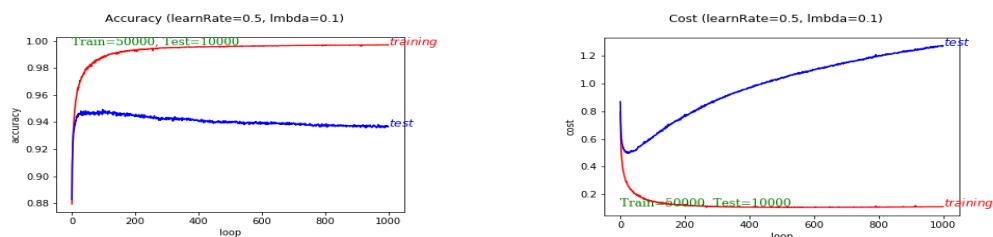
**lmbda = 0.1** 擾動最小，準確度穩定上升，較少上下波動。



lmbda = **0.2**, learnRate=**3.0**, LayerNeurons=[784, 30, 10]

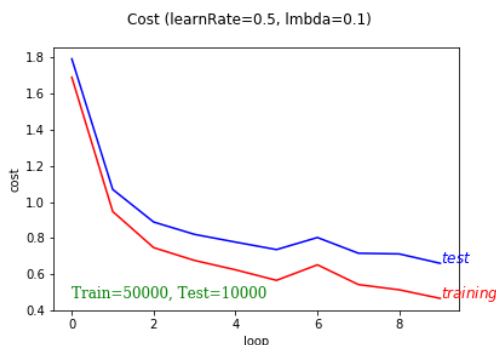
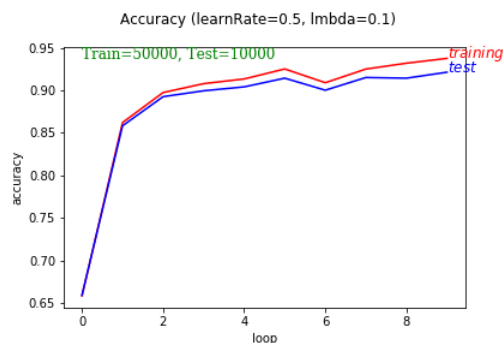


lmbda = **0.1**, **loop =1000**, learnRate=**0.5**, LayerNeurons=[784, 30, 10]

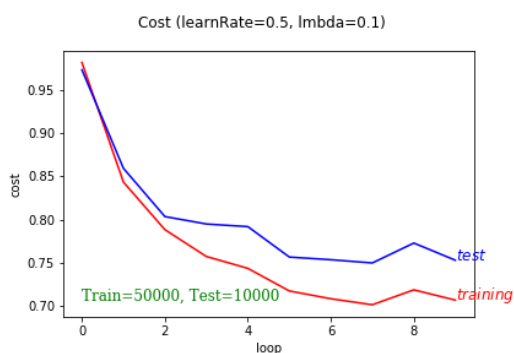
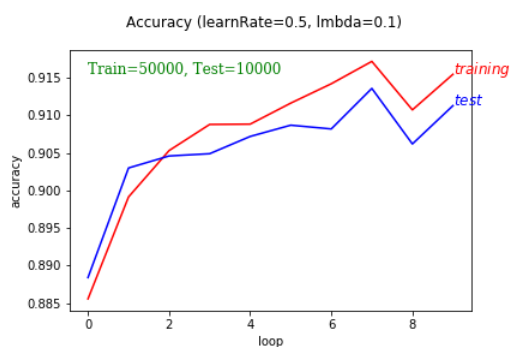


# 神經網路層數的影響

Imbda = **0.1**, learnRate=**0.5**, LayerNeurons= [784, 30, 10, 5, 10, 10]



Imbda = **0.1**, learnRate=**0.5**, LayerNeurons= [784, 10]

