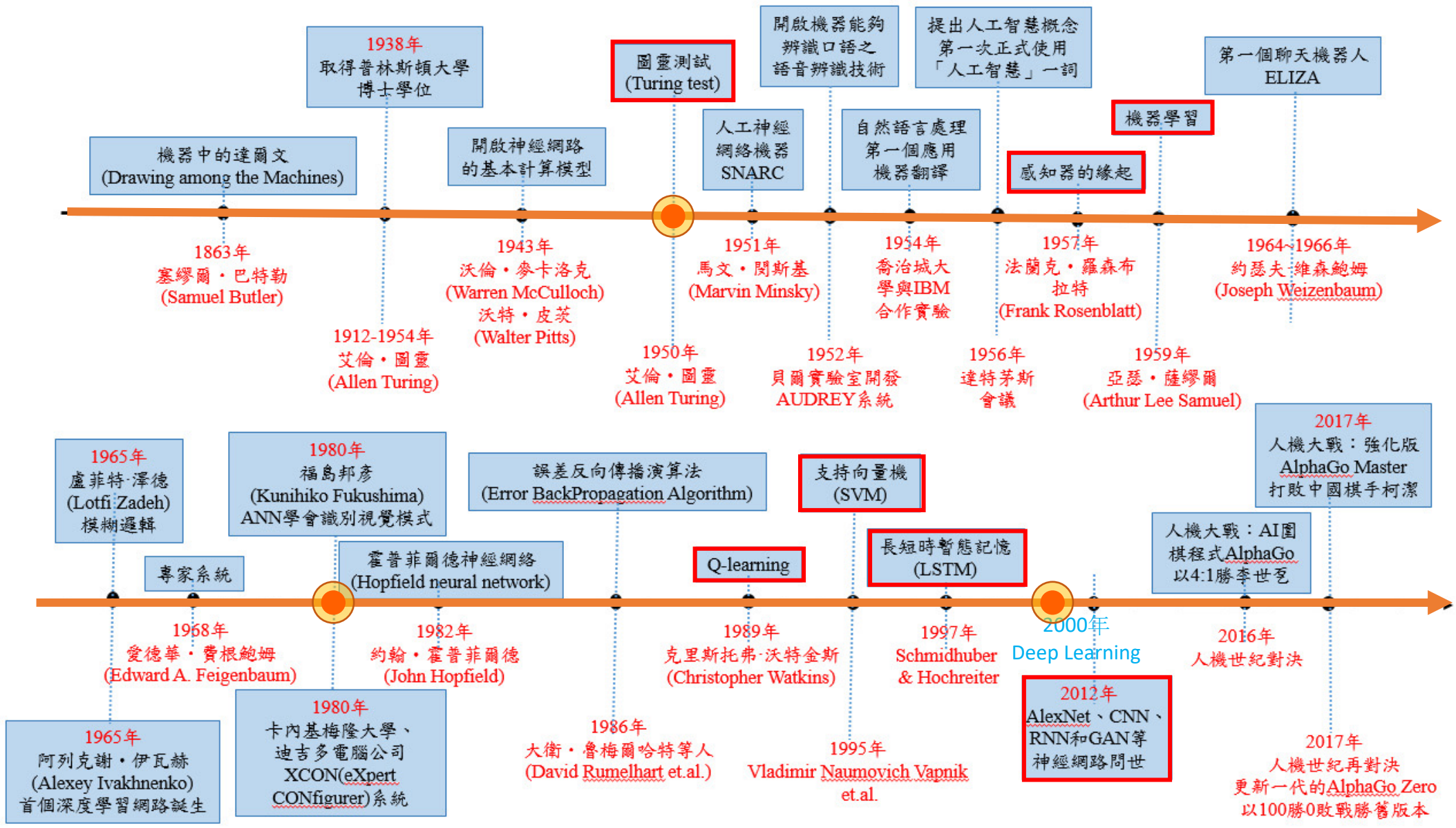
深度學習

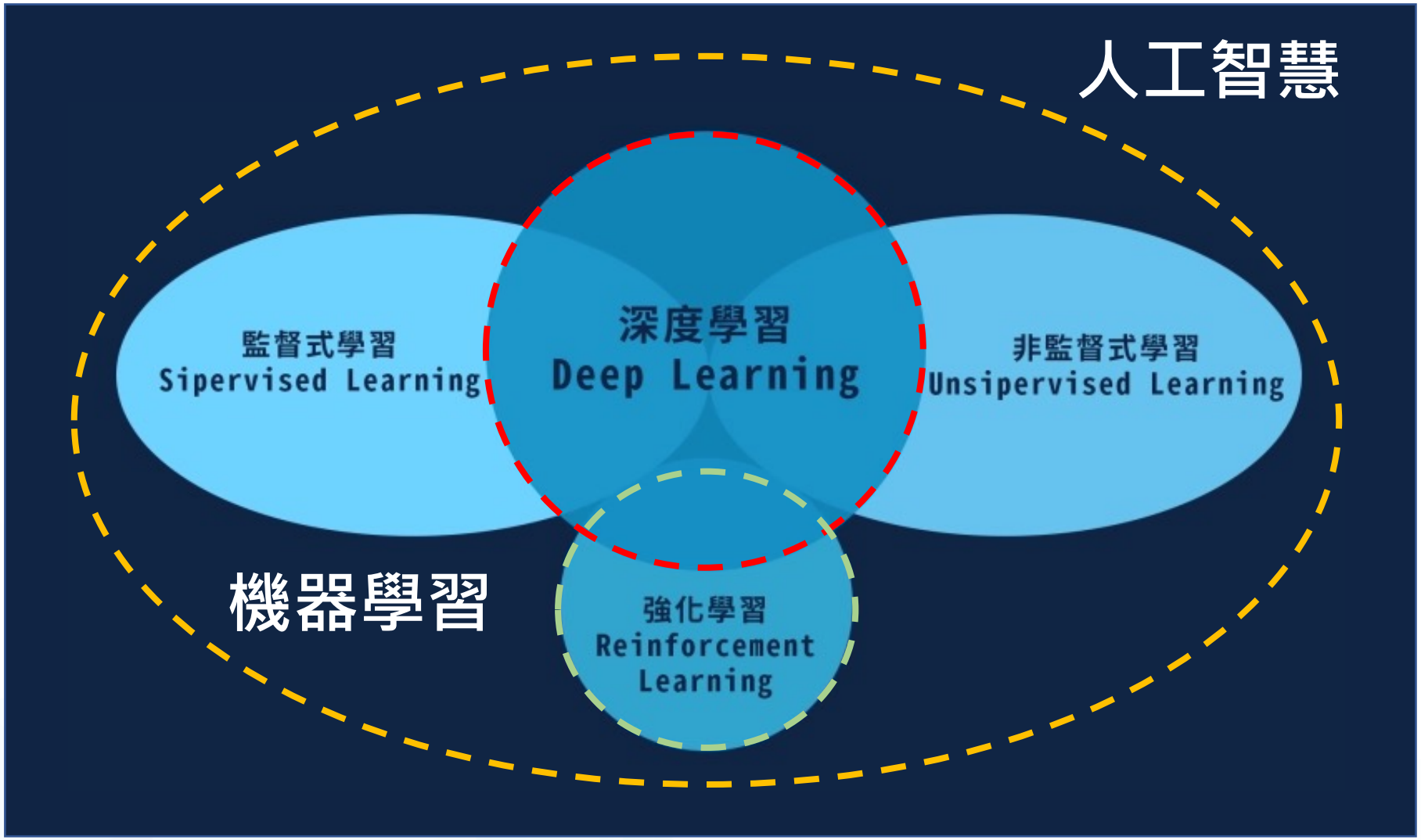
王一書

人工智慧演算法發展史



來源

機器學習與深度學習關係

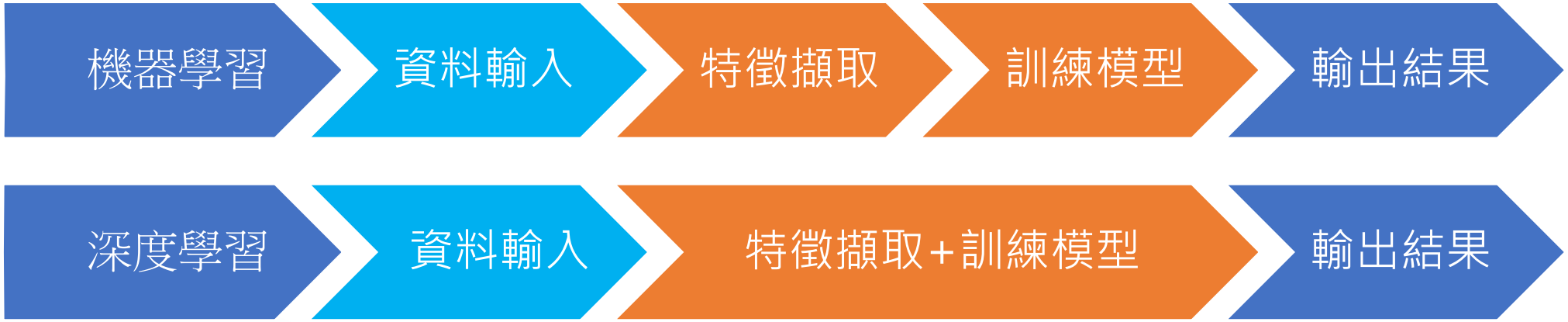


來源

機器學習 vs. 深度學習

深度學習是機器學習的延伸，它不需要人為進行特徵擷取。只要把資料輸入訓練模型，模型會主動進行特徵擷取。

當資料量夠多時，深度學習通常比傳統機器學習有更好的效果。



深度學習主要演算法

CNN 計算機視覺

- CNN原理
- LeNet、ResNet、VGG、YOLO
- 語義分割、定位、物件偵測，實例分割
- 應用：風格轉換、圖像辨識、人臉辨識

NLP 自然語言處理

- 語文前處理
- 語言模型(詞袋、中文分詞)
- RNN原理
- LSTM
- GRU
- 注意力機制
- 情緒分析、文本分類、文字生成

Transformer

- 原理介紹
- LLM語言模型
- 轉移學習
- ChatGPT

GAN 生成對抗網路

- 原理介紹
- ChatGPT、TextToImage
- 應用：圖片生成、文件生成

Reinforcement 強化學習

- 原理介紹
- 馬可夫決策
- 蒙地卡羅算法、Q-learning、DQN
- 策略評估
- 應用：井字遊戲

深度學習課程大綱

第一篇：神經網路 DNN
第二篇：計算機視覺 CNN
第三篇：自然語言處理 NLP
第四篇：Transformer
第五篇：生成對抗網路 GAN
第六篇：強化學習 Reinforcement
第七篇：進階演算法-動態規劃