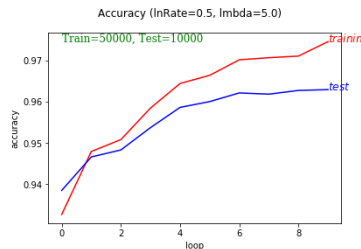
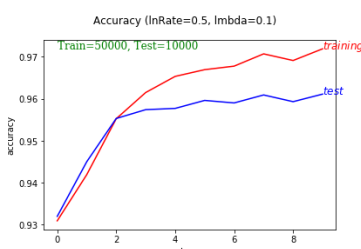
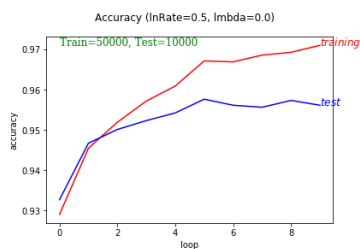
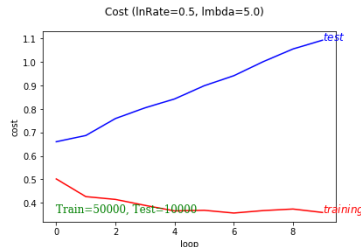
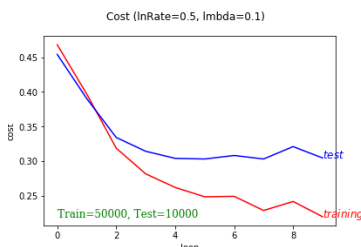
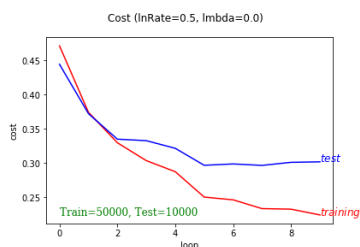


learnRate=**0.5**, LayerNeurons=[784, 30, 10]

Imbda = 0.0

Imbda = 0.1

Imbda = 5.0 (造成 Training Cost 逐步上升)

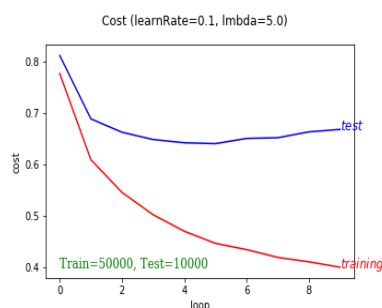


# 權重初始化方式的影響

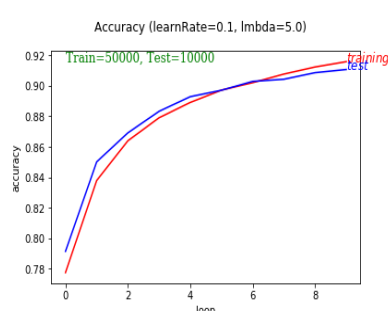
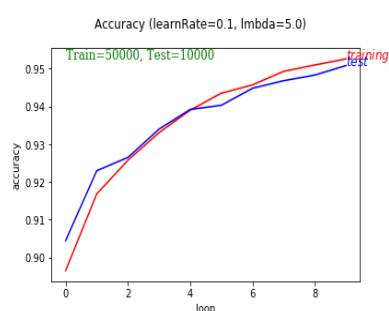
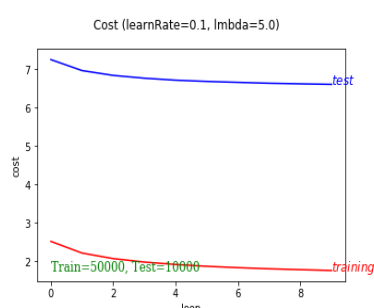
Small Initialization 大幅提高準確率。

loop=10, step=30, **learnRate=0.1**, **Lmbda =5.0**, LayerNeurons=[784,30,10],

**Small Initialization**

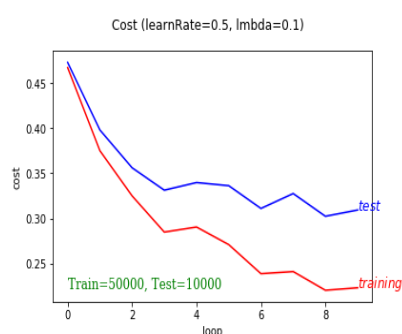


**Large Initialization**

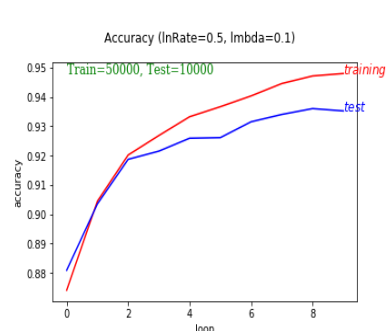
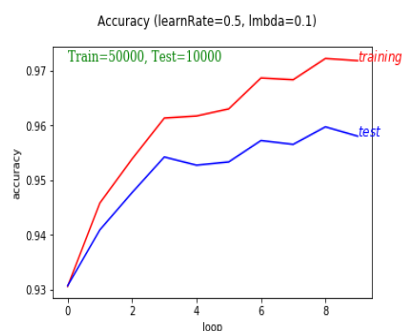
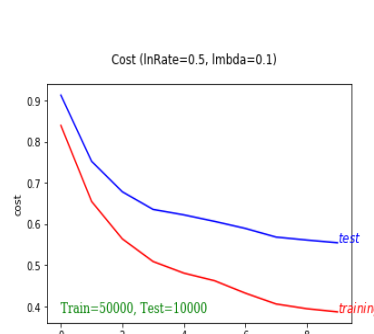


loop=10, step=30, **learnRate=0.5**, **Lmbda =0.1**, LayerNeurons=[784,30,10],

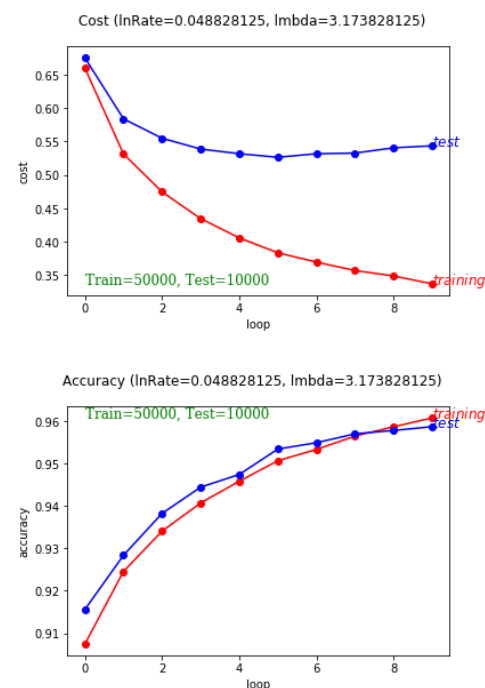
**Small Initialization**



**Large Initialization**



## 自動計算 learnRate, Lmbda





## 提高準確度 Accuracy 的方法：

### 增加參數與函數：

1. 以  $1/\sqrt{\text{inputNum}}$  初始化 weights.
2. 使用 Cross-Entropy Cost Function.
3. 增加 Regularization Imbda 參數.