第一篇：神經網路 ANN

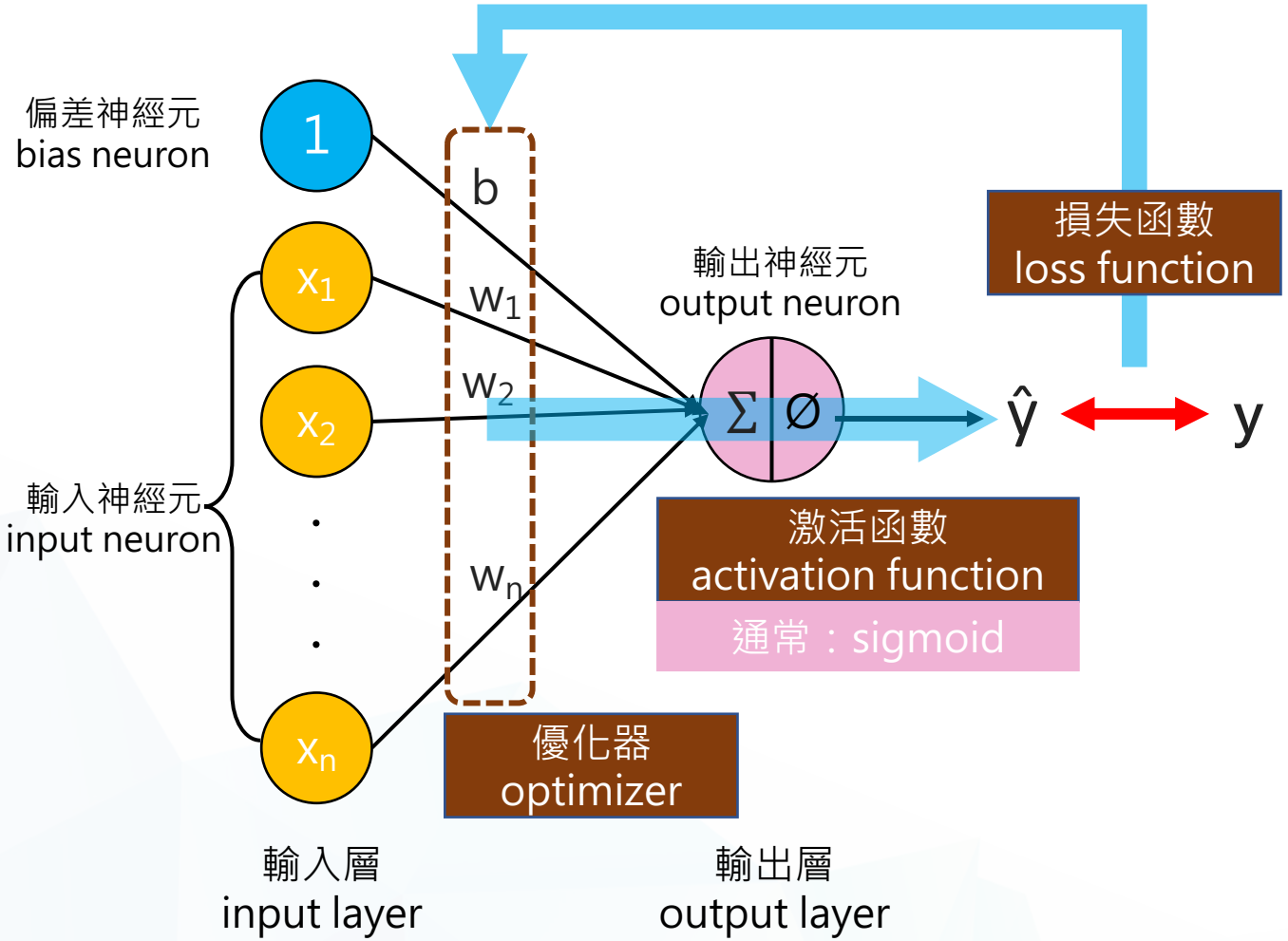
王一書

本章教學大綱

- 神經網路導論
- 激勵函數
- 損失函數
- 優化器

神經網路導論

感知器(Perceptron) ：單一神經元(Neurons)



$$z = b + w_1x_1 + w_2x_2 + \ldots + w_nx_n = w^Tx$$

$$\hat{y} = Ø(z) = Ø(w^Tx)$$

b：偏差值(bias)

$w_1, w_2 \ldots, w_n$：權重(weights)

Ø (z)：激活函數(activation function)

$\hat{y}$：預測結果

優化器：調整權重的算法

```
import numpy as np

X = np.array([[1,3,3],
              [1,4,31],
              [1,1,1],
              [1,0,2]])
y = np.array([[1],
              [1],
              [-1],
              [-1]])
w = (np.random.random([3,1])-0.5)*2

print(np.sign(np.dot(X, w)))

print(X.T.dot(y - out))/int(X.shape[0]))
```

二維data

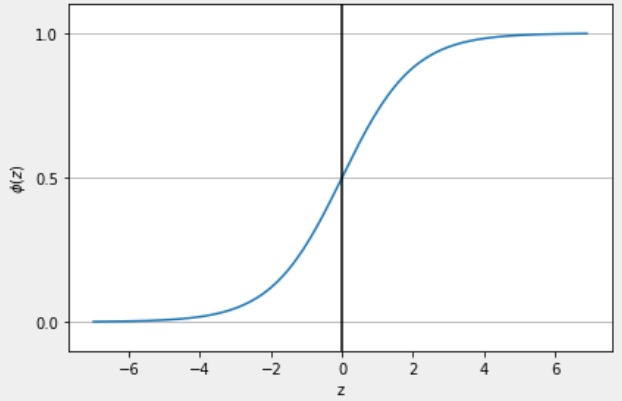
一維結果

二維權重

$$X = \begin{bmatrix} 1 & 3 & 3 \\ 1 & 4 & 31 \\ 1 & 1 & 1 \\ 1 & 0 & 3 \end{bmatrix}, W = \begin{bmatrix} 0.61 \\ -0.96 \\ 0.02 \end{bmatrix}$$

perceptron.ipynb

```python
import numpy as np

def sigmoid(z):
    return 1.0 / (1.0 + np.exp(-z))

z = np.arange(-7, 7, 0.1)

phi_z = sigmoid(z)

plt.plot(z, phi_z)
plt.axvline(0.0, color='k')
plt.xlabel('z')
plt.ylabel('$\phi (z)$')

plt.show()
```



sigmoid.ipynb

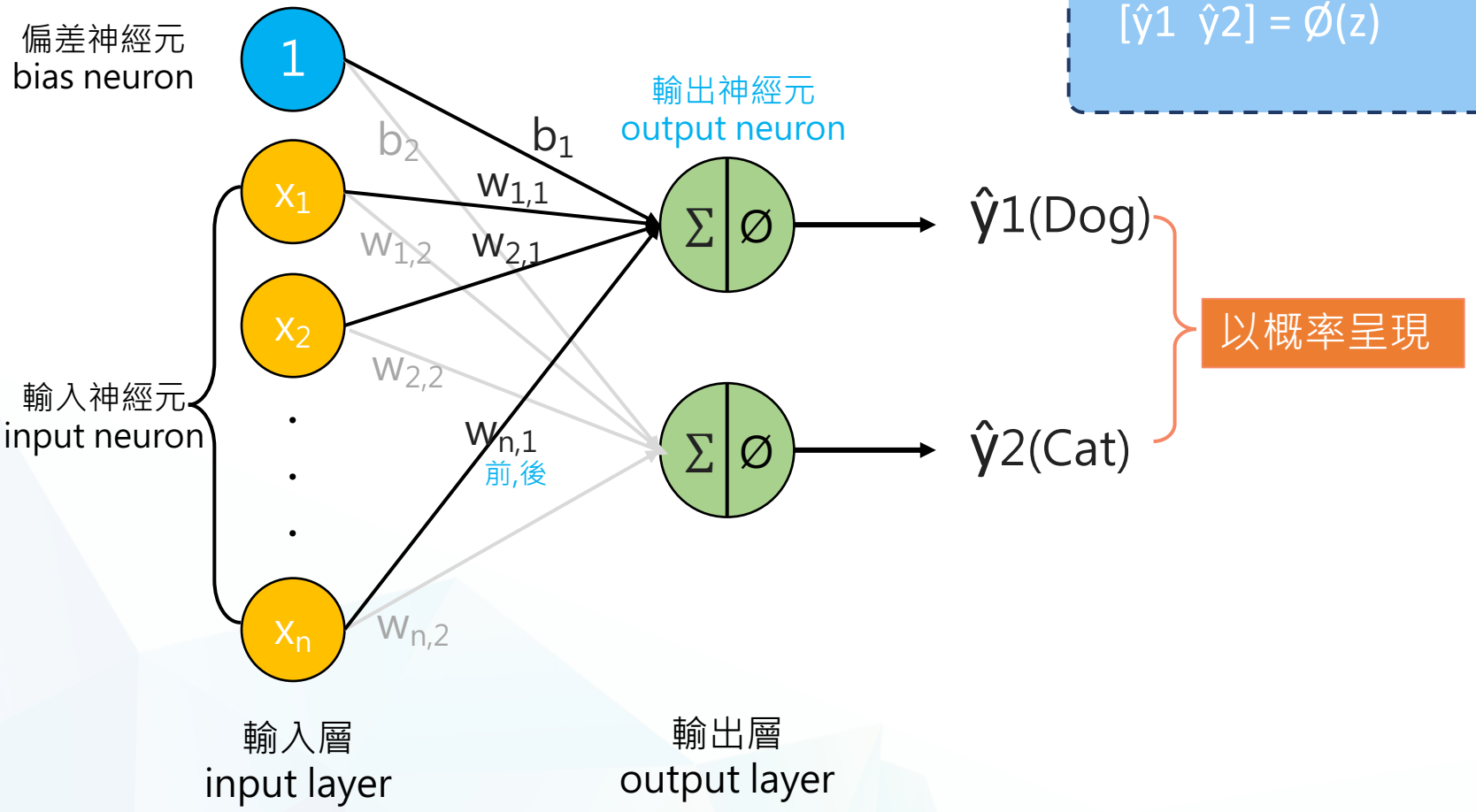
多個感知器

多個感知器可進行多分類(多輸出)。

$$z = [x_1\ x_2\ \dots\ x_n] * \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} + [b_1\ \ b_2]$$

y=1 y=2

$$[\hat{y}1\ \ \hat{y}2] = \emptyset(z)$$



```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense

model = Sequential([
    #輸入層：將 28*28攤平成一維度
    Flatten(input_shape=x_train.shape[1:]),
    #輸出層：10類別，10個神經元
    Dense(units=10, activation='softmax')
])

model.summary()
# Compile
model. compile(loss='sparse_categorical_crossentropy', optimizer='sgd',
metrics= ['accuracy’])
# Train
train = model.fit(x_train, y_train,epochs=20, validation_data=(x_valid,
y_valid))
```