### 神經網路參數(hyper-parameters)的選取：

LayerNeurons: 層數，神經元數的變化

Weights Initialization: 權重初始化方法

Cost Function: Quadratic 或 Cross-Entropy

Regularization Imbda: 校正參數

Sigmoid or Softmax: 使用的激活值計算方式

Use Dropout: 神經元分批隱藏計算

Test Size 和 Step Num: 訓練集抽樣筆數和步進數

Learning Rate: 學習率

# 自動產生更多訓練資料

將每個數字圖案，做多種微調 旋轉、位移，存出新的圖案。

其他各種圖形，也可依照此方式自動產生更多訓練集合，加強神經網路辨識樣本空間。

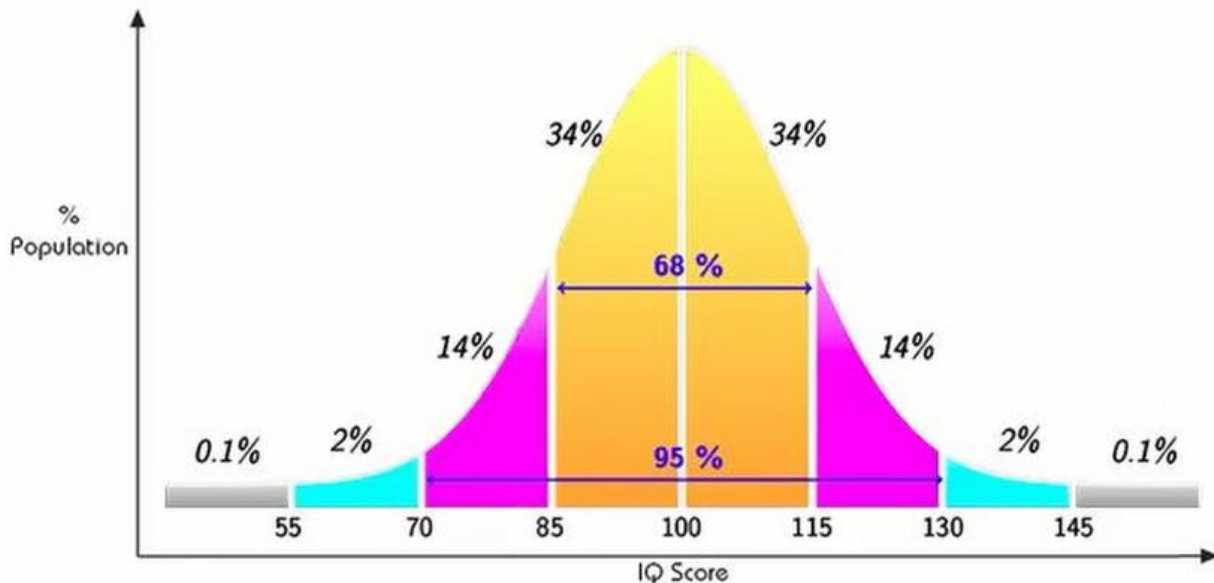
# 權重偏向初始化

均值： $\text{avgX} = (x_1 + x_2 + x_3 + x_4 + \dots + x_n) / n$

方差： $\text{errSqrX} = \text{Sum}((x_1 - \text{avgX})^2 + (x_2 - \text{avgX})^2 + (x_3 - \text{avgX})^2 + \dots + (x_n - \text{avgX})^2) / (n-1)$

標準差： $\text{stdX} = \text{Sqrt}(\text{errSqrX})$

均值相同時，標準差大表示數據分散，標準差小表示數據集中。



如果均值=0, 方差=1, 标准正太分布

智商, 均值= 100, 标准差= 15

均值: mean, average  $\bar{x} = \text{sum}(x_i, i = 1 \dots n) / n$ , 一组数据它的中心趋势的衡量

标准差: standard deviation,  $\sigma = \text{sqrt}(\text{sum}((x_i - \bar{x})^2, (i=1, \dots, n)) / (n-1))$

方差: 标准差<sup>2</sup> = 方差

例： 假設有一個神經元，值  $Z = \text{Sum}(w_i * x_i) + b$ 。有 1000 個輸入  $X$ ，其中 500 個的  $X = 0$ ，另外 500 個  $X = 1$ ，且每個  $w_i = 1, b=1$ 。則

$$Z = (w_1 * x_1 + w_2 * x_2 + \dots + b) = (1 + 0 + 1 + 0 + \dots + b) = 501.$$

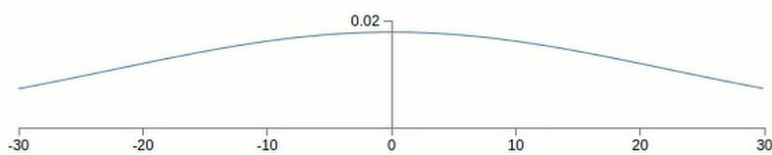
假設  $\text{avgX}(Z) = 0$ ，則  $\text{errSqrX} = 501$ ， $\text{stdX} = \sqrt{501} = 22.4$

以 0 為均值，標準差=22.4 的正態分布圖如下：

輸入  $x$ ：一半是 0，一半是 1，一共 1000 個神經元的輸入層

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

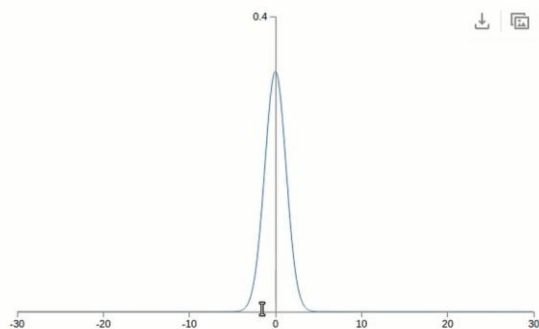
結果：一半消失了，剩下的 500 個 1，加上  $b$ ，分布：標準差： $\sqrt{501} = 22.4$



更好的初始化 Weights 方式：設均值=0，標準差= $1/\sqrt{n}$

上面例子  $\Rightarrow$  標準差 =  $1/\sqrt{1000} \Rightarrow$  方差=(標準差) $^2 = 1/1000$ .

$Z$  的方差 =  $500 * (1/1000) + b = 1/2 + 1 = 3/2$ ， $Z$  的標準差  $\Rightarrow \sqrt{3/2}$



大部分  $z$  在 1 和 -1 之間，神經元沒有飽和，學習過程不會被減慢

# 消失的梯度 Vanish Gradient

## 公式

假設每一層只有一個神經元，則第一層  $b_1$  一個小變化，對最後輸出結果的影響為，  
cost 相對於  $b_1$  的變化率，下面公式為 梯度下降

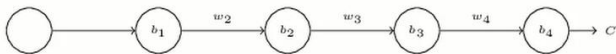
$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$

$$\sigma(z_j)$$

$$z_j = w_j a_{j-1} + b_j$$

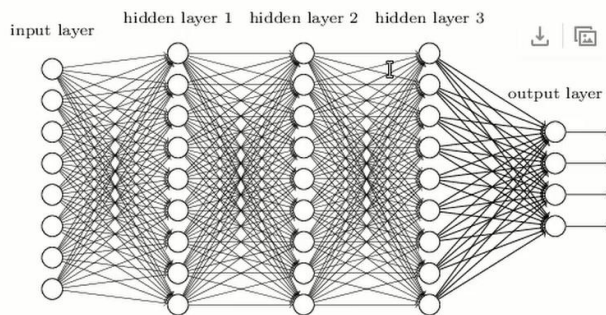
得到的  $\partial C / \partial b_1$  的表达式:

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$



多個神經元的矩陣公式

$$\delta^l = \Sigma'(z^l)(w^{l+1})^T \Sigma'(z^{l+1})(w^{l+2})^T \dots \Sigma'(z^L) \nabla_a C$$



# 解決方法

⌚ Sigmoid unit :

$$f(x) = \frac{1}{1 + \exp(-x)}$$

Tanh unit:

$$f(x) = \tanh(x)$$

Rectified linear unit (ReLU):

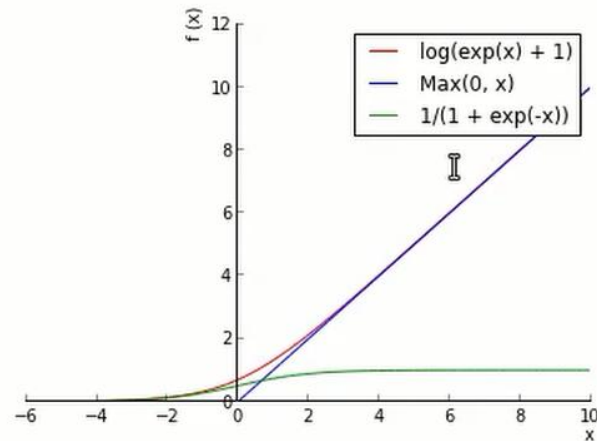
$$f(x) = \sum_{i=1}^{\infty} \sigma(x - i + 0.5) \approx \log(1 + e^x)$$

we refer

- $\sum_{i=1}^{\infty} \sigma(x - i + 0.5)$  as **stepped sigmoid**
- $\log(1 + e^x)$  as **softplus function**

softplus函数可以被max函数模拟  $\max(0, x + N(0,1))$

max函数叫做Rectified Linear Function (ReLU)