密集層序列

輸出維度

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 10)                7850

=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
```

輸出神經元數量

可訓練參數

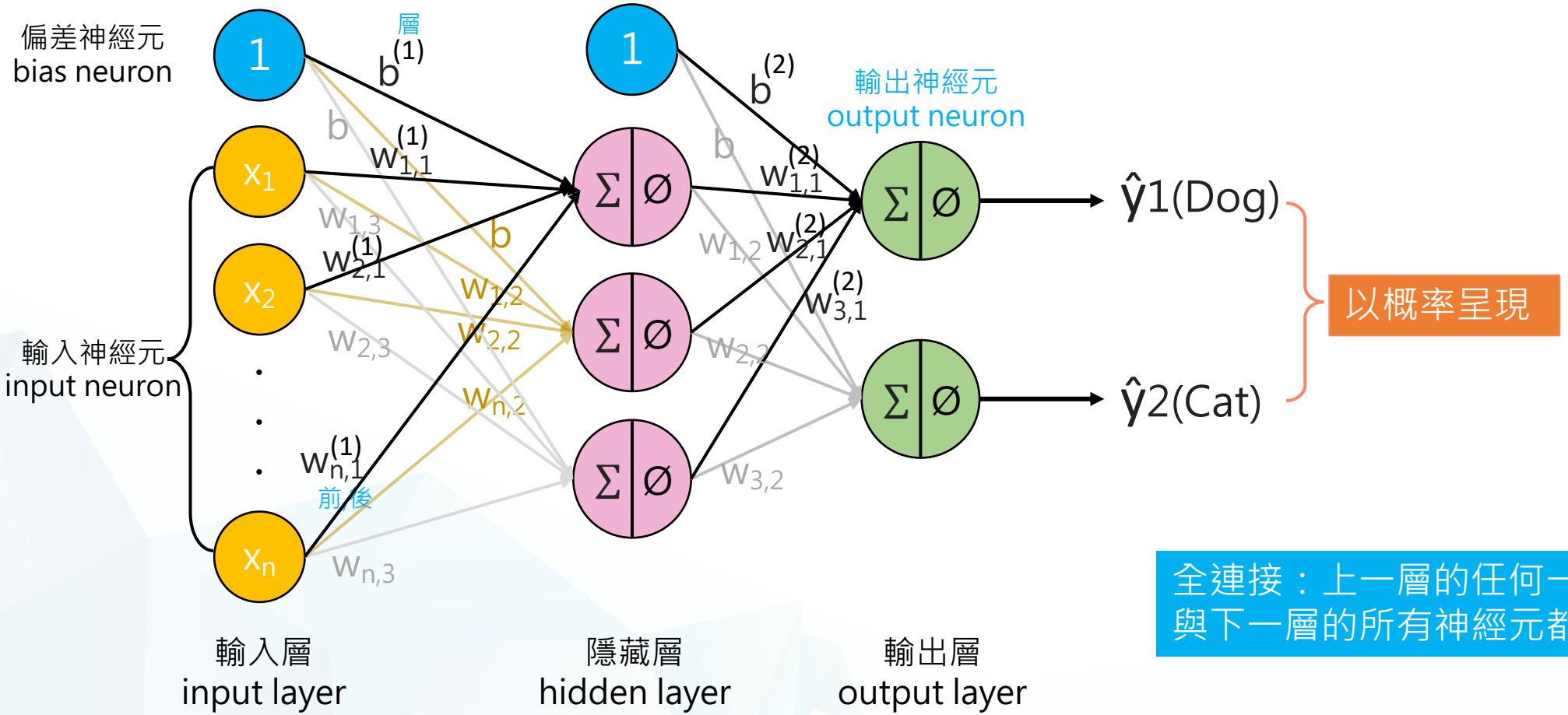
(784 + 1)*10=7850

bias

MP.ipynb

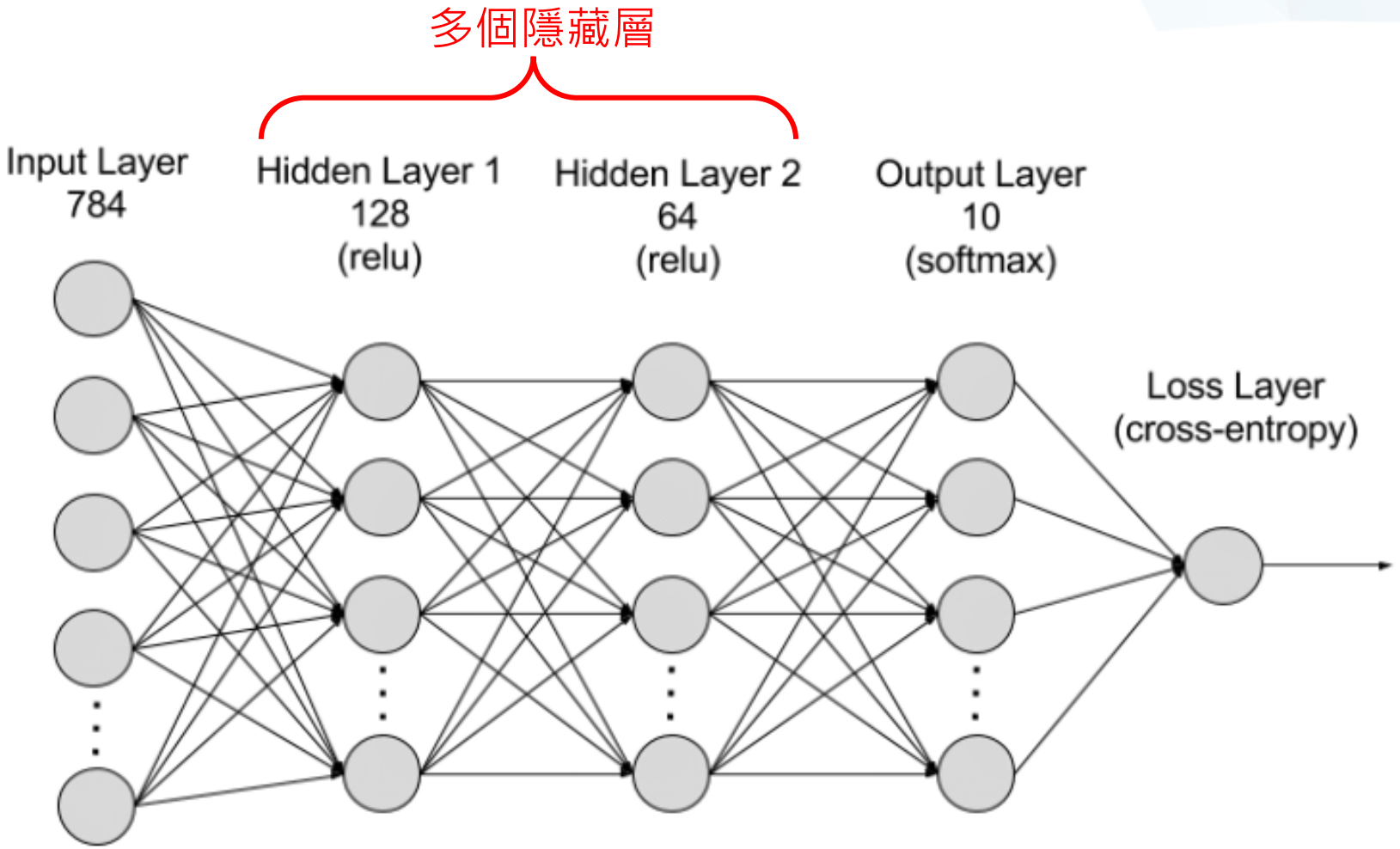
多層感知器(MultiLayerPerceptron, MLP)

多層感知器(Multi-Layer Perceptron，MLP)也叫人工神經網路(Artificial Neural Network，ANN）或類神經網路，可進行分類和迴歸預測。 MLP的層與層之間是全連結的，最底層是輸入層，中間是隱藏層，最後是輸出層。



全連接：上一層的任何一個神經元與下一層的所有神經元都有連接

深層神經網路(Deep Neural Network，DNN)



```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense

model = Sequential([
    #第一層：將 28*28攤平成一維度
    Flatten(input_shape=x_train.shape[1:]),
    #隱藏層
    Dense(units=300, activation='relu'),
    Dense(units=200, activation='relu'),
    Dense(units=100, activation='relu'),
    #輸出層：10類別，10個神經元
    Dense(units=10, activation='softmax')
])