Sigmoid 和 ReLU 主要區別：

Sigmoid 函數值在[0,1], ReLU 函數值在[0, inf], 所以 Sigmoid 函數適合用來描述概率，而 ReLU 適合用來描述實數。

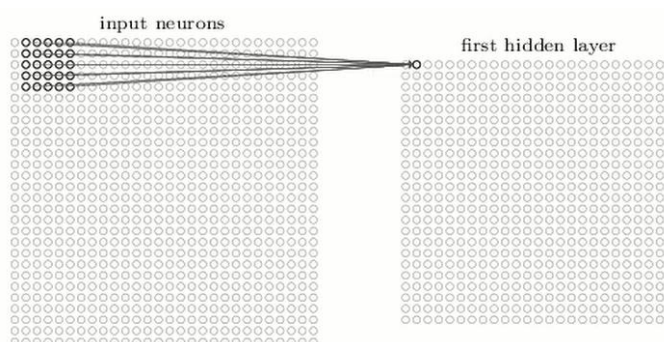
Sigmoid 函數的 Gradient 隨著 x 增大或減小而消失。

ReLU 函數不會，Gradient=0(if x<0), Gradient=1(if x>0)

Rectified Linear Unit 在神經網路的優點是不會產生 Vanishing Gradient 的問題。

# CNN 網路(Convolution Neural Network)

二維輸入：



原始圖像為 28x28，假設 特徵 Filter=5x5，  
Stride=1(每次掃描移動一格)，則可產生輸入  
(28-5+1) x (28-5+1) = 24 x 24 個神經元的層，此 24x24  
的層就叫做 Feature Map。

每次以一個特定的特徵 Filter (5x5)，對第一個隱  
藏層的 24x24 所有神經元做此特定的特徵判別。

下面左邊，以 3 個 Filter 產生 3 個 Feature Map，  
判斷三個特徵。

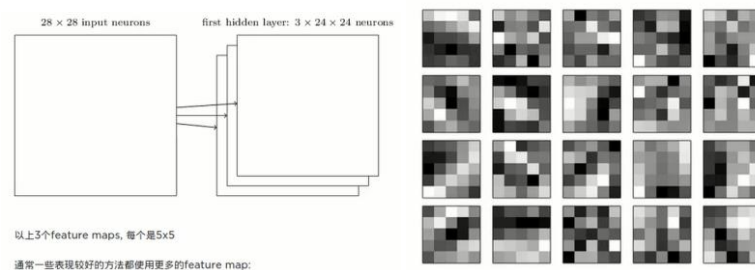
下面右邊，以 20 個 5x5 的特徵 Filter 用來判斷  
20 個不同特徵。

28x25, 5x5 => 24x24

stride: 每次移动多少

共享权重和偏向(shared weights and biases):

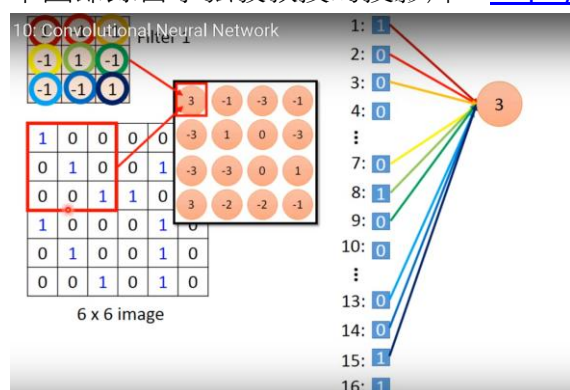
$$\sigma \left( b + \sum_{l=0}^4 \sum_{m=0}^4 w_{l,m} a_{j+l,k+m} \right)$$