model.compile(loss='sparse_categorical_crossentropy', optimizer='sgd', metrics=
['accuracy'])
train = model.fit(x_train, y_train,epochs=20, validation_data=(x_valid, y_valid))
```

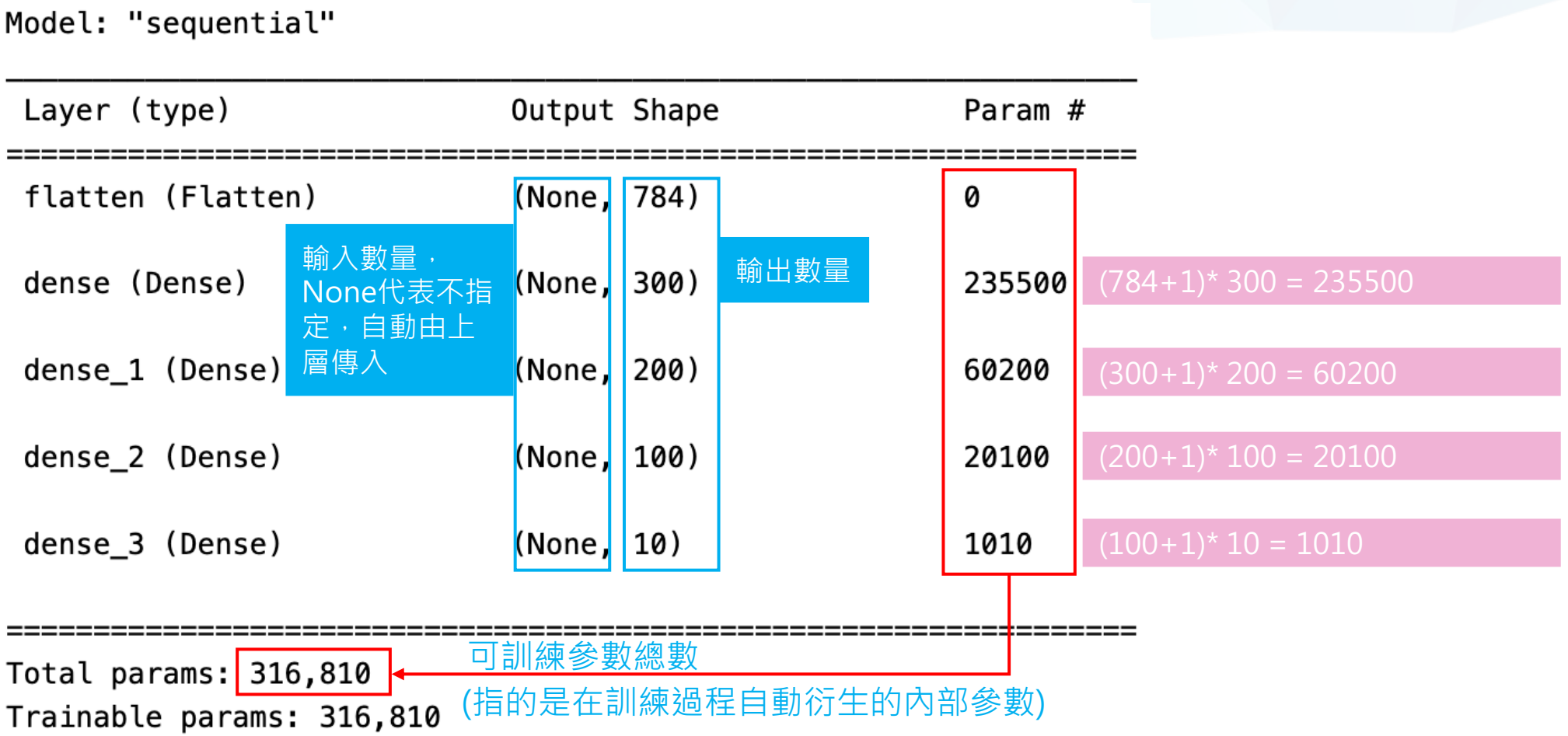
Dense 參數：

- units：輸出神經元數。
- activation：後面要接的activation function。
- use_bias：是否使用偏差項(bias)。
- kernel_initializer、bias_initializer： w、b初始值。
- kernel_regularizer、bias_regularizer： w、b、activation function是否使用正則化(Regularizer function)。
- kernel_constraint、bias_constraint： w、b限制函數。

fit( ) 參數：

- epochs：代表你要執行多少次，資料量多的圖像分析時建議次數大一些，可避免遺漏。
- batch_size ：代表一次要執行的訓練的數量，資料變化性大的圖像分析適合較大數值。

MLP_clf.ipynb


Model: "sequential"
Layer (type)
Output Shape
Param #
flatten (Flatten)
(None, 784)
0
dense (Dense)
(None, 300)
235500
dense_1 (Dense)
(None, 200)
60200
dense_2 (Dense)
(None, 100)
20100
dense_3 (Dense)
(None, 10)
1010
輸入數量，None代表不指定，自動由上層傳入
輸出數量
(784+1)* 300 = 235500
(300+1)* 200 = 60200
(200+1)* 100 = 20100
(100+1)* 10 = 1010
Total params: 316,810
Trainable params: 316,810
Non-trainable params: 0
可訓練參數總數
(指的是在訓練過程自動衍生的內部參數)

激勵函數(Activation Function)

在神經網路中，先以線性組合計算輸入，再以非線性函數進行轉換得到輸出值。此非線性函數即為激勵函數(Activation Function)。

常用的激勵函數有：Sigmoid、tanh、ReLU等。

須注意：

- 激勵函數需選擇可微分之函數，因為在反向傳遞運算時，需要進行微分計算。
- 在深度學習中，當隱藏層之層數過多時，激勵函數不可隨意選擇，因為會造成梯度消失以及梯度爆炸等問題。

超參數：

演算法超參數 (Algorithm Hyperparameters)

在模型訓練時使用的參數，例：學習率、批次數量、優化器等等。

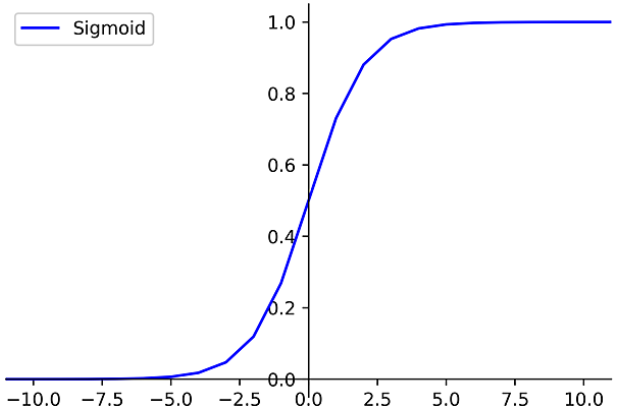
模型超參數 (Model Hyperparameters)

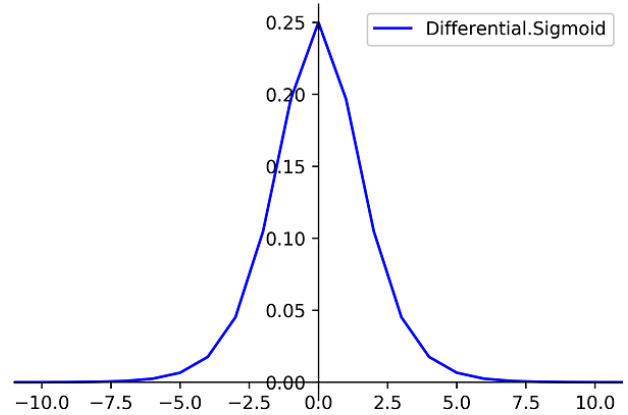
整個神經網路架構用的參數，例：隱藏層數、每層神經元個數、每層使用的激勵函數

Sigmoid函數

Sigmoid函數是深度學習領域開始時使用頻率最高的activation function，其輸出範圍限制在 [0,1] 之間，適合用於二分法。

$$\sigma(x) = \frac{1}{1+e^{-x}}$$





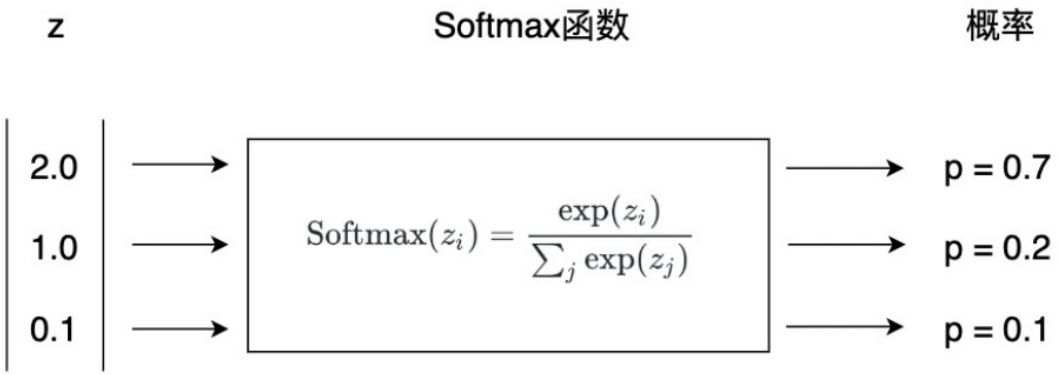
缺點:

- 容易出現梯度消失 (gradient vanishing)
- Sigmoid函數中，當後面神經元之輸入皆為正數時，對權重值求梯度時，梯度數值恆為正，因此在誤差反向傳遞的過程中，權重都正方向更新或往負方向更新，導致收斂曲線不平滑，形成一種綑綁現象，也影響模型的收斂速度。
- 指數運算較為耗時。

Softmax函數

Softmax函數，也稱指數函數，是二分類Sigmoid函數的延伸，目的是將多分類的結果以概率形式展現出來。每一元素值變成 0～1之間的小數，並且總元素的和為1。

$$\text{Softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$



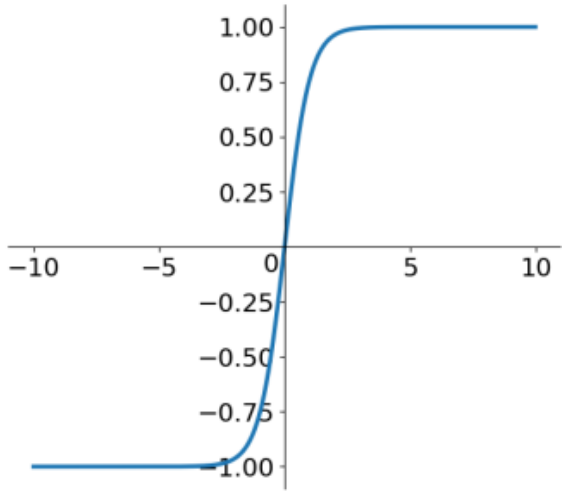
缺點:

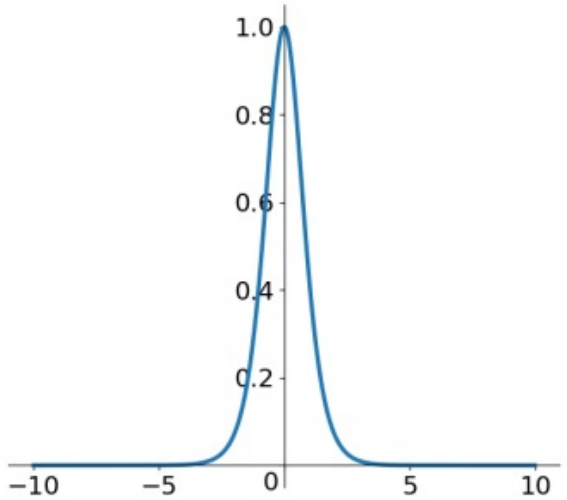
- 指數函數會讓值的間距變大，最大值經過softmax變換後就會逼近1，而其他值則變很小，所以當softmax的輸出值變成 [0,0,1]時，對softmax作反向傳播時，其梯度也基本上接近0，無法繼續學習。

tanh(Hyperbolic Tangent)函數

tanh 通常是優於 Sigmoid的，因為 tanh的輸出在 -1~1之間，均值為0，更方便下一層網路的學習。與Sigmoid函數相比，其收斂速度要比Sigmoid快，減少迭代次數。

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$





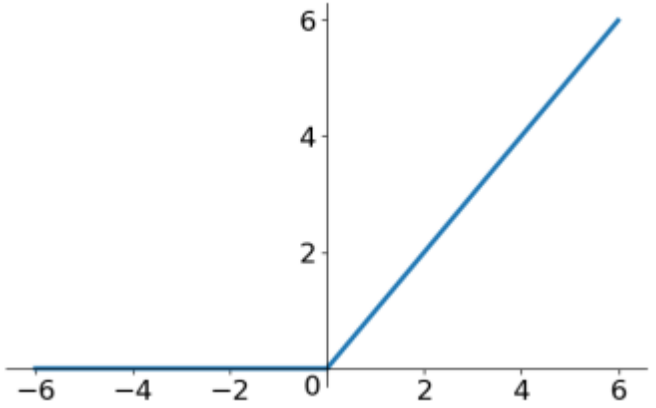
缺點:

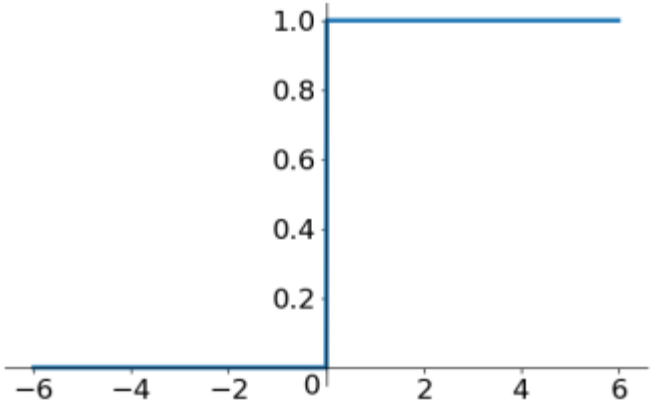
- 容易出現梯度消失
- 跟Sigmoid一樣，指數函數計算量較大，運算較為耗時。
- 如果預測二分類，仍建議在輸出層使用 Sigmoid，因為sigmoid可以算出屬於某一類的概率。

ReLU函數

ReLU的值若為正數，則輸出該值大小，若值為負數，則輸出 0，ReLU函數並不是全區間皆可微分。ReLU是近年來最頻繁被使用的激勵函數，可解決梯度爆炸問題、計算數度相當快、收斂速度快等。(解決sigmoid & tanh的缺點)

$$\mathrm{ReLU} = \max(0, x)$$





缺點:

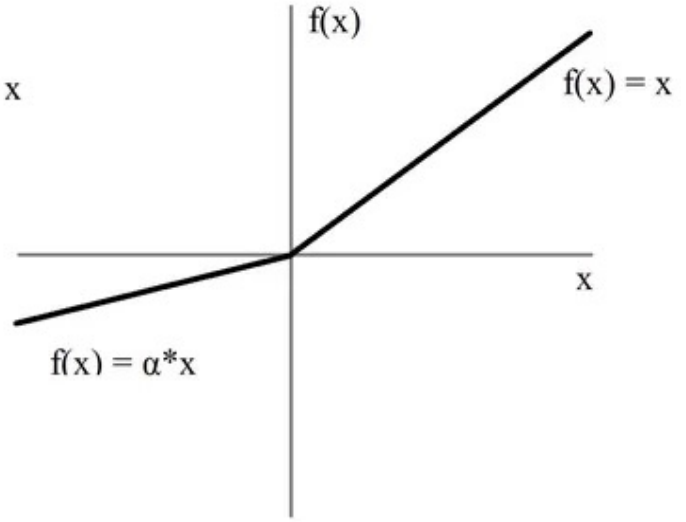
- 當某個神經元輸出為0後，就難以再度輸出，導致dead ReLU發生。
- 學習率設置過大，在剛開始進行誤差反向傳遞時，容易修正權重值過大，導致權重梯度為0，神經元即再也無法被激活。

Leaky ReLU函數

LeakyReLU可以解決神經元「死亡」問題，因為它將負值轉作非零的斜率。 也可解決sigmoid & tanh的缺點)

$$y_i = \begin{cases} x_i & \text{if } x_i \geq 0 \\ \frac{x_i}{a_i} & \text{if } x_i < 0, \end{cases}$$

α可任意指定，一般為 0.01



另外還有延伸：

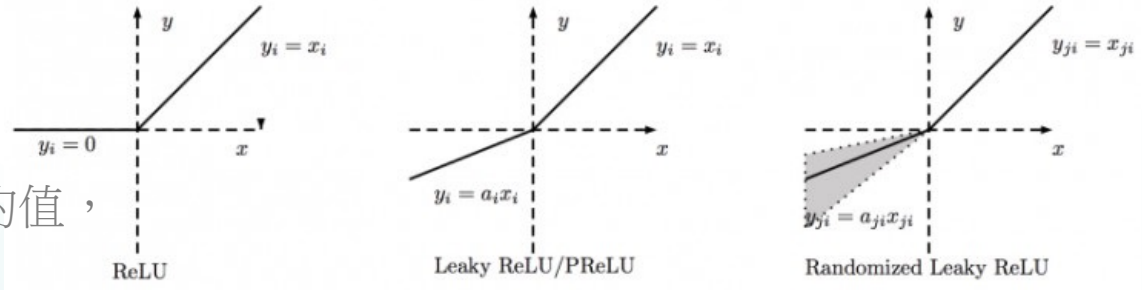
- 參數化修正線性單元（PReLU）

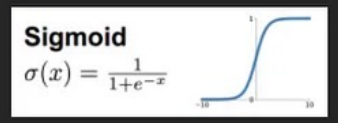
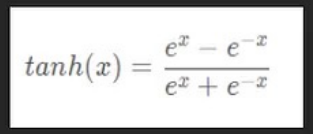
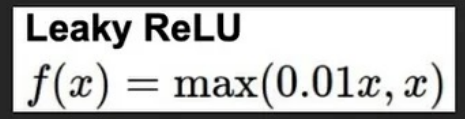
  在PReLU中，負值部分的斜率是根據資料來定的，而非預先定義的。

- 隨機糾正線性單元（RReLU）

  在RReLU中，負值的斜率在訓練中是隨機的，在之後的測試中就變成了固定的了。

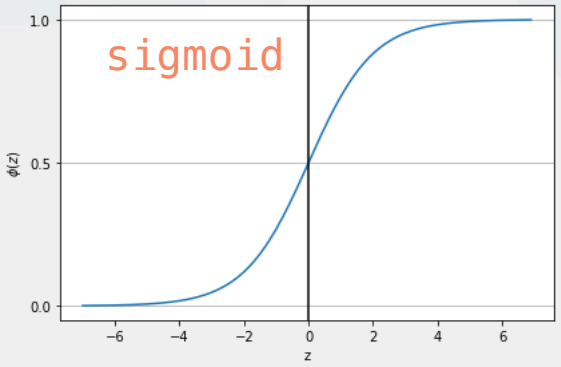
Leaky ReLU中的ai是固定的
PReLU中的ai是根據數據變化的；
RReLU中的aji是一個在一個給定的範圍內隨機抽取的值，
這個值在測試環節就會固定下來。

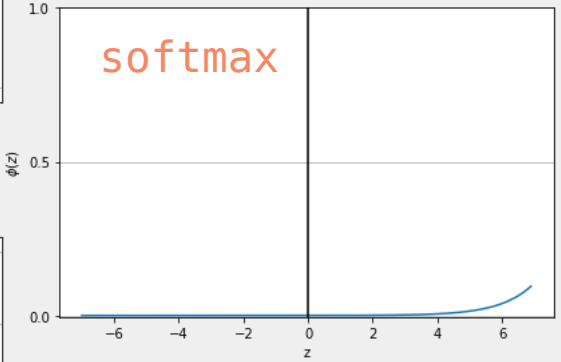


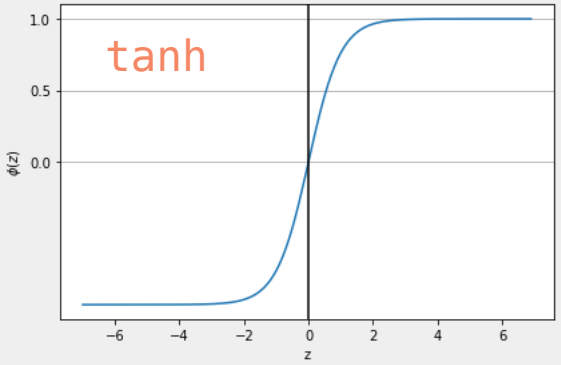
	Sigmoid	tanh	Relu	Leaky Relu/ELU
				
可不可微分	可微	可微	部分可微	可微
計算速度	指數運算很慢	指數運算很慢	很快	很快
函數值域	0~1	-1~1	0~＋無窮大	- 無窮大～＋無窮大
梯度消失問題	容易發生	容易發生	不容易發生	不容易發生
分佈是否為 zero center	X	O	X	O
Dead activate problem	X	X	O	X

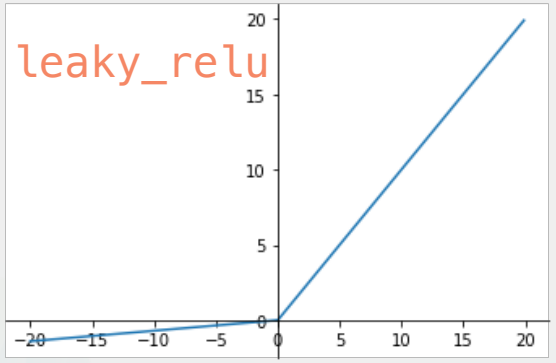
來源

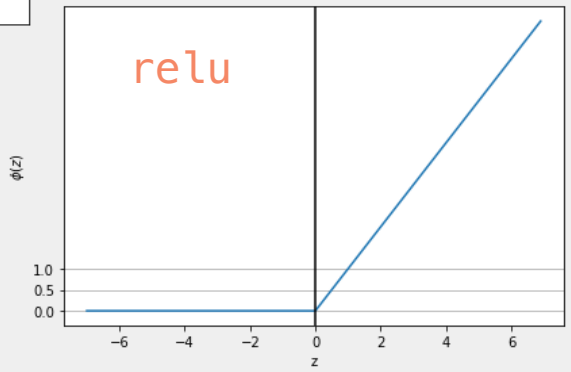
```python
import numpy as np
z = np.arange(-7, 7, 0.1)

def sigmoid(z):
    return 1.0 / (1.0 + np.exp(-z))

def softmax(x):
    return np.exp(x)/sum(np.exp(x))

def tanh(x):
    s1 = np.exp(x) - np.exp(-x)
    s2 = np.exp(x) + np.exp(-x)
    s = s1 / s2
    return s

def relu(x):
    s = np.where(x < 0, 0, x)
    return s

def leaky_relu(x, a):
    if x < 0: return a*x
    else: return x
```











activationFunction.ipynb

損失函數(Loss functions)

衡量模型估算的預測值與實際值之間的差距。損失函數 (loss function) 也稱成本函數(cost function)、目標函數(Objective Function)。

損失函數通常與學習準則與優化問題有關，即通過最小化損失函數求解、評估模型效能。

損失函數（Loss Function）通常是針對單一訓練樣本而言
代價函數（Cost Function）通常是針對整個訓練集
目標函數（Objective Function）通常是一個較通用的術語

常見的損失函數有：

- 迴歸：MSE、MAE
- 分類：Crossentropy
  - 二分類：binary_crossentropy
  - 多分類：categorical_crossentropy

迴歸：MSE、MAE

MSE：均方差損失(Mean Squared Error Loss)是機器學習、深度學習迴歸任務中最常用的一種損失函數，也稱為 L2 Loss。 其基本形式如下：

$$J_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

MAE：平均絕對誤差(Mean Absolute Error Loss，MAE)是另一類常用的損失函數，也稱為L1 Loss。 其基本形式如下：

$$J_{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

- MSE 比 MAE 能夠更快收斂：當使用梯度下降演算法時，MSE的梯度會隨誤差大小而變化，而MAE的梯度則一直保持為1，不利於模型的訓練。
- MAE 針對異常值較穩定：從損失函數來看，MSE對誤差平方化，使得異常值的誤差過大。

```python
# MSE

def squared_error(y, yhat):
return (y - yhat)**2

print(squared_error(1, 1))
print(squared_error(1, 0.9997))
print(squared_error(0, 0.1192))
```

MSE.ipynb

分類：交叉熵 Cross Entropy Loss

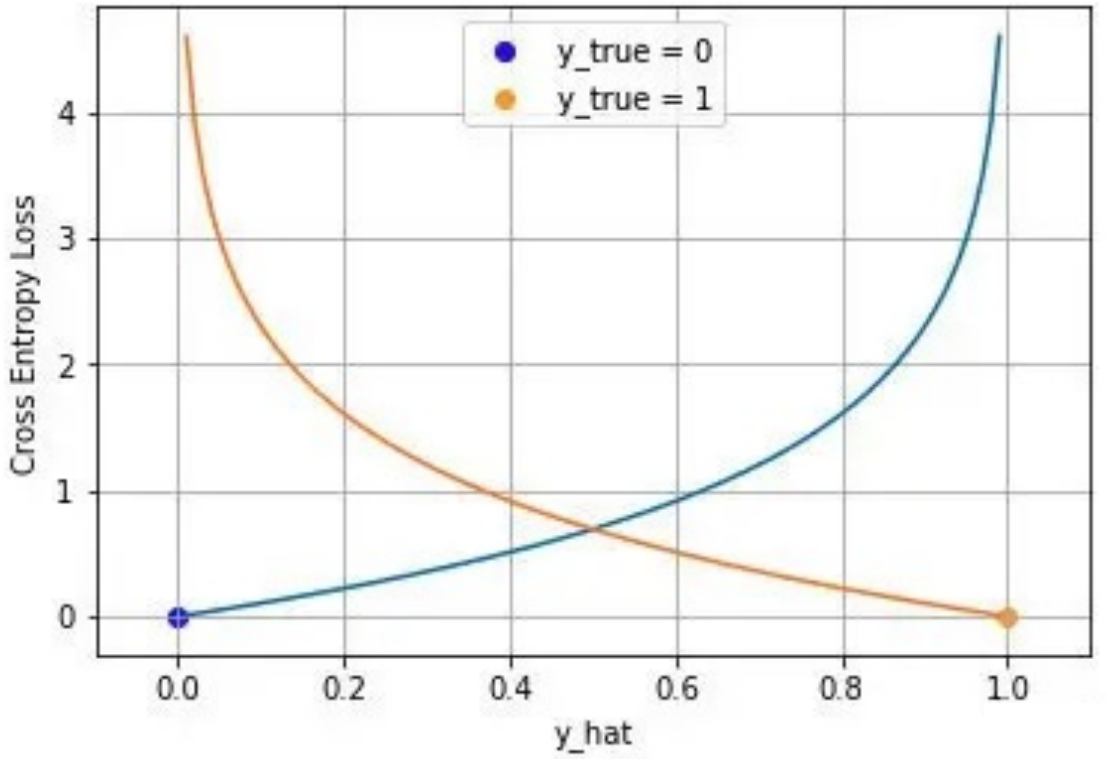
在二分類中通常使用非線性函數將模型的輸出壓縮到某一區間內，例如(0，1)，可依此用來代表輸入值 X_i 應輸出正類與負類的概率：

$$p(y_i = 1|x_i) = \hat{y}_i$$ 正類概率

$$p(y_i = 0|x_i) = 1 - \hat{y}_i$$ 負類概率

合併

$$p(y_i|x_i) = (\hat{y}_i)^{y_i}(1 - \hat{y}_i)^{1-y_i}$$



藍線是目標值為 0 時依不同輸出的損失，黃線是目標值為 1 時的損失。 可以看到愈接近目標值損失越小，隨著誤差變差，損失呈指數成長

```python
from numpy import log

def cross_entropy(y, yhat):
    return -1*(y*log(yhat) + (1-y)*log(1-yhat))

print(cross_entropy(1, 0.9997))
print(cross_entropy(0, 0.1192))
```

cross_entropy_loss.ipynb

多分類時，也被稱為 Softmax Loss 或者 Categorical Cross Entropy Loss。

與二分類不同的是真實值 y 是一個One-hot 向量，同時模型輸出的壓縮由原來的Sigmoid函數換成 Softmax函數。 Softmax 函數將每個維度的輸出範圍都限定在(0，1)之間，同時所有維度的輸出總和為1，用來表示一個機率分佈。

模擬計算：

Softmax 轉換

輸出層	ŷ0	ŷ1	ŷ2	ŷ3	ŷ4	ŷ5	ŷ6	ŷ7	ŷ8	ŷ9
預測值	0.09	0.07	0.08	0.08	0.09	0.10	0.10	0.21	0.10	0.09

One-hot encoding 轉換

實際值	0	0	0	0	0	0	0	1	0	0

C=10

$$J_{CE} = -\sum_{i=1}^{N} y_i^{c_i} \log\left(y_i^{\hat{c}_i}\right)$$

One-hot向量，除了目標類別為1之外其他類別上的輸出都為 0

結果：數字 7

y0=0, ŷ0=0.09, 0*log(0.09)=0
y2=0, ŷ2=0.08, 0*log(0.08)=0
y5=0, ŷ5=0.10, 0*log(0.10)=0
y7=1, ŷ1=0.21, 1*log(0.21)=-0.156

CE = $-\sum y_i \log(\hat{y}i) = 0.156$

CE 值愈低，代表整體誤差愈小

```
# Model
model = Sequential([
    Flatten(input_shape=x_train.shape[1:]),
    Dense (units=300, activation='relu'),
    Dense (units=200, activation='relu'),
    Dense (units=100, activation='relu'),
    Dense (units=10, activation='softmax')
])

# Compile
model. compile(loss='sparse_categorical_crossentropy',
               optimizer='sgd',
               metrics= ['accuracy'])
```

也可以：

```
loss = tf.keras.losses.SparseCategoricalCrossentropy( )
```

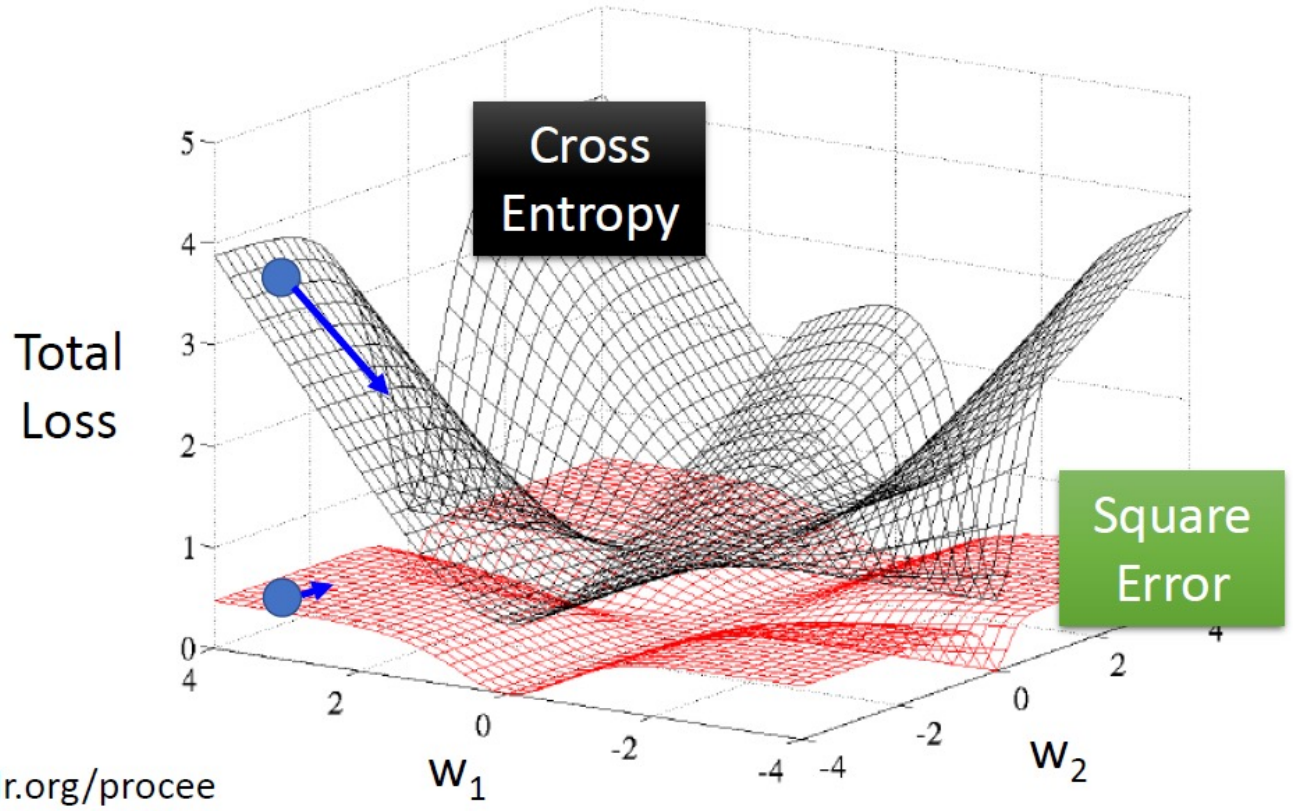
sparse_categorical_crossentropy：會將資料先進行 one-hot 編碼

categorical_crossentropy：接收的資料必須已完成 one-hot 編碼

可參閱先前的程式碼

使用交叉熵可在反向傳播時更快速的下降 Loss

Cross Entropy v.s. Square Error



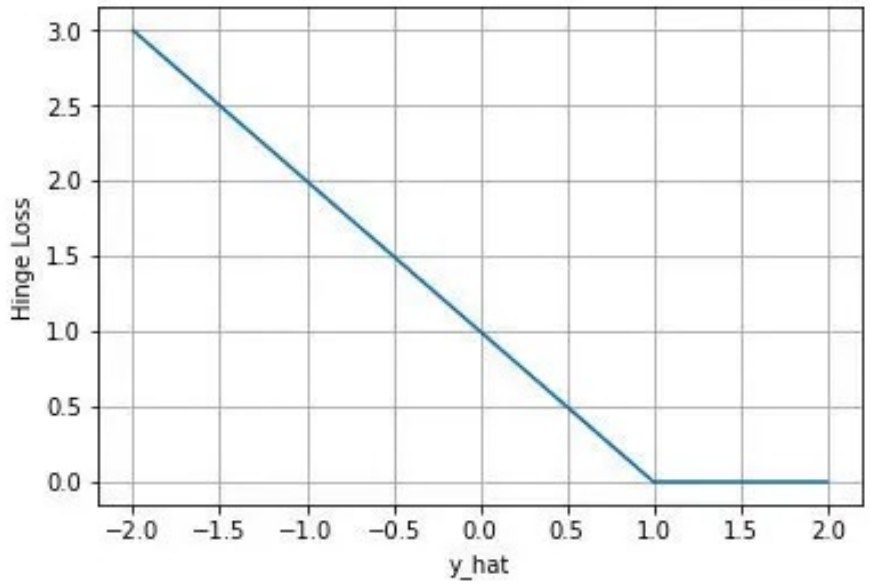
http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf

分類：合頁損失 Hinge Loss

合頁損失（Hinge Loss）是另一種二分類損失函數，適用於 maximum-margin 的分類。支援向量機Support Vector Machine (SVM)模型的損失函數本質上就是Hinge Loss + L2正規化。
合頁損失的公式如下：

$$L(y) = max(0, 1 - \hat{y}y)$$

如果 ŷy < 1，則損失為：1 - ŷy
如果 ŷy > 1，則損失為：0



當 y = 1時，模型輸出負值會有較大的懲罰，當模型輸出為正值且在 (0, 1) 區間時還會有一個較小的懲罰。 即合頁損失不僅懲罰預測錯的，並且對於預測對了但是置信度不高的也會給一個懲罰，只有置信度高的才會有零損失。