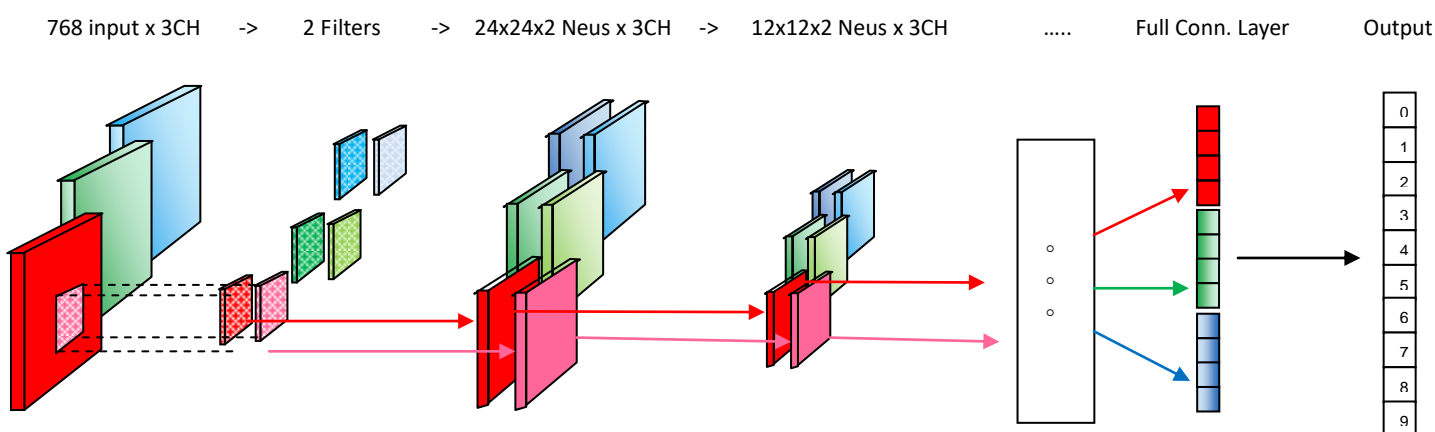
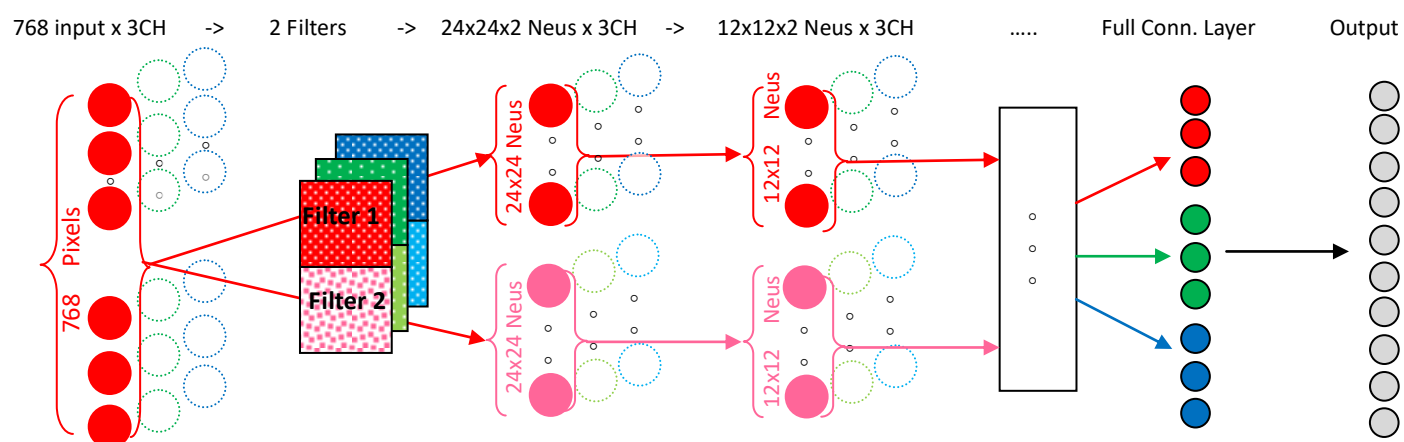
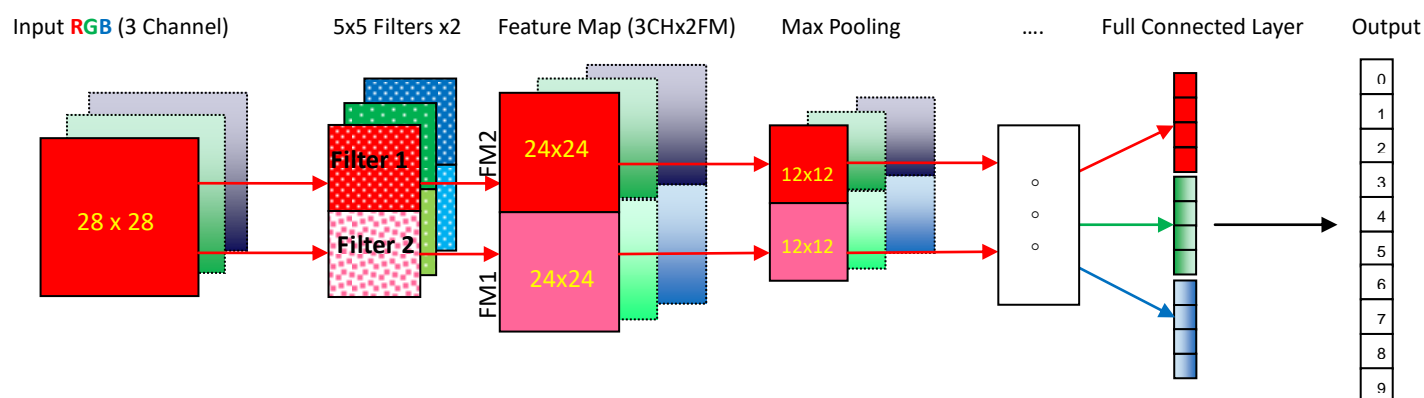
CNN 相當於 DNN 跳著選取輸入。