```
from sklearn.svm import SVC

model = SVC()

model.fit(X_train_std, y_train)

model.score(X_test_std, y_test)
```

SVC 分類的算法採用Hinge Loss

SVM_乳癌診斷預測.ipynb

綜合練習：

- 手寫阿拉伯數字辨識.ipynb
- MLP_clf02.ipynb

優化器 Optimizer

優化器是指在深度學習反覆計算過程中，指引權重往正確的方向更新，使得更新後的權重能讓 Loss值不斷逼近全域最小。

常用的優化算法就是梯度下降。

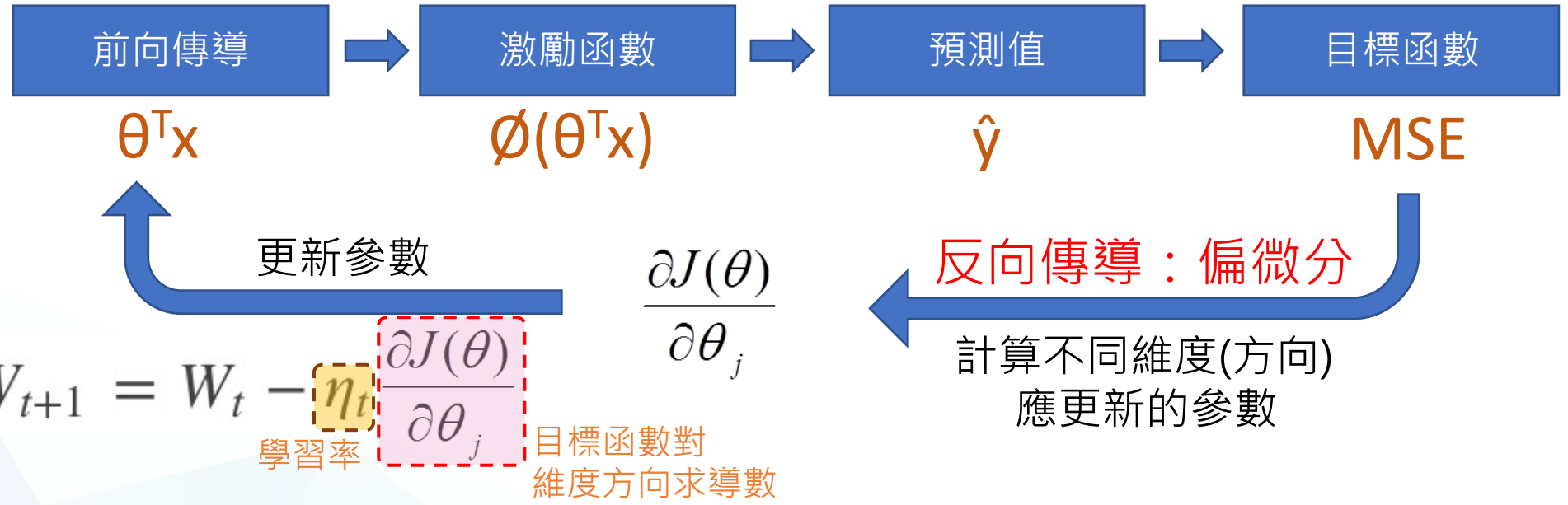
算法原理是使用各參數的梯度值來最小化或最大化損失函數E(X)。

需要關心的主要有二項：

1. 優化方向：決定“前進的方向是否正確”，在優化器中為梯度或動量。
2. 步長：決定“每一步跨多遠”，在優化器中為學習率。

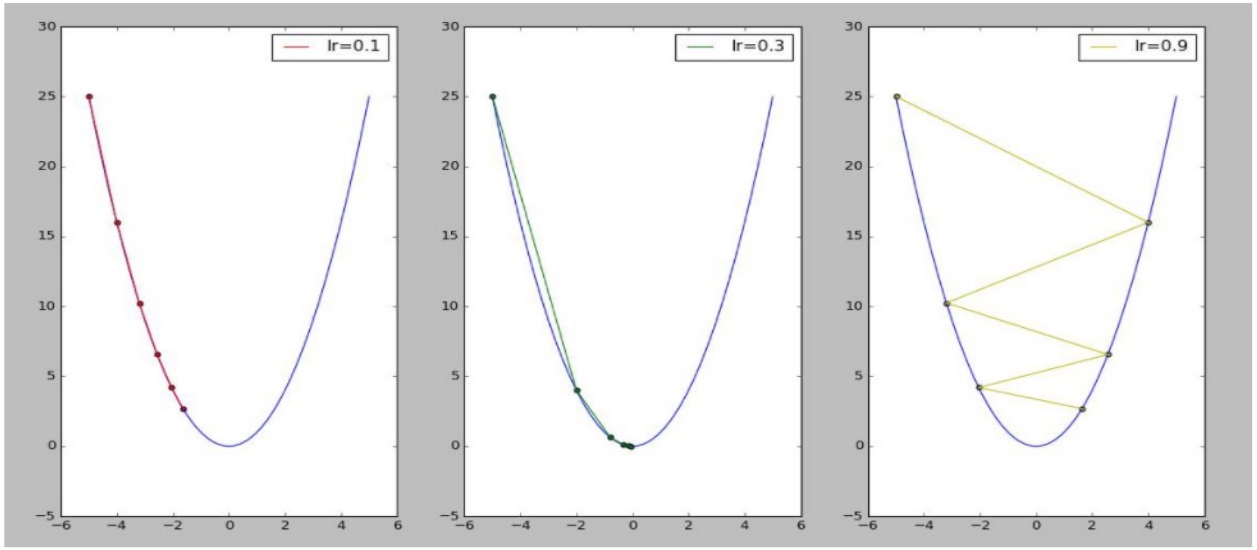
再談：梯度下降 Gradient Descent, GD

前向傳導的計算結果經激勵函數轉換後可得到預測值，再透過損失函數進行偏微分，反向求得最佳的參數(斜率)，直到找出微分等於0的解。



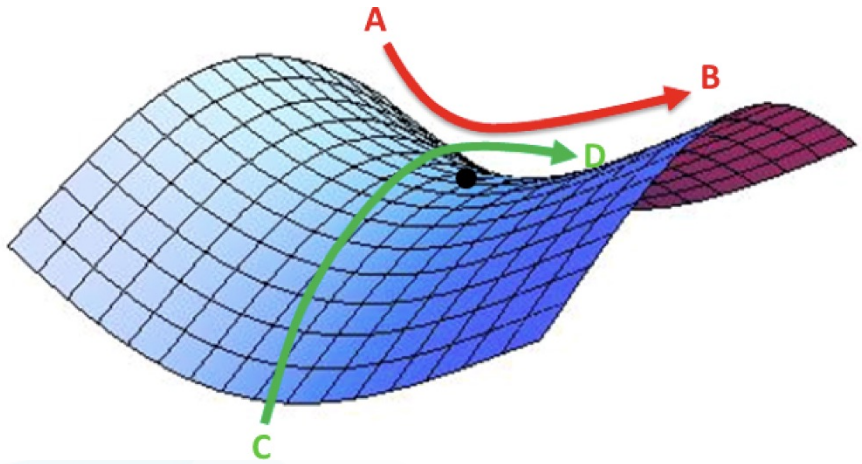
除了方向外，學習率也受到重視：

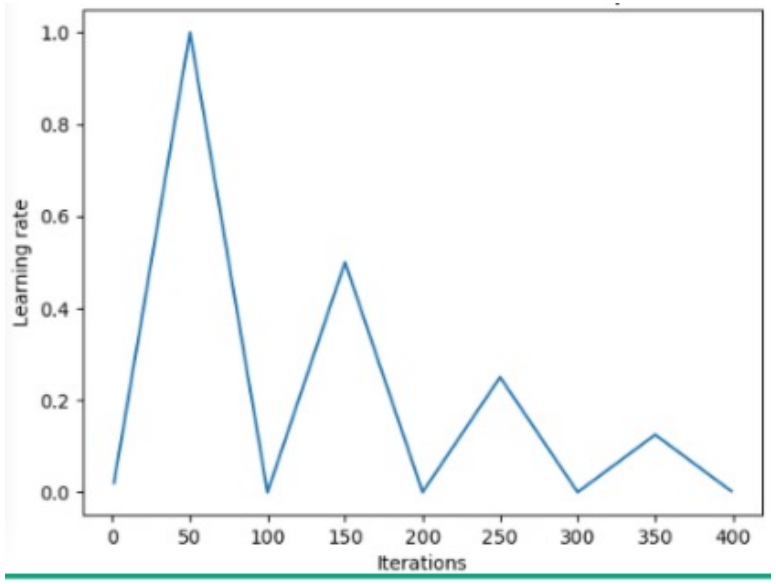
- 操控學習率(Learning rate)，以快速且準確地找到最佳解。
- 學習率過大，會找不到最佳解。
- 學習率過小，訓練速度太慢，或無法收斂。
- 要兼顧速度與準確性，可採取動態設定學習率(Adaptive)，初始較大，越接近最佳解，學習率越小。



馬鞍點 (saddle point)

- 從A往B方向，黑點是最小值。
- 從C往D方向，黑點是最大值。
- 採取週期性學習率。





參閱『Understanding Learning Rates and How It Improves Performance in Deep Learning』

批量梯度下降 Batch Gradient Descent,BGD

最原始形式的梯度下降，當它每次更新參數時都使用所有的樣本來進行更新，導致 BGD 可能會非常慢，而且在訓練集太大而不能全部載入記憶體的時候會很棘手。

BGD 對於凸函數可以得到全域最優點，而對於非凸函數則只能是局部最優點。

隨機梯度下降 Stochastic Gradient Descent, SGD

SGD 每次使用一個訓練樣本和標籤來進行一次參數更新。因此通常 SGD 的速度會非常快。

但也因為 SGD 每次只採一個樣本來參數更新，所以導致目標函數嚴重震盪，可能就跳到更好的局部最優點。

當慢慢的降低學習率時，SGD 擁有和 BGD 一樣的收斂性能，對於非凸和凸曲面幾乎同樣能夠達到局部或全局最優點。

```
tf.keras.optimizers.SGD(
    learning_rate=0.01,
    momentum=0.0,
    nesterov=False,
    weight_decay=None,
    clipnorm=None,
    clipvalue=None,
    global_clipnorm=None,
    use_ema=False,
    ema_momentum=0.99,
    ema_overwrite_frequency=None,
    jit_compile=True,
    name="SGD",
    **kwargs
)
```

加入Momentum 動量的SGD：SGDM

為了抑制SGD的震盪，SGDM認為梯度下降過程可以加入慣性。這就意味著下降方向主要是參考先前累積的下降方向，並且略微偏向。
可以簡單理解為：汽車轉彎時，高速向前的同時加上略微偏向的情形。
加入 momentum 後能夠加速 SGD 求解，並且能夠減少震盪。

Mini-batch 梯度下降，MBGD

MBGD 採取 SGD 和 BGD 折中的辦法，每次從訓練集中取出 batchsize 個樣本作為一個mini-batch，以此來進行一次參數更新。

batch size 越大，批次越少，訓練時間會更快一點，但可能造成資料的很大浪費；而batch size越小，對資料的利用越充分，浪費的資料量越少，但批次會很大，訓練會更耗時。

Adagrad 優化器

在梯度下降更新參數過程中，學習率的操控是重要的，太大會找不到最佳解；太小則造成訓練速度過慢或無法收斂。

Adagrad 又稱自適應學習率梯度下降，主要能夠解決標籤分佈極為不均勻的情況，並自動調整學習率直至收斂，也適合處理稀疏梯度。

但它的缺點是訓練到後期可能造成梯度會趨近於 0 ，導致訓練提前結束。

```
tf.keras.optimizers.Adagrad(
    learning_rate=0.001,
    initial_accumulator_value=0.1,
    epsilon=1e-07,
    weight_decay=None,
    clipnorm=None,
    clipvalue=None,
    global_clipnorm=None,
    use_ema=False,
    ema_momentum=0.99,
    ema_overwrite_frequency=None,
    jit_compile=True,
    name="Adagrad",
    **kwargs
)
```

Adadelta 優化器

這個算法是對 Adagrad 的改進，AdaGrad會累加之前所有的梯度平方，而 Adadelta 只是計算對應的平均值。

特點：

- 訓練初、中期的加速效果不錯，很快。
- 訓練後期則是反覆在局部極小值附近抖動。

```
tf.keras.optimizers.Adadelta(
    learning_rate=0.001,
    rho=0.95,
    epsilon=1e-07,
    weight_decay=None,
    clipnorm=None,
    clipvalue=None,
    global_clipnorm=None,
    use_ema=False,
    ema_momentum=0.99,
    ema_overwrite_frequency=None,
    jit_compile=True,
    name="Adadelta",
    **kwargs
)
```

RMSProp 優化器

RMSProp 類似 AdaGrad，改良學習率調整方式，避免了學習率越來越低的問題，並且依然具備自適應學習率調整的能力。

在實際應用上，RMSProp 已被證明是一種有效的深度學習網路最佳化演算法。

```
tf.keras.optimizers.RMSprop(
    learning_rate=0.001,
    rho=0.9,
    momentum=0.0,
    epsilon=1e-07,
    centered=False,
    weight_decay=None,
    clipnorm=None,
    clipvalue=None,
    global_clipnorm=None,
    use_ema=False,
    ema_momentum=0.99,
    ema_overwrite_frequency=100,
    jit_compile=True,
    name="RMSprop",
    **kwargs
)
```

Adam 優化器

Adam 其實就是加入了動量概念的 RMSprop，且在更新梯度過程中考慮了偏差校正 (bias-correction)。

Adam 結合了 Adagrad、RMSprop 及 Momentum 的優點，能有效控制學習率步長和梯度方向，防止梯度的振盪和在鞍點的靜止。

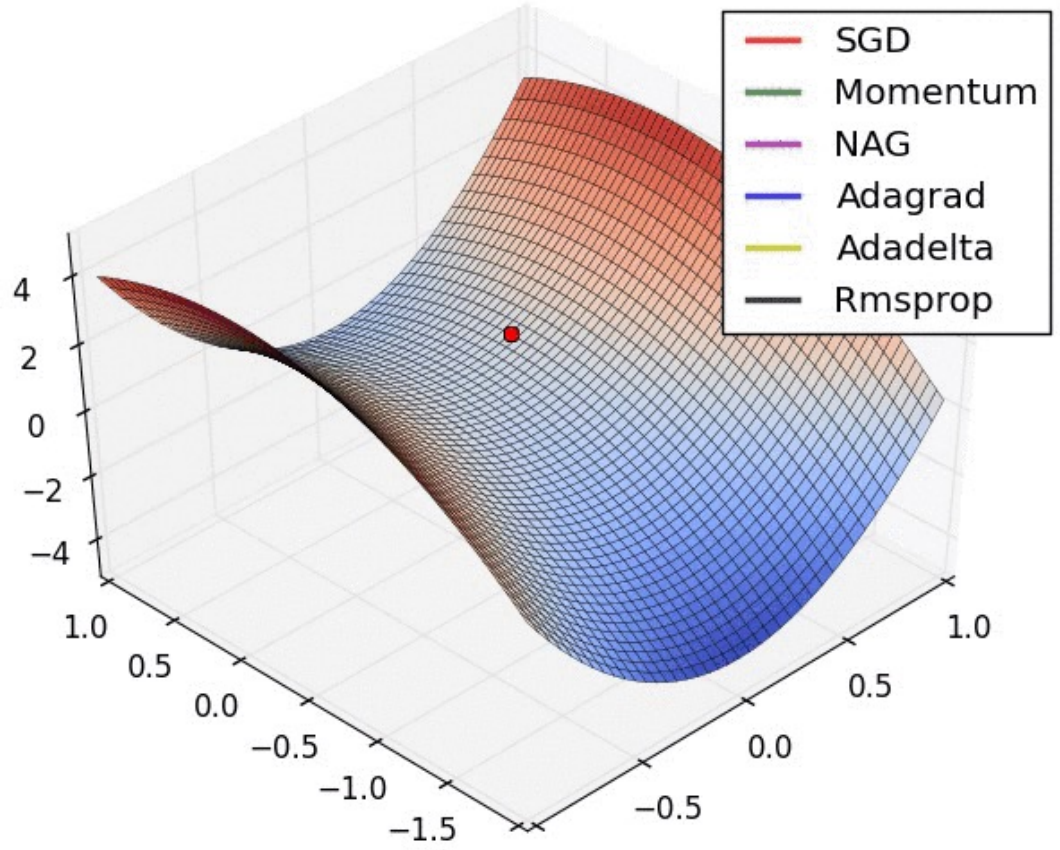
Adam 各種狀況均適用，是目前較為推薦的優化方式。

```
tf.keras.optimizers.Adam(
    learning_rate=0.001,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07,
    amsgrad=False,
    weight_decay=None,
    clipnorm=None,
    clipvalue=None,
    global_clipnorm=None,
    use_ema=False,
    ema_momentum=0.99,
    ema_overwrite_frequency=None,
    jit_compile=True,
    name="Adam",
    **kwargs
)
```

梯度下降公式：

Name	Update Rule
SGD	$\Delta\theta_t = -\alpha g_t$
Momentum	$m_t = \gamma m_{t-1} + (1-\gamma) g_t,$ $\Delta\theta_t = -\alpha\, m_t$
Adagrad	$G_t = G_{t-1} + g_t^2,$ $\Delta\theta_t = -\alpha\, g_t G_t^{-1/2}$
Adadelta	$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$ $\Delta\theta_t = -\alpha g_t v_t^{-1/2} D_{t-1}^{1/2},$ $D_t = \beta_1 D_{t-1} + (1-\beta_1)(\Delta\theta_t/\alpha)^2$
RMSprop	$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$ $\Delta\theta_t = -\alpha\, g_t v_t^{-1/2}$
Adam	$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t,$ $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$ $\hat{m}_t = m_t/(1-\beta_1^t),$ $\hat{v}_t = v_t/(1-\beta_2^t),$ $\Delta\theta_t = -\alpha\, \hat{m}_t \hat{v}_t^{-1/2}$

梯度下降優化速度與正確性比較：



tf_lr_finder.ipynb

動畫參考：[關於深度學習優化器 optimizer 的選擇，你需要了解這些]

TensorFlow提供的優化器：link

- SGD
- RMSprop
- Adam
- AdamW
- Adadelta
- Adagrad
- Adamax
- Adafactor
- Nadam
- Ftrl

優化器程式寫法：

```
model.compile(loss='sparse_categorical_crossentropy’,
              optimizer='sgd’,
              metrics= ['accuracy’])

-----------------------------------------------------------
opt = keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='sparse_categorical_crossentropy',
              optimizer= opt,
              metrics= ['accuracy’])

-----------------------------------------------------------
model.compile(loss='sparse_categorical_crossentropy’,
       optimizer=tf.keras.optimizers.SGD(learning_rate=0.01),
       metrics= ['accuracy'])
```

MLP_clf_optimizer.ipynb

深度神經網路注意事項

在深度神經網路訓練時常出現下列情形：

1. 梯度消失、梯度爆炸
2. 訓練速度過於緩慢
3. 過於擬合(Overfitting)
4. 訓練資料不足

梯度消失、梯度爆炸

梯度消失 (vanishing gradients)

指權重梯度隨著訓練愈來愈小，最後梯度為 0(消失)。因此權重無法更新，最終所得到的權重並非最佳，即代入損失函數，所得到的loss 值並非最小。

梯度爆炸 (exploding gradients)

指權重梯度會隨著訓練次數愈來愈大，導致梯度下降法無法收斂。

解決方法：

- 權重初始化 (Weight initialization)
- 不飽和觸發函數 (Nonsaturating activation function)
- 批次正規化 (Batch normcilization)
- 梯度修剪 (Gradient clipping)

訓練速度過於緩慢

解決方法：

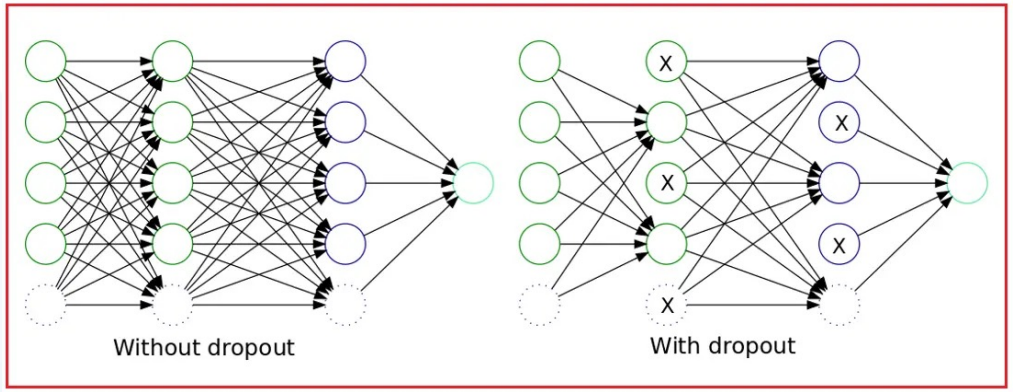
- 使用比一般梯度下降法更快速的優化法 (Optimizer)
  - 動量優化法 (Momentum Optimization)
  - Nesterov加速梯度 (NAG, Nesterov Accelerated Gradient)
  - AdaGrad (Adaptative Gradient)
  - RMSProp (Root Mean Square Propagation)
  - Adam (Adaptive Moment Estimation)
  - AdaMax
  - Nadam
- 使用學習速率排程 (Learning rate scheduling)

過於擬合(Overfitting)

解決方法：

- 正則化 (Regularization)
- Dropout

在訓練時每一次的迭代 (epoch)皆以一定的比率丟棄隱藏層神經元，而被丟棄的神經元不會傳遞訊息。
在反向傳播時，被丟棄的神經元其梯度是 0，所以在訓練時不會過度依賴某一些神經元，藉此達到對抗過擬合的效果。




```
tf.keras.layers.Dense(
    units,
    activation=None,
    use_bias=True,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

官網

```
from tensorflow.keras import layers
from tensorflow.keras import regularizers

layer = layers.Dense(
    units=64,
    kernel_regularizer=regularizers.L1L2(l1=1e-5, l2=1e-4),
    bias_regularizer=regularizers.L2(1e-4),
    activity_regularizer=regularizers.L2(1e-5)
)
```

```
tf.keras.layers.Dropout(rate, noise_shape=None, seed=None, **kwargs)
```

訓練資料不足

解決方法：

- 遷移學習 (Transfer Learning)

  利用已訓練好模型的較底層（ lower layers ），來建構新的模型。

綜合練習：

- 梯度消失_爆炸.ipynb
- 學習速率排程.ipynb

Keras 建模方式01

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential(name="my_example_model")
model.add(layers.Dense(64, activation="relu", name="my_first_layer"))
model.add(layers.Dense(10, activation="softmax", name="my_last_layer"))

model.build((None, 3))
model.summary()

#=================================================
model = keras.Sequential()
model.add(keras.Input(shape=(3,)))
model.add(layers.Dense(64, activation="relu"))
model.add(layers.Dense(10, activation="softmax"))

model.summary()
```

nn_codeDemo01.ipynb

Keras 建模方式02

```
from tensorflow import keras
from tensorflow.keras import layers

vocabulary_size = 10000
num_tags = 100
num_departments = 4

title = keras.Input(shape=(vocabulary_size,), name="title")
text_body = keras.Input(shape=(vocabulary_size,), name="text_body")
tags = keras.Input(shape=(num_tags,), name="tags")
features = layers.Concatenate()([title, text_body, tags])

features = layers.Dense(64, activation="relu")(features)
priority = layers.Dense(1, activation="sigmoid",
                        name="priority")(features)
department = layers.Dense(
num_departments, activation="softmax", name="department")(features)

model = keras.Model(inputs=[title, text_body, tags],
                    outputs=[priority, department])

model.summary()
```

nn_codeDemo02.ipynb

Keras 建模方式03

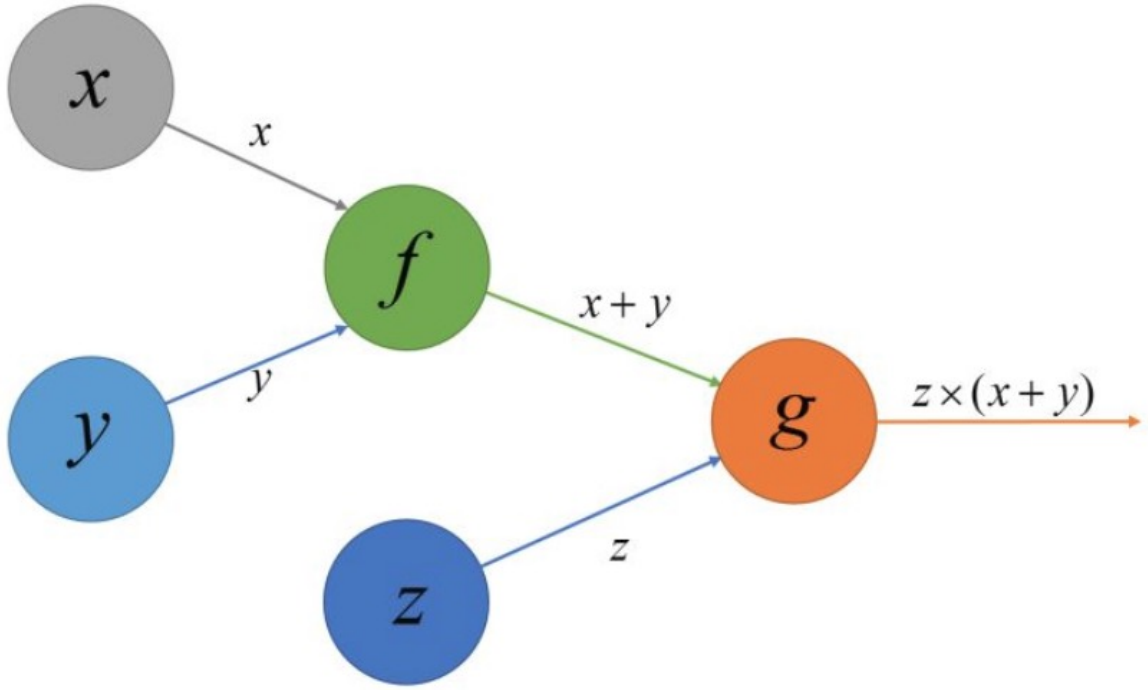
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense

model = Sequential([
        #第一層：將 28*28攤平成一維度
        Flatten(input_shape=x_train.shape[1:]),
        #第二層
        Dense (units=300, activation='relu'),
        Dense (units=200, activation='relu'),
        Dense (units=100, activation='relu'),
        #輸出層：10類別，10個神經元
        Dense (units=10, activation='softmax')
])

model.summary()
```

MLP_clf.ipynb

手算前向/反向傳導

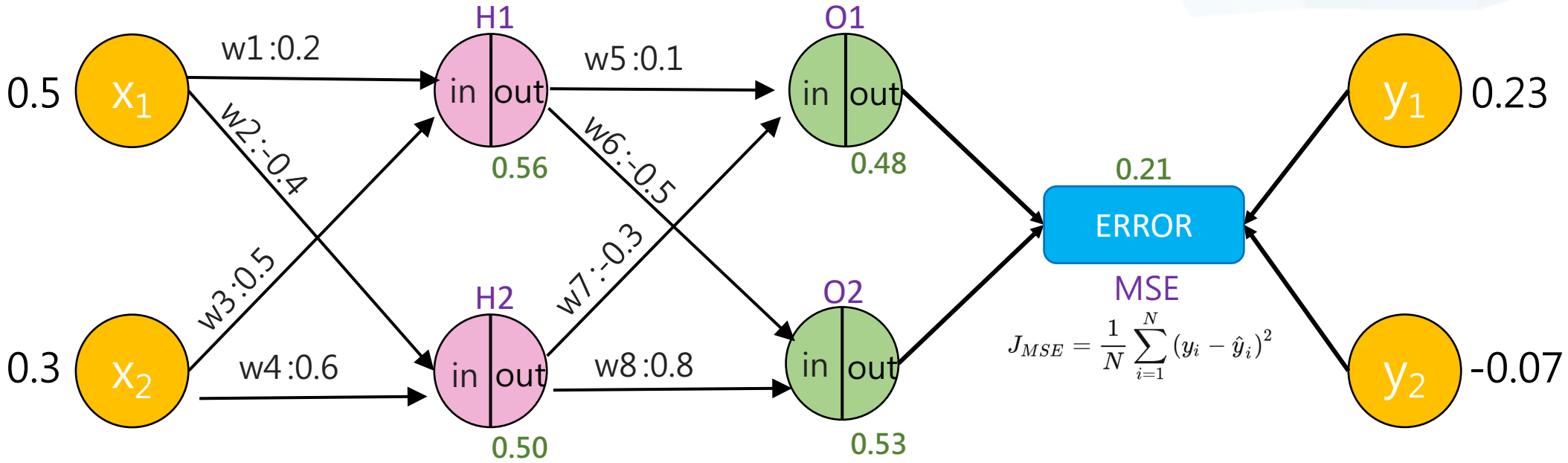
利用偏微分的連鎖律可一次更新所有神經層的權重。

$$\frac{\partial Loss}{\partial x} = \frac{\partial Loss}{\partial g}\frac{\partial g}{\partial f}\frac{\partial f}{\partial x}$$



TF_02_自動微分.ipynb

為減化過程，先忽略 bias，著重於權重的更新



正向傳導

H1out = sigmoid (0.2x0.5 + 0.5x0.3) = 0.56

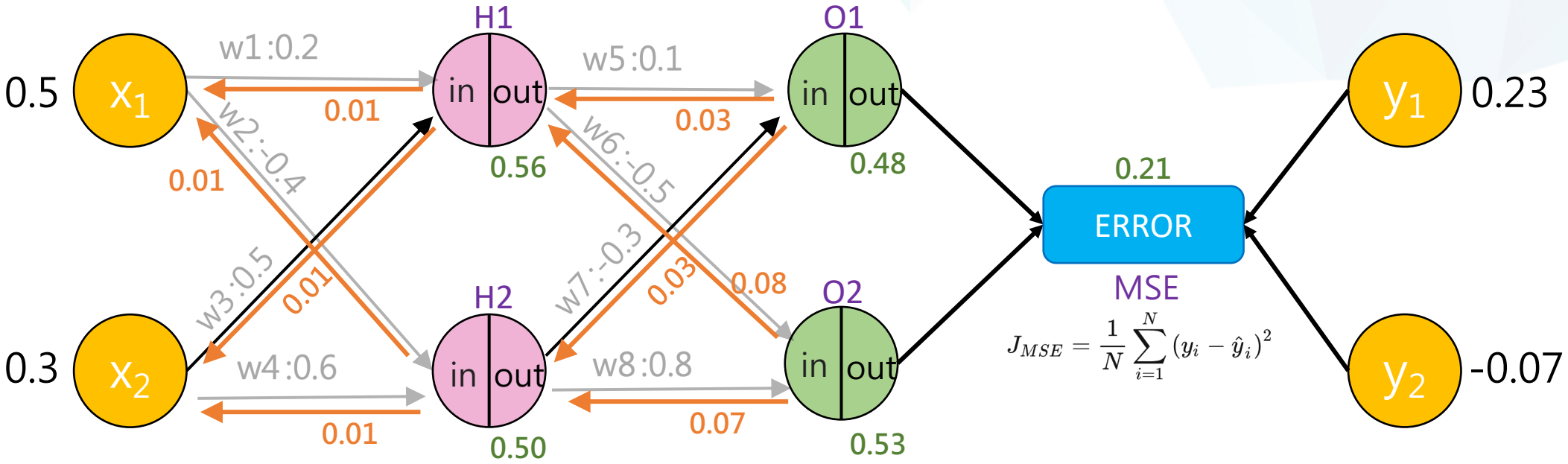
H2out = sigmoid (-0.4x0.5 + 0.6x0.3) = 0.50

O1out = stgmoid (0.1x0.56 + (-0.3x0.50) = 0.48

O2out = stgmoid (-0.5x0.56 + 0.8x0.50) =0.53

Error = $1/2(0.48-0.23)^2 + 1/2(0.53-(-0.07))^2=0.21$

為減化過程，先忽略 bias，著重於權重的更新



反向傳導

$\delta O1 = \frac{\partial Error}{\partial O1} = O1-y1 = 0.48-0.23 = 0.25$

$\delta O2 = \frac{\partial Error}{\partial O2} = O2-y2 = 0.53-(-0.07) = 0.60$

$\delta w5 = \frac{\partial Error}{\partial w5} = \frac{\partial Error}{\partial O1} * \frac{\partial O1}{\partial O1in} * \frac{\partial O1in}{\partial w5}$

$= 0.25*0.48*(1-0.48)*0.56 = 0.03$

$\delta w7 = \frac{\partial Error}{\partial w7} = \frac{\partial Error}{\partial O1} * \frac{\partial O1}{\partial O1in} * \frac{\partial O1in}{\partial w7}$

$= 0.25*0.48*(1-0.48)*0.50 = 0.03$

$\delta w6 = \frac{\partial Error}{\partial w6} = \frac{\partial Error}{\partial O2} * \frac{\partial O2}{\partial O2in} * \frac{\partial O2in}{\partial w6} = 0.60*0.53*(1-0.53)*0.56 = 0.08$

$\delta w8 = \frac{\partial Error}{\partial w8} = \frac{\partial Error}{\partial O2} * \frac{\partial O2}{\partial O2in} * \frac{\partial O2in}{\partial w8} = 0.60*0.53*(1-0.53)*0.50 = 0.07$

$\delta w1 = \frac{\partial Error}{\partial w1} = \delta w5+\delta w6 * \frac{\partial H1}{\partial H1in} * \frac{\partial H1in}{\partial w1} = (0.03+0.08)*0.56*(1-0.56)*0.5 = 0.01$

$\delta w3 = \frac{\partial Error}{\partial w3} = \delta w5+\delta w6 * \frac{\partial H1}{\partial H1in} * \frac{\partial H1in}{\partial w3} = (0.03+0.08)*0.56*(1-0.56)*0.3 = 0.01$

$\delta w2 = \frac{\partial Error}{\partial w2} = \delta w7+\delta w8 * \frac{\partial H2}{\partial H2in} * \frac{\partial H2in}{\partial w2} = (0.03+0.07)*0.50*(1-0.50)*0.5 = 0.01$

$\delta w4 = \frac{\partial Error}{\partial w4} = \delta w7+\delta w8 * \frac{\partial H2}{\partial H2in} * \frac{\partial H2in}{\partial w4} = (0.03+0.07)*0.50*(1-0.50)*0.3 = 0.01$

本章節結束