假設原始資料維度 n1=6，Filter 維度 m=3，Stride=1，則第二層的神經元的維度 n2 = (n1-m+1) = 4。

下圖節錄自李弘毅教授的投影片 <https://www.youtube.com/watch?v=FrKWiRv254g>



# 多 Channel, 多 Filter 卷積神經網路(CNN)





# 準備工作

1. 先安裝 Anaconda ( 包含完整必備的 Python, 開發工具 Spyder, NumPy, ScyPy.....開發包)

<https://goo.gl/4v8Qrk>

2. Python 語言學習

Pytho/Spyder: <https://goo.gl/YHBsB8>

<http://datasciencesource.com/PythonWithSpyderMaterials/>

3. 學習 Machine Learning, Deep Learning

神經網路: <https://goo.gl/hMufrf>

開發者學堂: <https://goo.gl/e6DyzT>

彭亮課程: <https://goo.gl/QUewd8>

線上書籍: <http://neuralnetworksanddeeplearning.com/index.html>

我參考一篇 NN (Neural Network) 教學網頁，修改程式碼，增加詳細註解和製作圖示說明:

下載下列測試檔後，解壓縮內有 \*.py 檔案。

在之前安裝的 Anaconda 開始選單內，選取 Spyder，讀入 \*.py，按下 F5 執行看看是否成功。.

下載 NN 測試檔案:

[https://drive.google.com/open?id=1u\\_rk97coBO71lvznphRmdnCWwg--w2o7](https://drive.google.com/open?id=1u_rk97coBO71lvznphRmdnCWwg--w2o7)

參考教學:

<https://iamtrask.github.io/2015/07/12/basic-python-network/#>

其他:

3Blue1Brown: [https://www.youtube.com/channel/UCYO\\_jab\\_esuFRV4b17AJtAw](https://www.youtube.com/channel/UCYO_jab_esuFRV4b17AJtAw)

神經網路:

[https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQQObOWTQDNU6R1\\_67000Dx\\_ZCJB-3pi](https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQQObOWTQDNU6R1_67000Dx_ZCJB-3pi)

Andrew Ng: [http://ufldl.stanford.edu/wiki/index.php/UFLDL\\_Tutorial](http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial)

ML v.s. DL : <https://www.youtube.com/watch?v=-SgkLEuhfbg>

數字辨識解說: <https://www.youtube.com/watch?v=aircAruvnKk>

NN 原理: <https://www.youtube.com/watch?v=ILsA4nyG7IQ>

反向傳播: <https://www.youtube.com/watch?v=tIeHLnjs5U8>

Convolution NN : [https://www.youtube.com/watch?v=DEcvz\\_7a2Fs](https://www.youtube.com/watch?v=DEcvz_7a2Fs)

# 數學教學

3Blue1Brown:

[https://www.youtube.com/channel/UCYO\\_jab\\_esuFRV4b17AJtAw](https://www.youtube.com/channel/UCYO_jab_esuFRV4b17AJtAw)

線性代數基礎:

[https://www.youtube.com/watch?v=kjBOesZCoqc&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE\\_ab](https://www.youtube.com/watch?v=kjBOesZCoqc&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab)

微積分：

<https://www.youtube.com/watch?v=WUvTyaaNkzM&list=PLZHQObOWTQDMSr9K-rj53DwVRMYO3t5Yr>

神經網路：

[https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1\\_67000Dx\\_ZCJB-3pi](https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi)