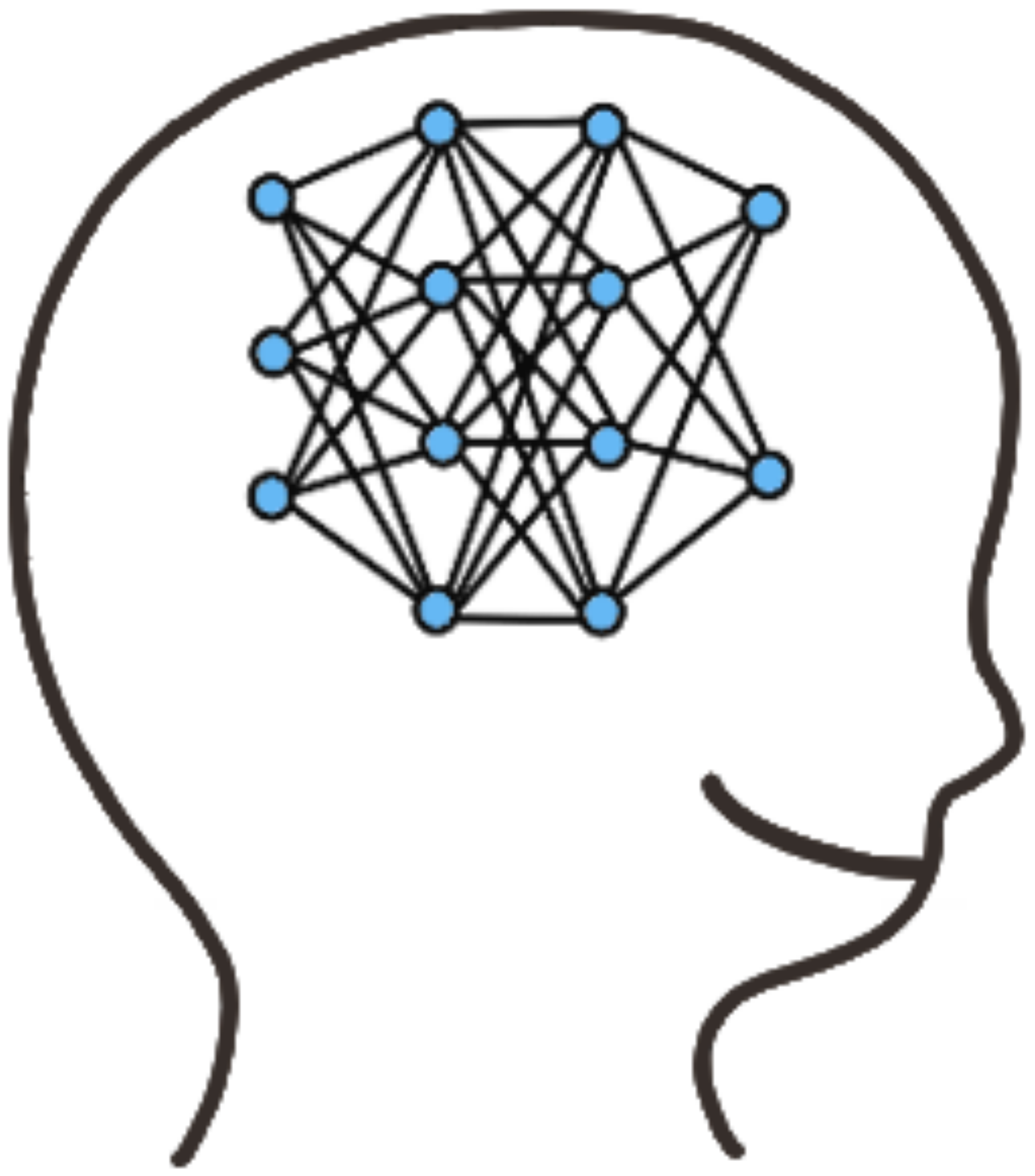
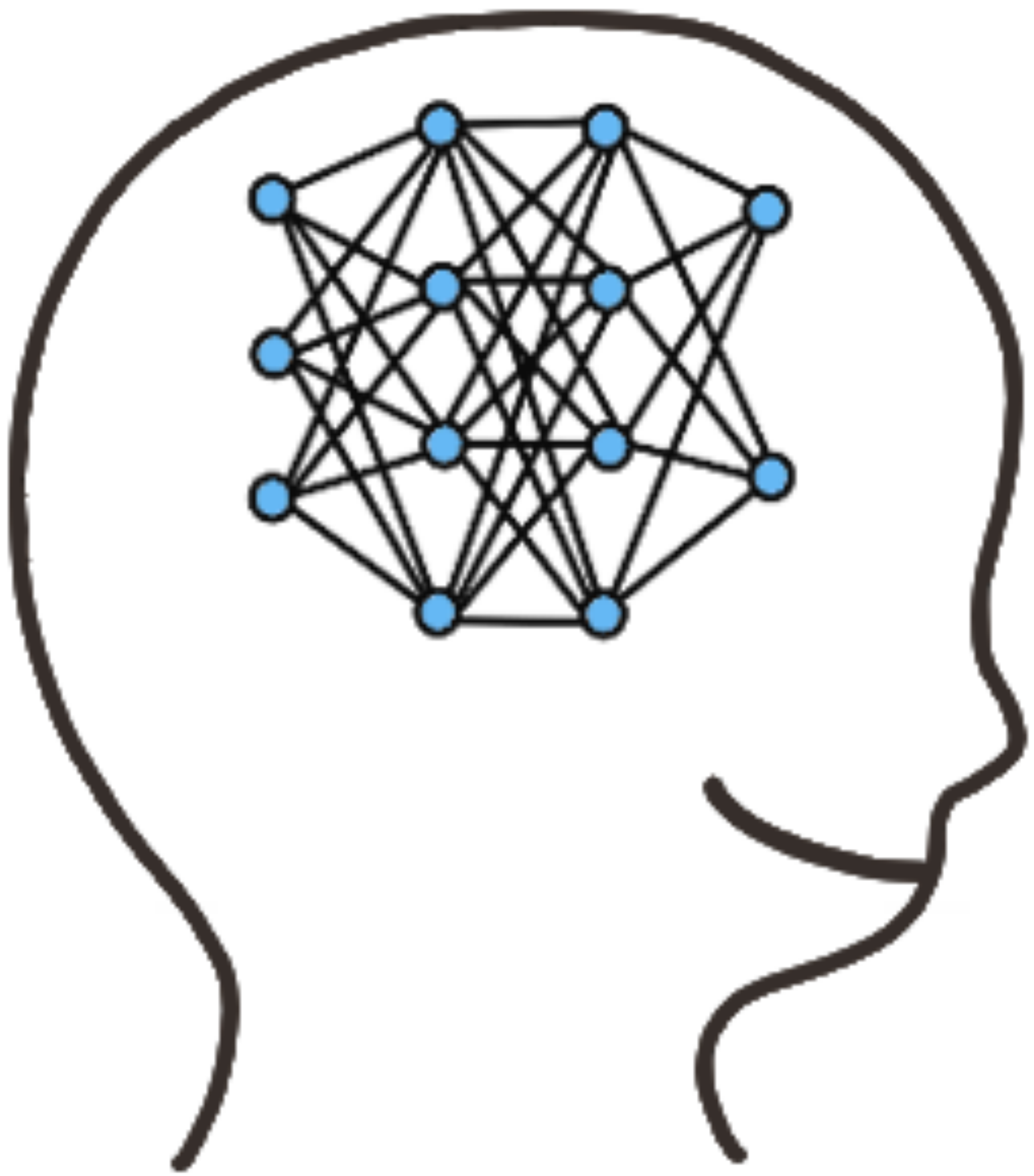


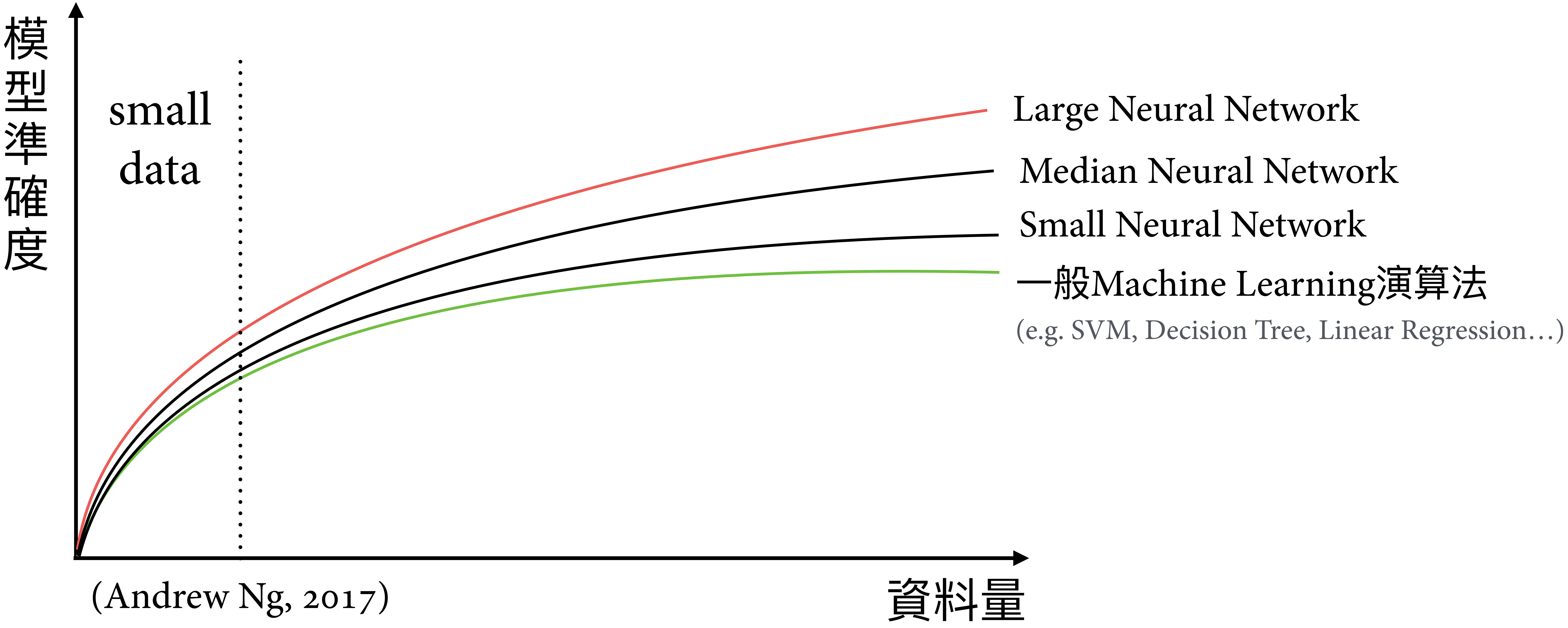
深度學習
Deep Learning



機器學習 vs. 深度學習

Machine Learning vs. Deep Learning

深度學習



(Andrew Ng, 2017)

機器學習 vs. 深度學習

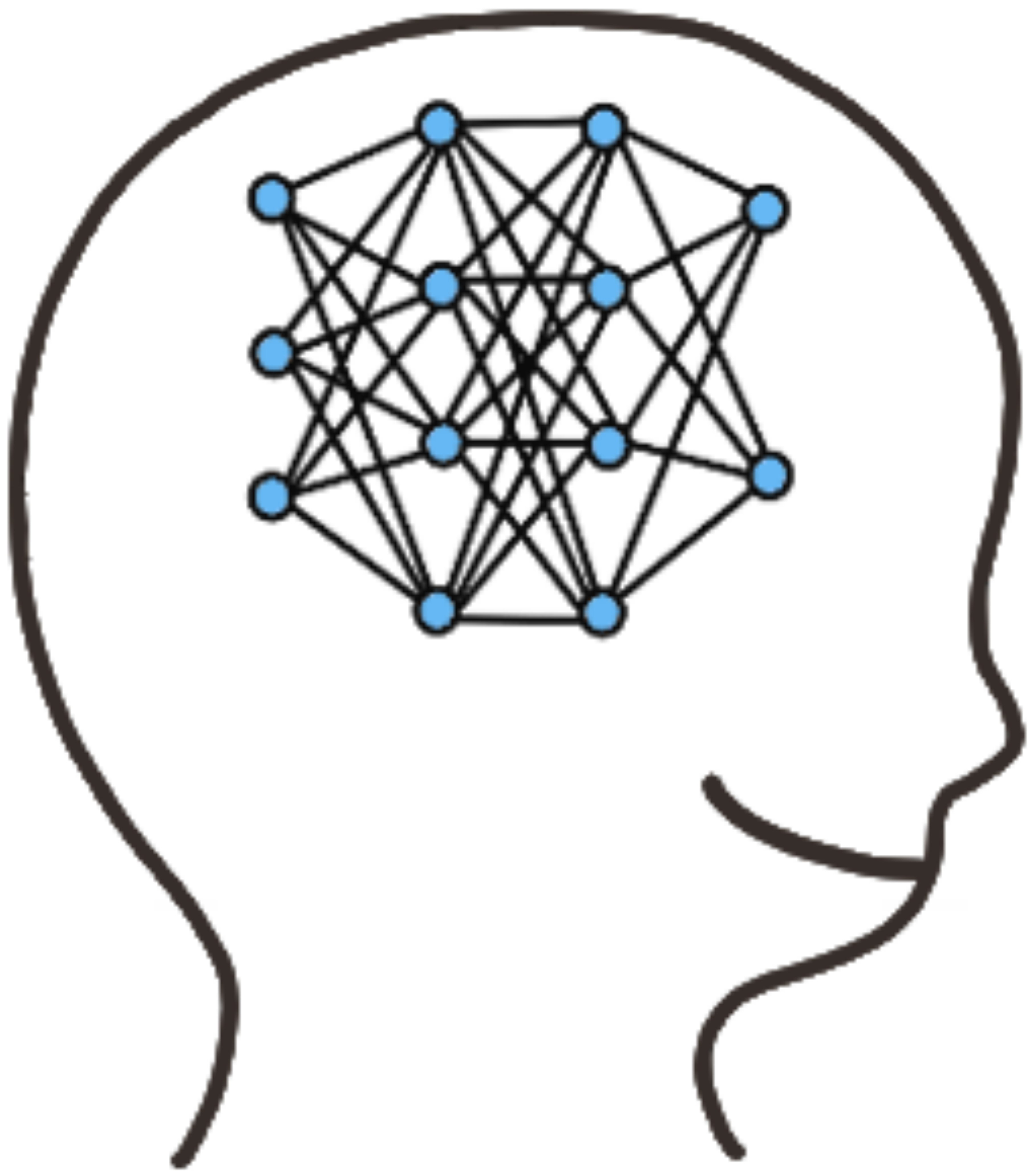
- 深度學習可處理高維度特徵、非結構化的資料建立高複雜度模型
- 機器學習**仰賴**人為處理過較好的特徵
- 深度學習**降低**人為處理特徵的需要，透過複雜的神經網路組合特徵

深度學習成功應用

- Google Deep Mind - AlphaGo
- Google Translate
- Apple - Siri
- Facebook 相片自動標註人名

驅動深度學習的關鍵因素

- **大數據**：越大量的數據可以讓深度學習的效果越好
- **硬體運算能力**：深度學習的建模速度比機器學習慢很多(數分鐘到數天都有可能)，所以仰賴硬體運算能力的支援如: 夠好的CPU/GPU
- **演算法**：良好的深度學習架構不僅能提高建模速度，更能有效的擬合特定應用之資料，例如：卷積神經網路(CNN)適合用於影像辨識

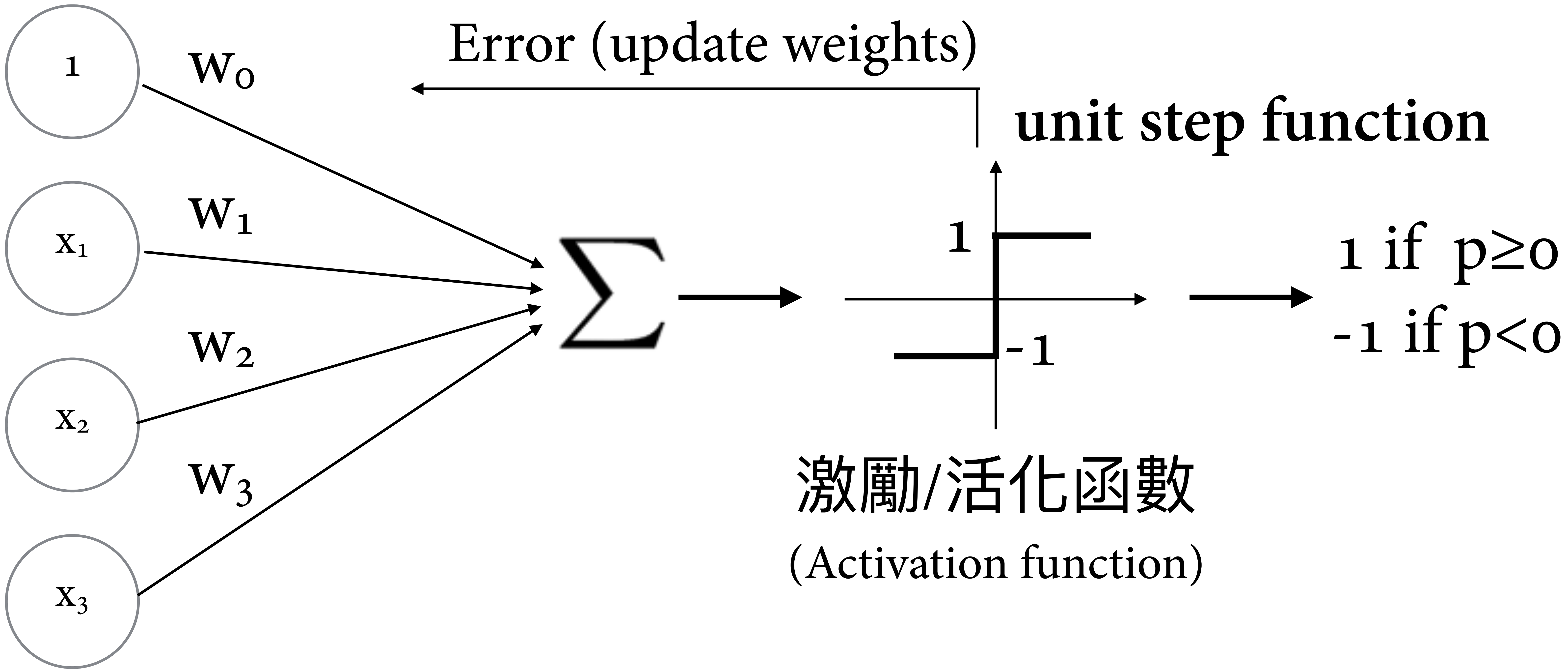


感知器

Perceptron

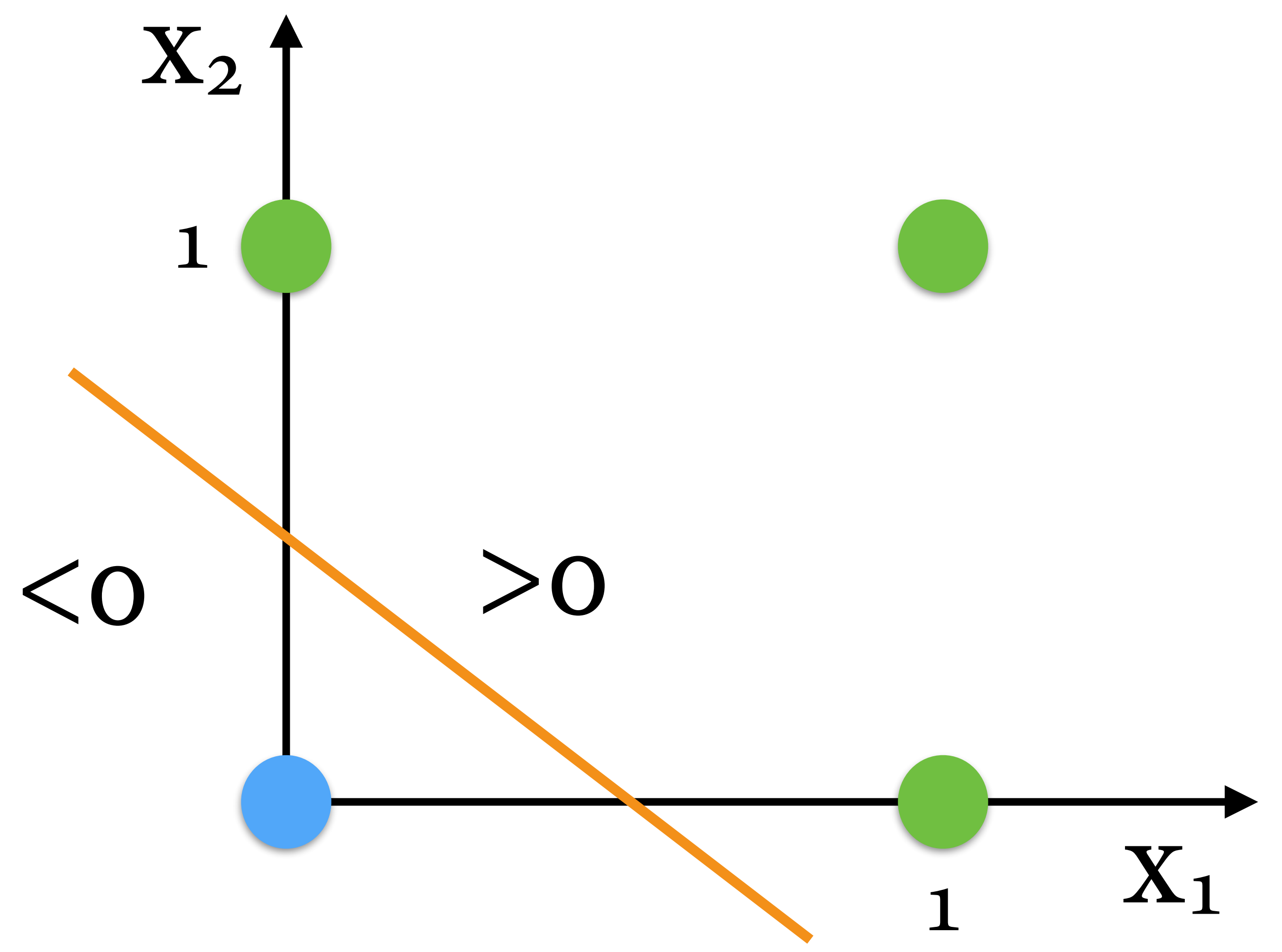
感知器 (Perceptron)

- Frank Rosenblatt (1957)

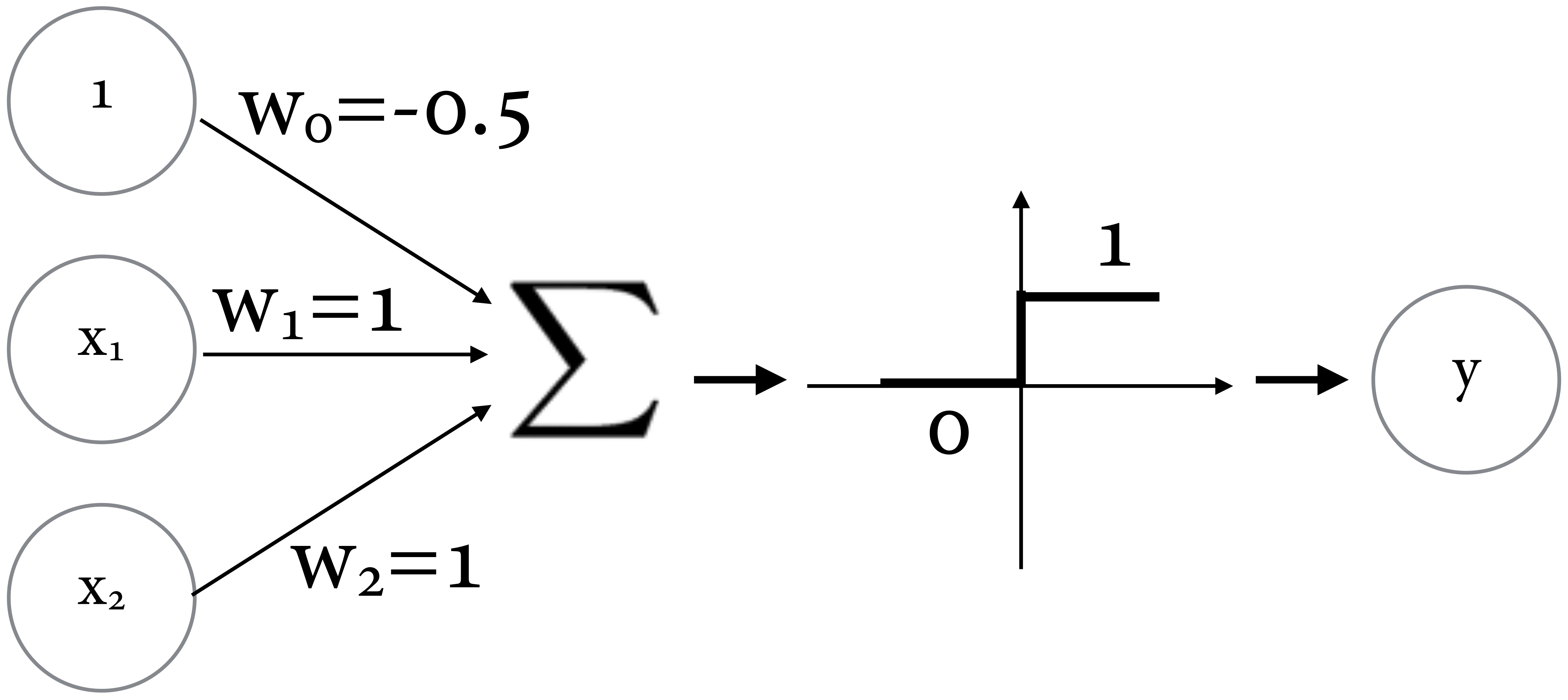


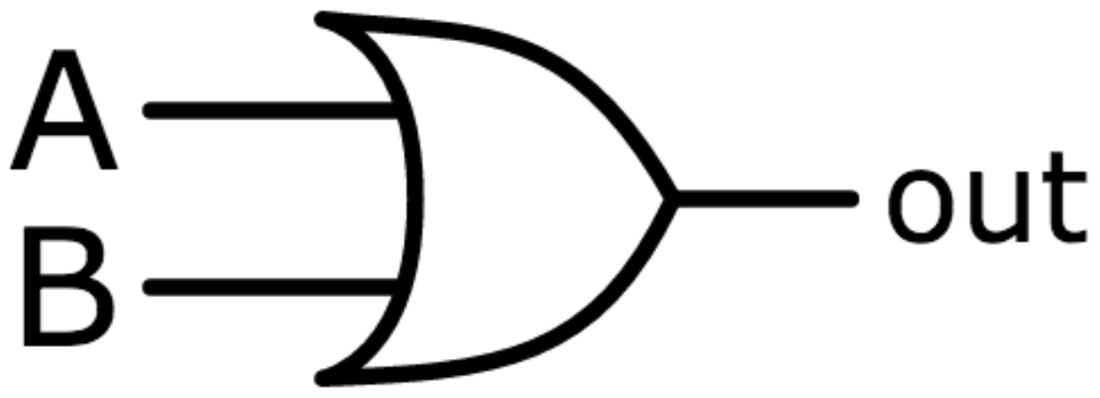
感知器 (Perceptron)

- 類似邏輯閘：訊號輸入、輸出 —> OR Gate 分類問題



$x_1 + x_2 - 0.5 = 0$

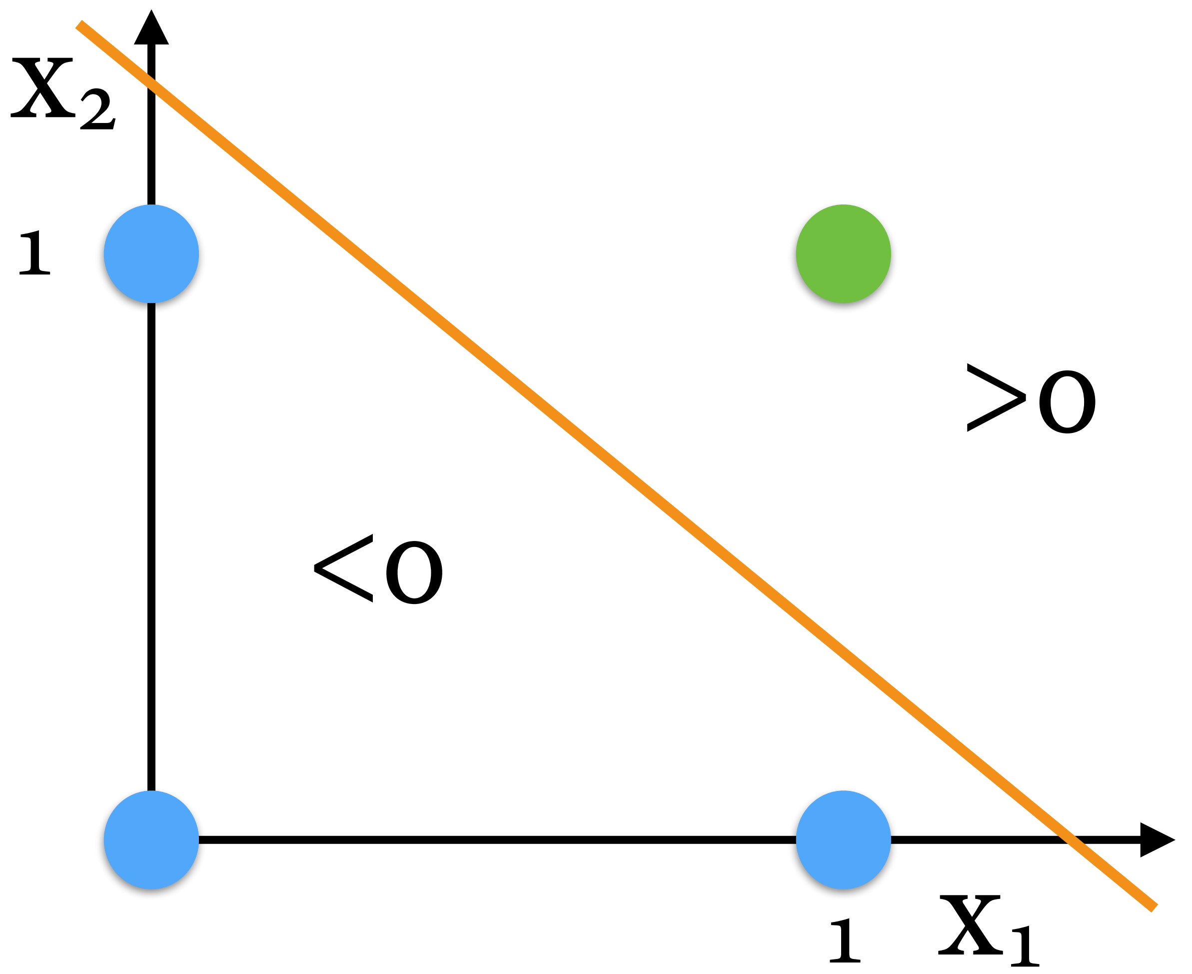




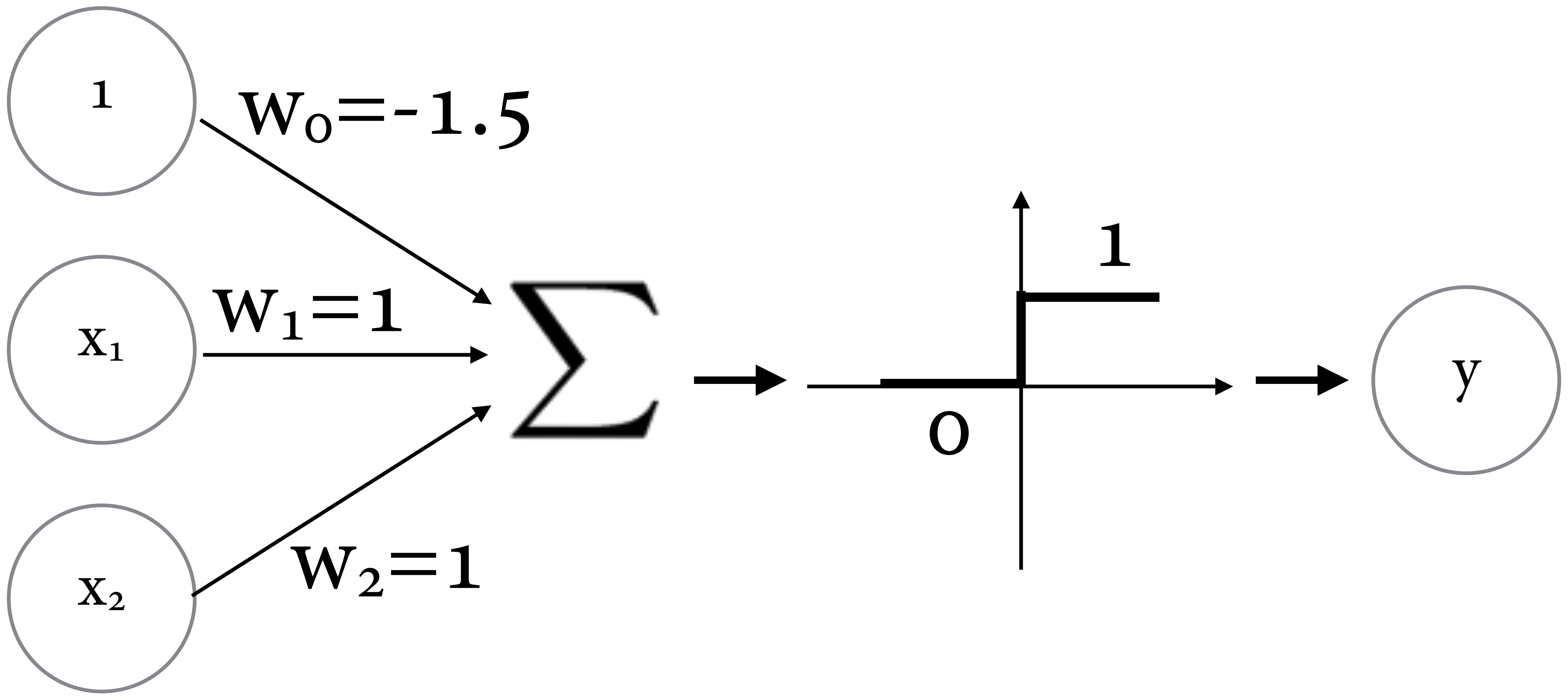
A	B	out
0	0	0
0	1	1
1	0	1
1	1	1

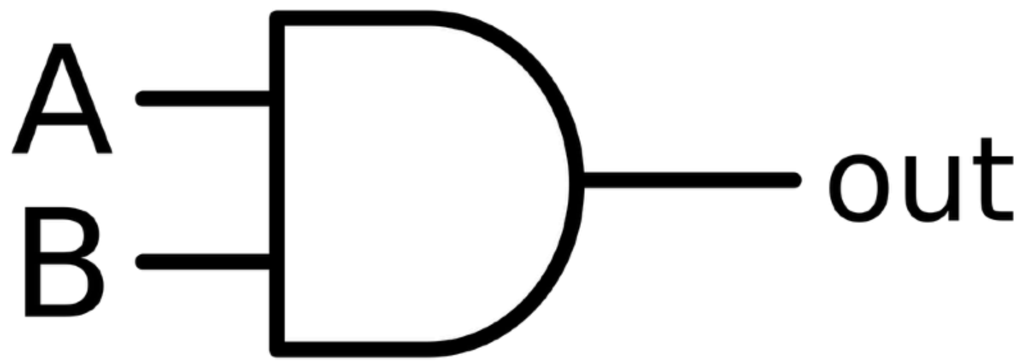
感知器 (Perceptron)

- AND Gate 分類問題



$x_1 + x_2 - 1.5 = 0$

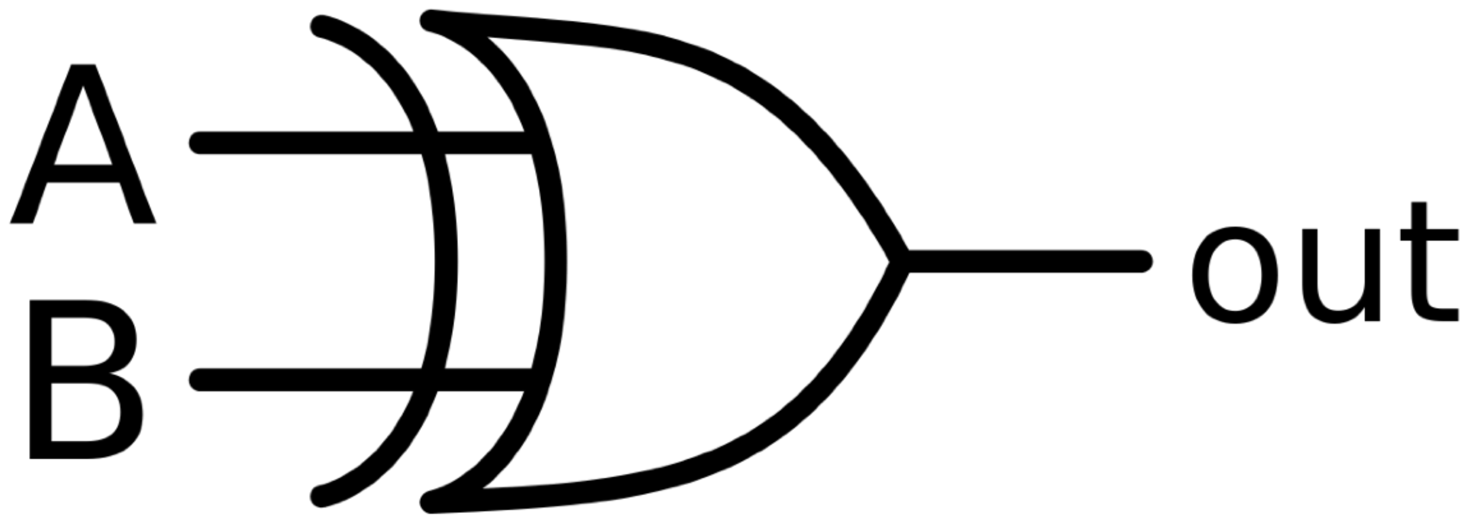




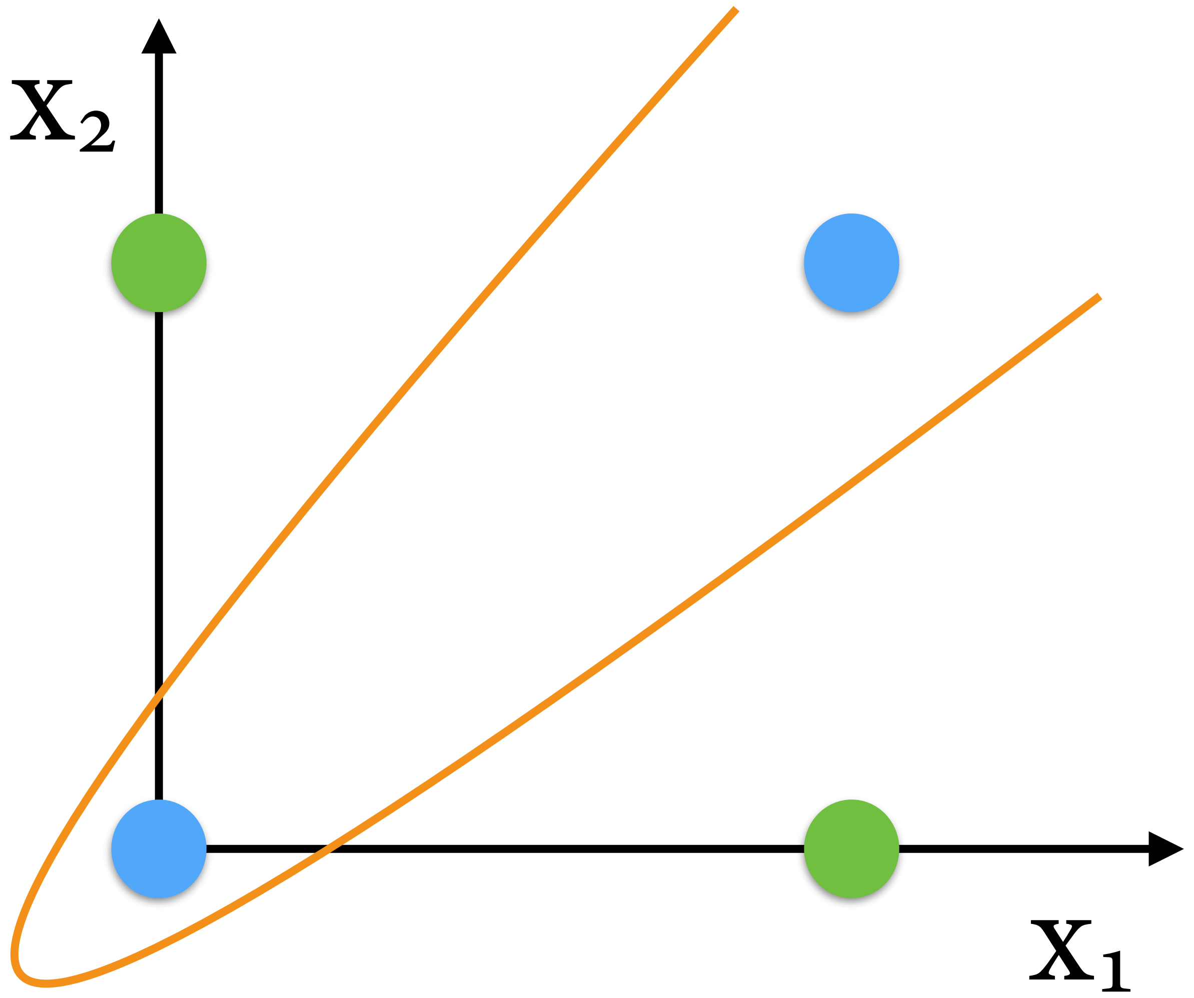
A	B	out
0	0	0
0	1	0
1	0	0
1	1	1

單層Perceptron的限制

- XOR Gate



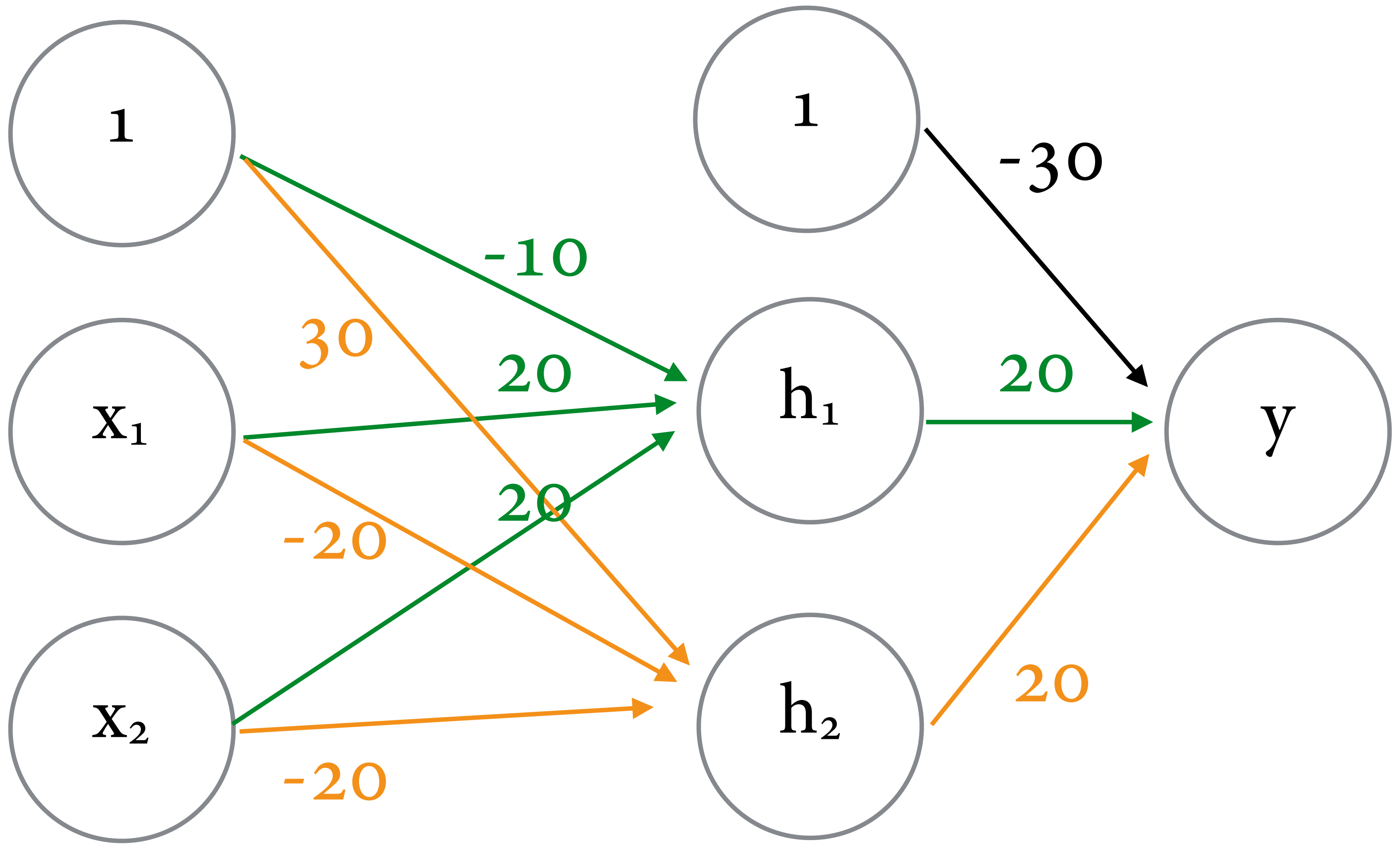
A	B	out
0	0	0
0	1	1
1	0	1
1	1	0



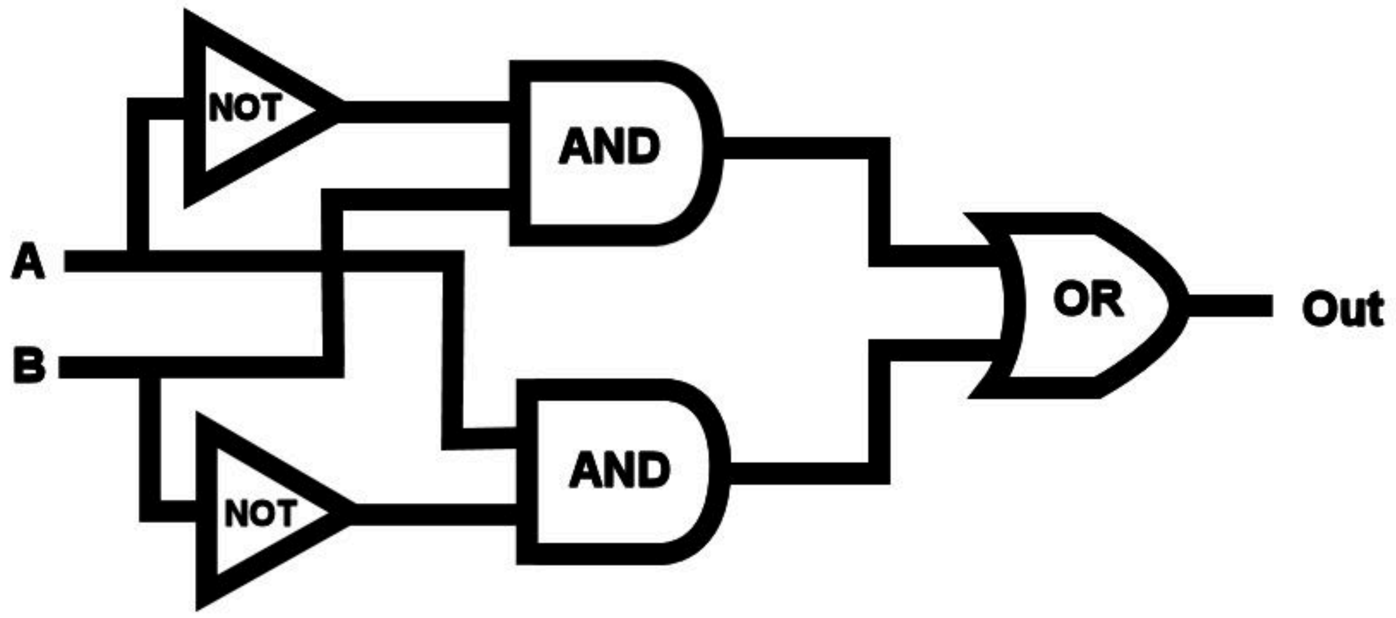
非線性可分

多層感知器(Multi-layered Perceptron)

- XOR Gate



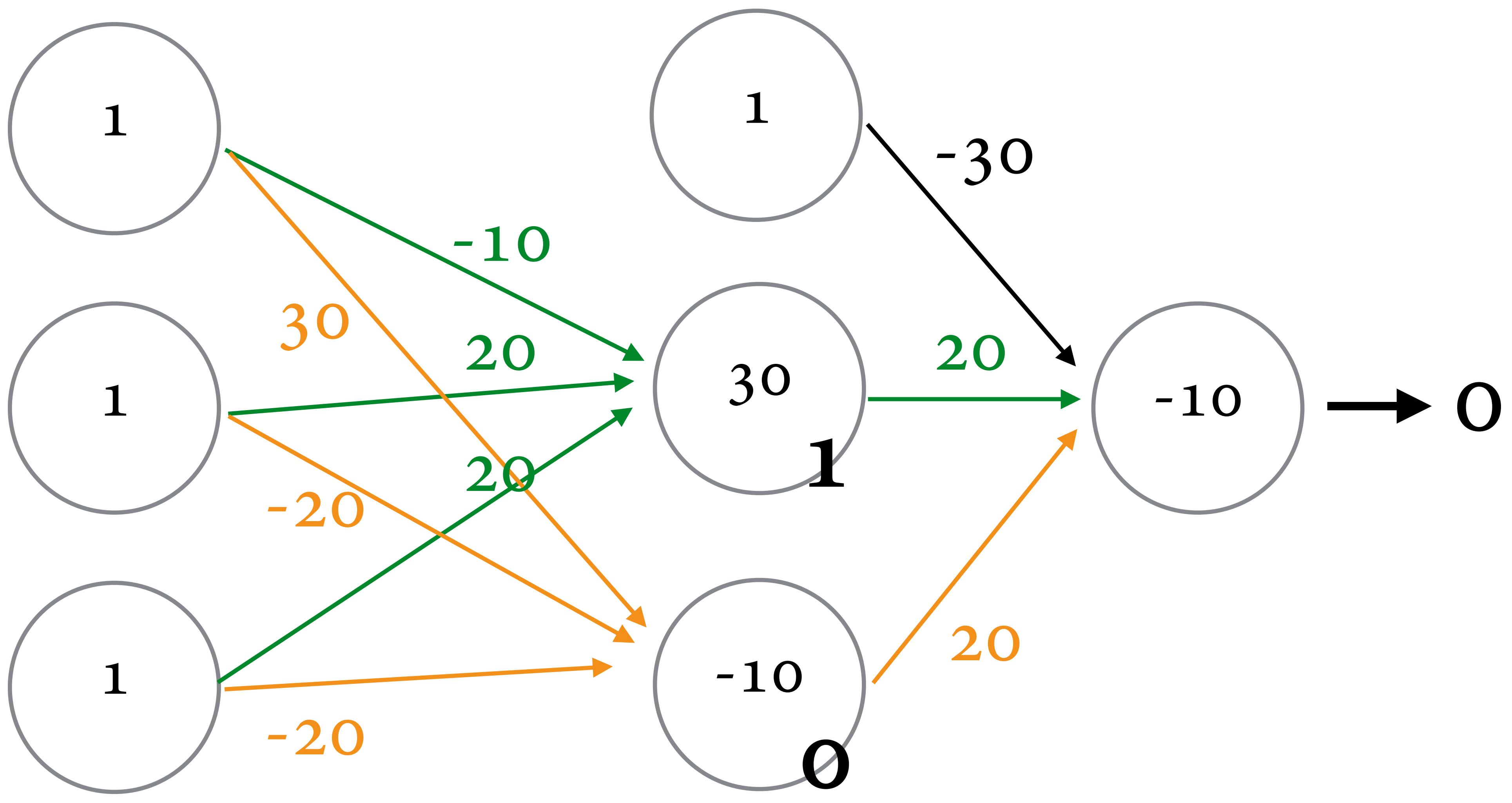
$$h_1 = 20x_1 + 20x_2 - 10$$
$$h_2 = -20x_1 - 20x_2 + 30$$
$$y = 20h_1 + 20h_2 - 30$$

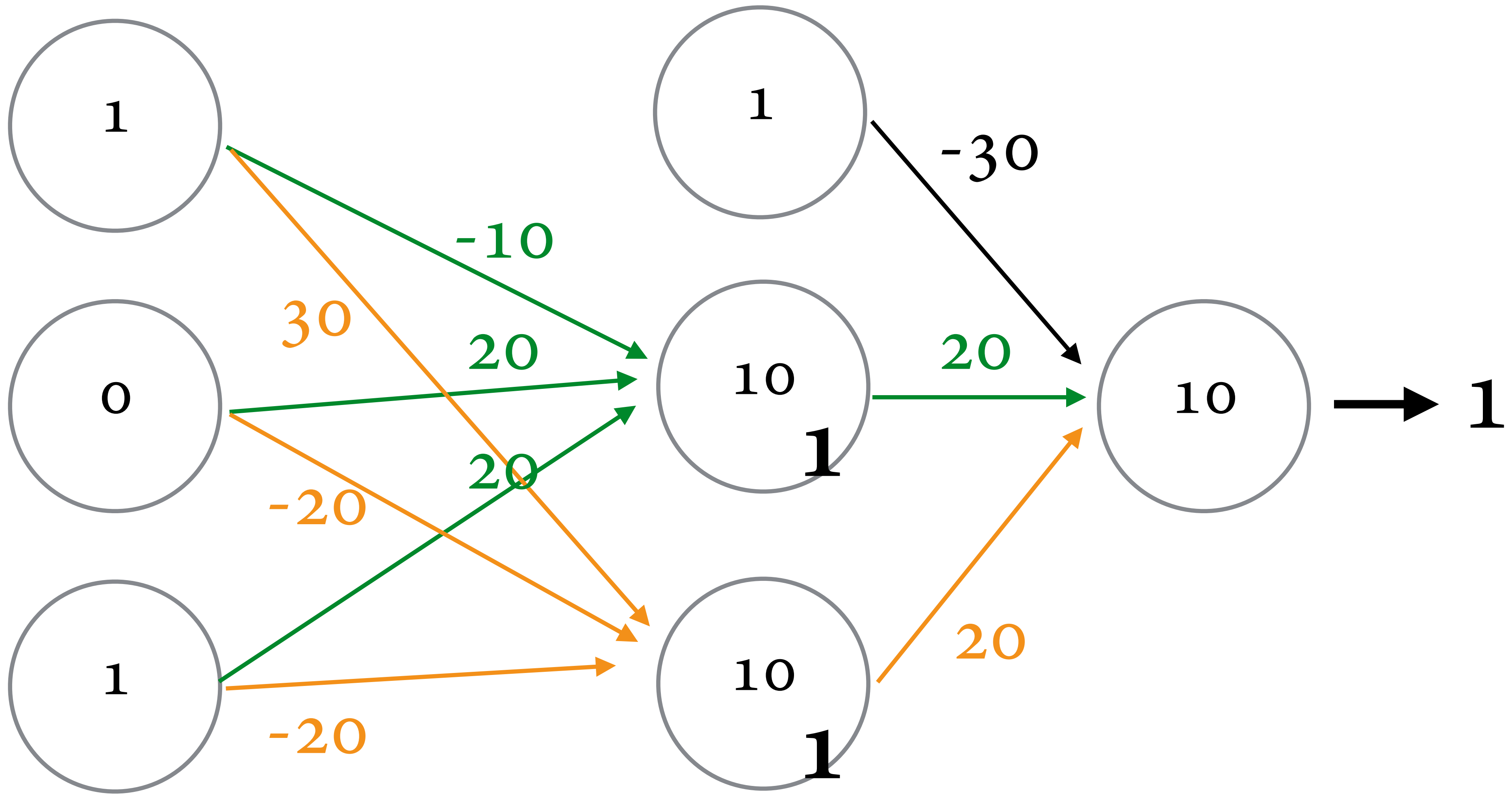


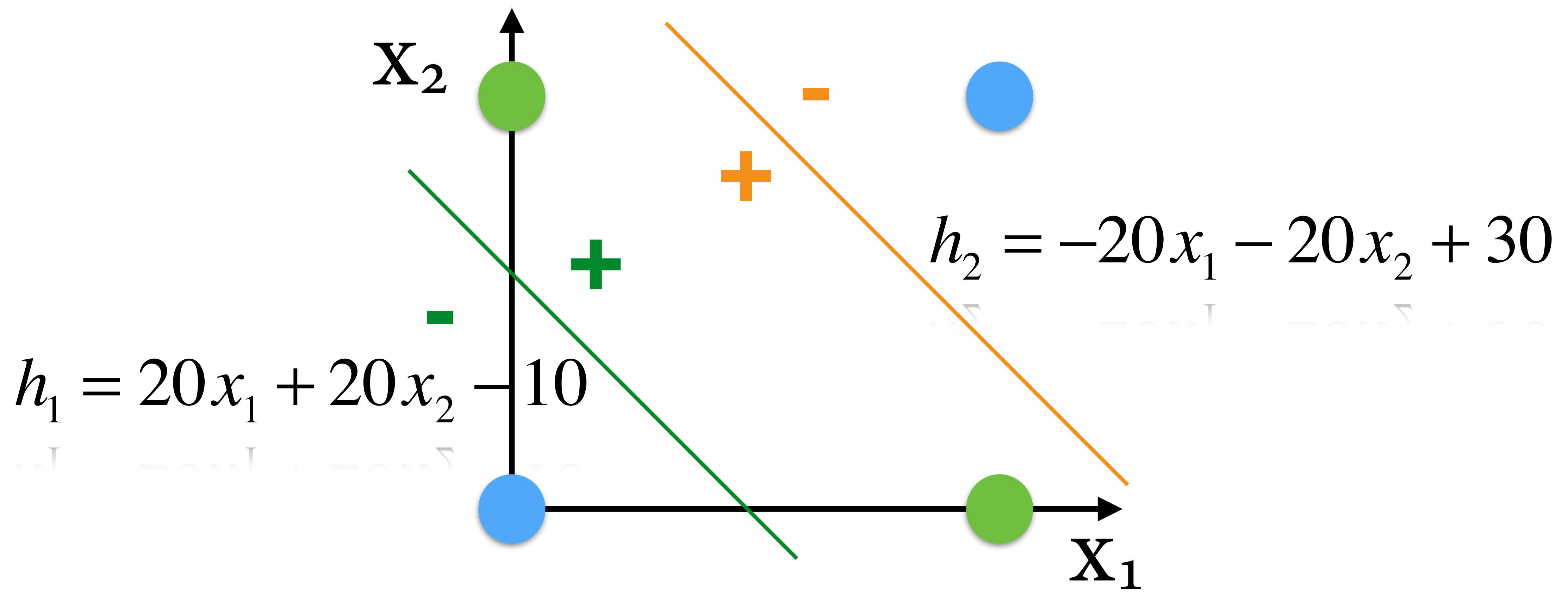
A	B	out
0	0	0
0	1	1
1	0	1
1	1	0

多層感知器(Multi-layered Perceptron)

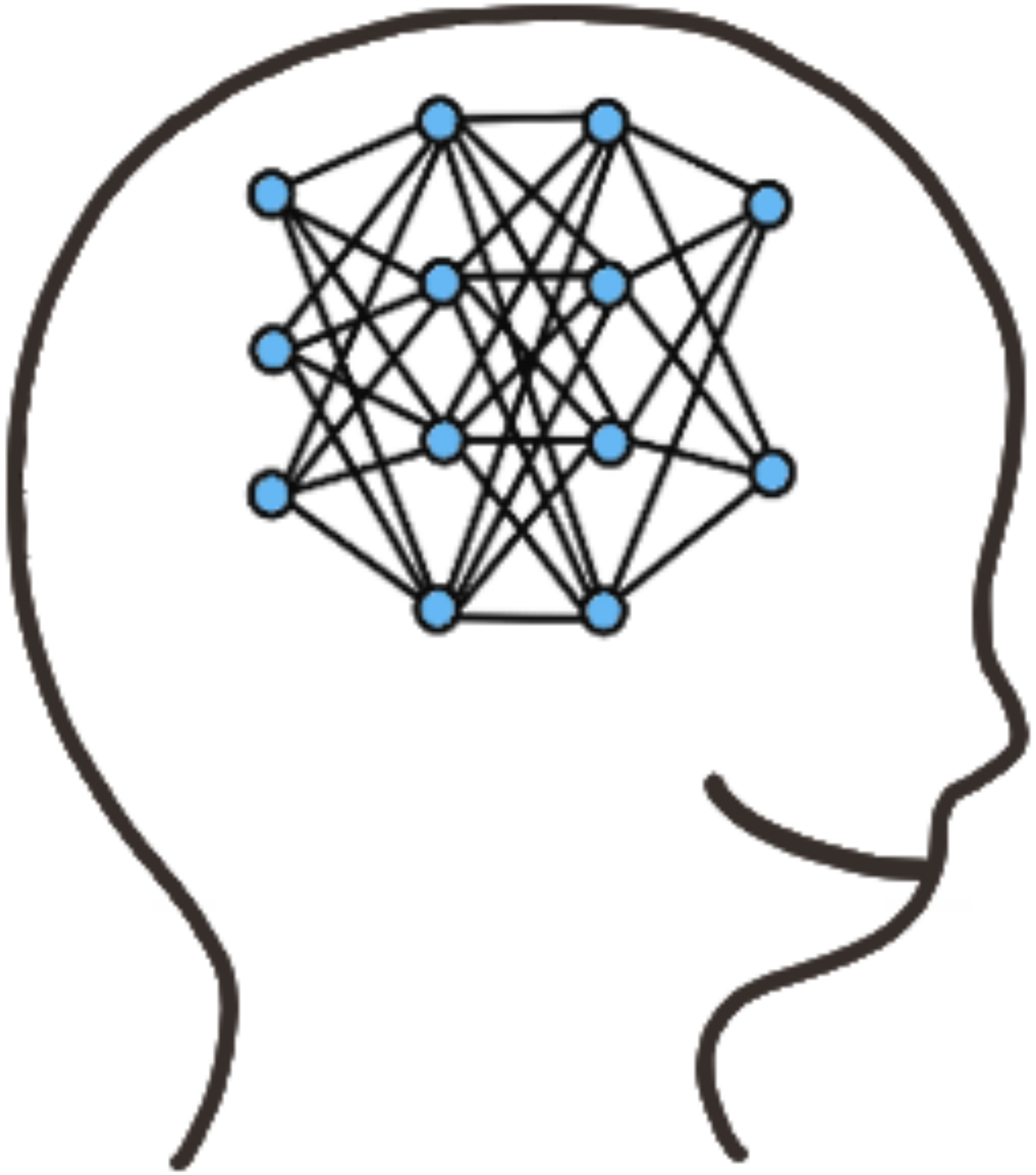
- XOR Gate







A	B	out
0	0	0
0	1	1
1	0	1
1	1	0

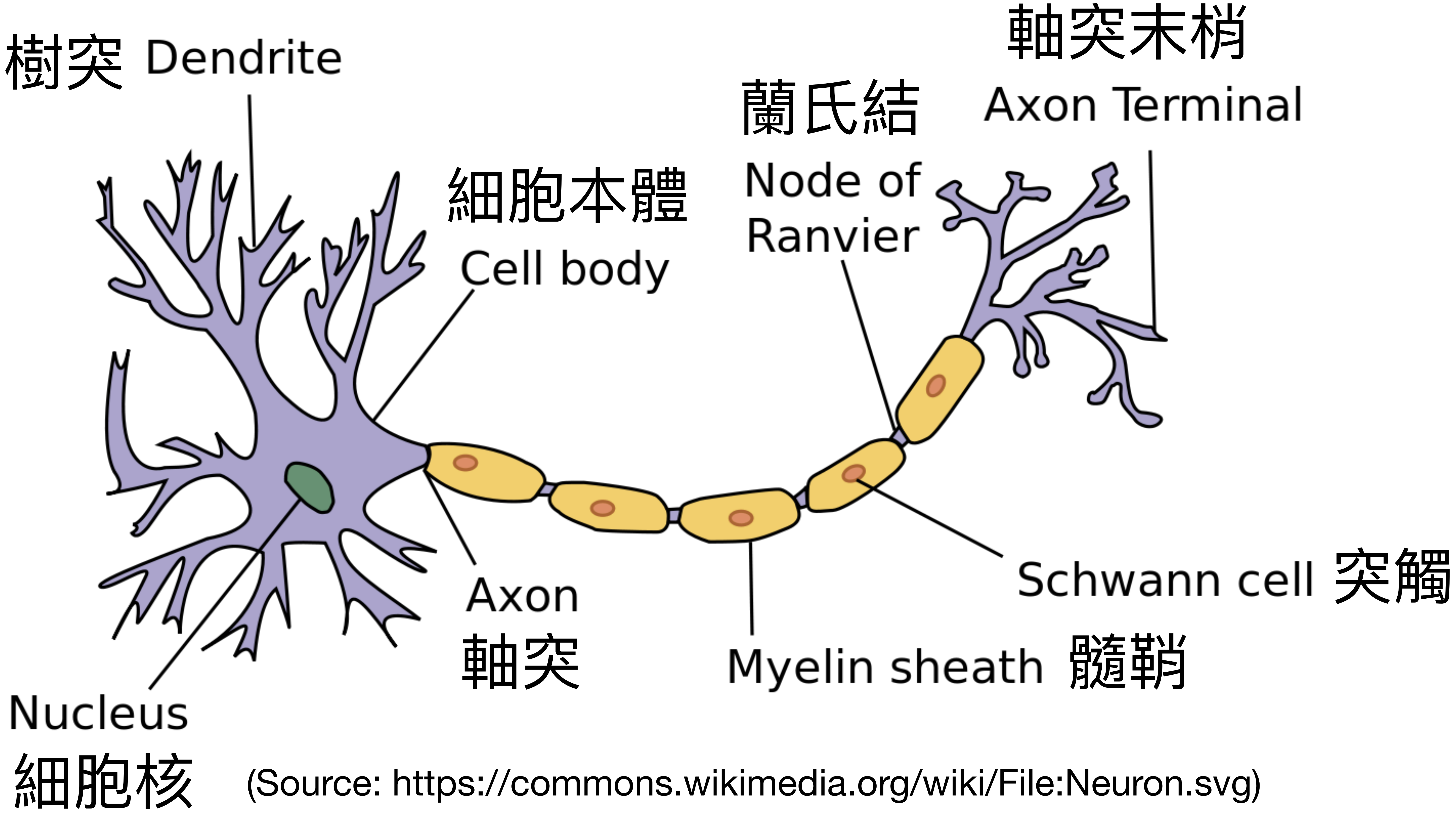


神經網路

Neural Network

Neuron

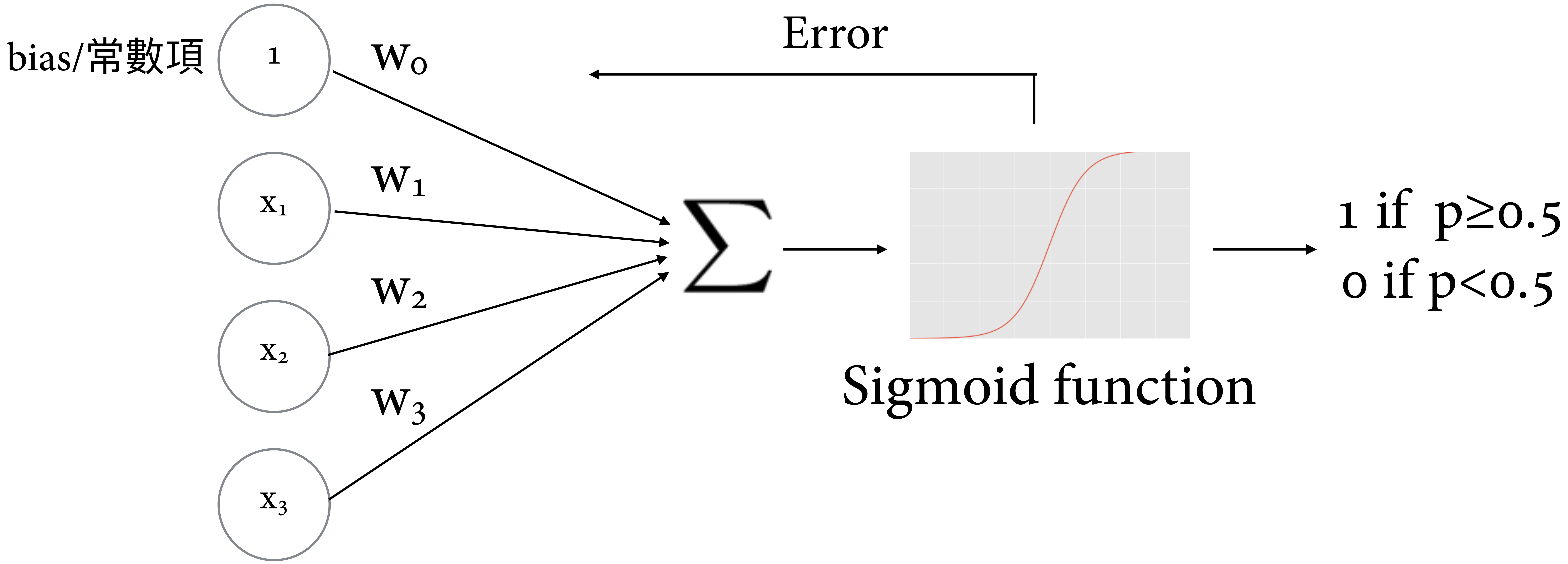
- 模擬大腦神經元(Neuron)運作方式



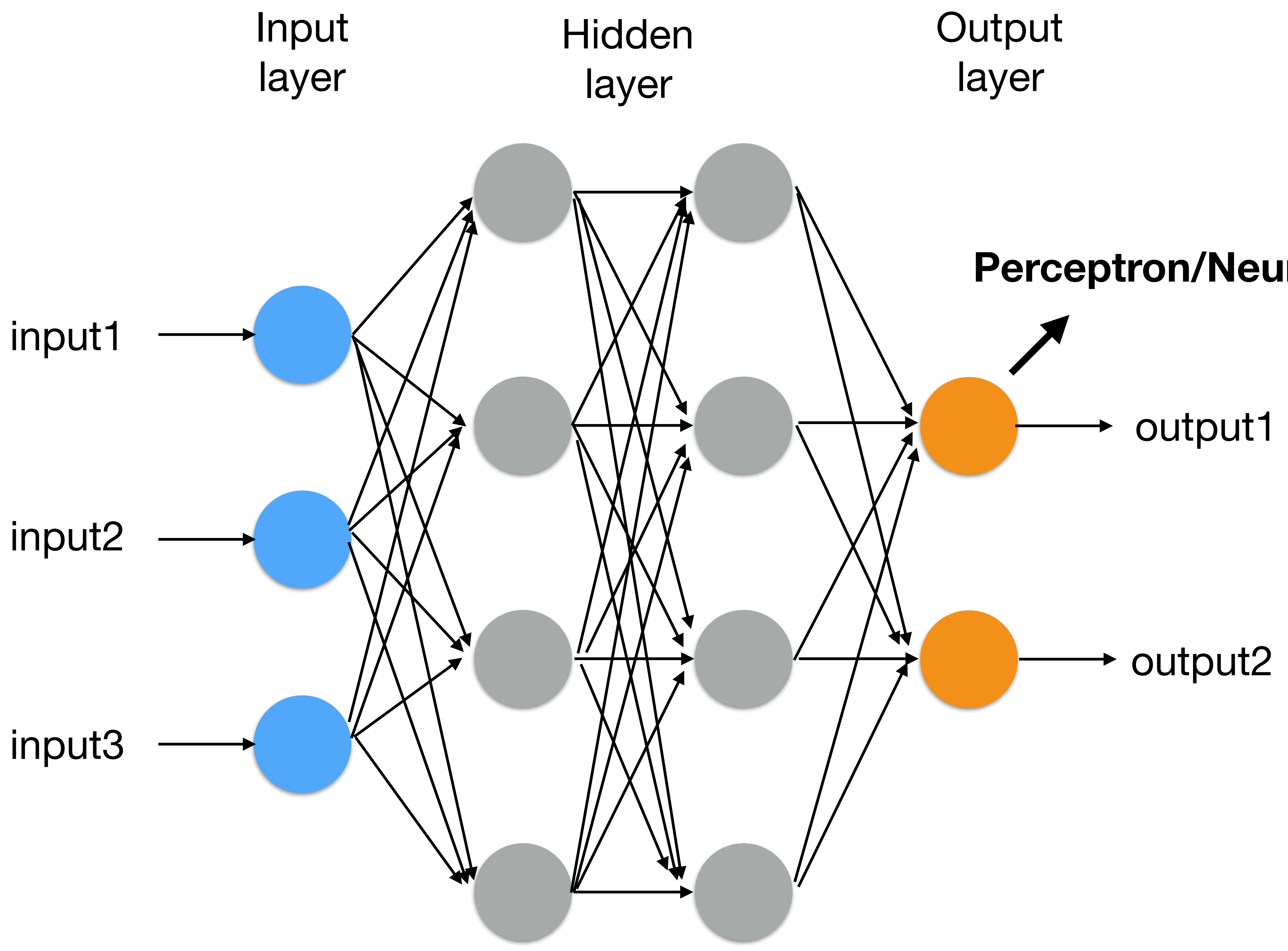
(Source: https://commons.wikimedia.org/wiki/File:Neuron.svg)

Neuron

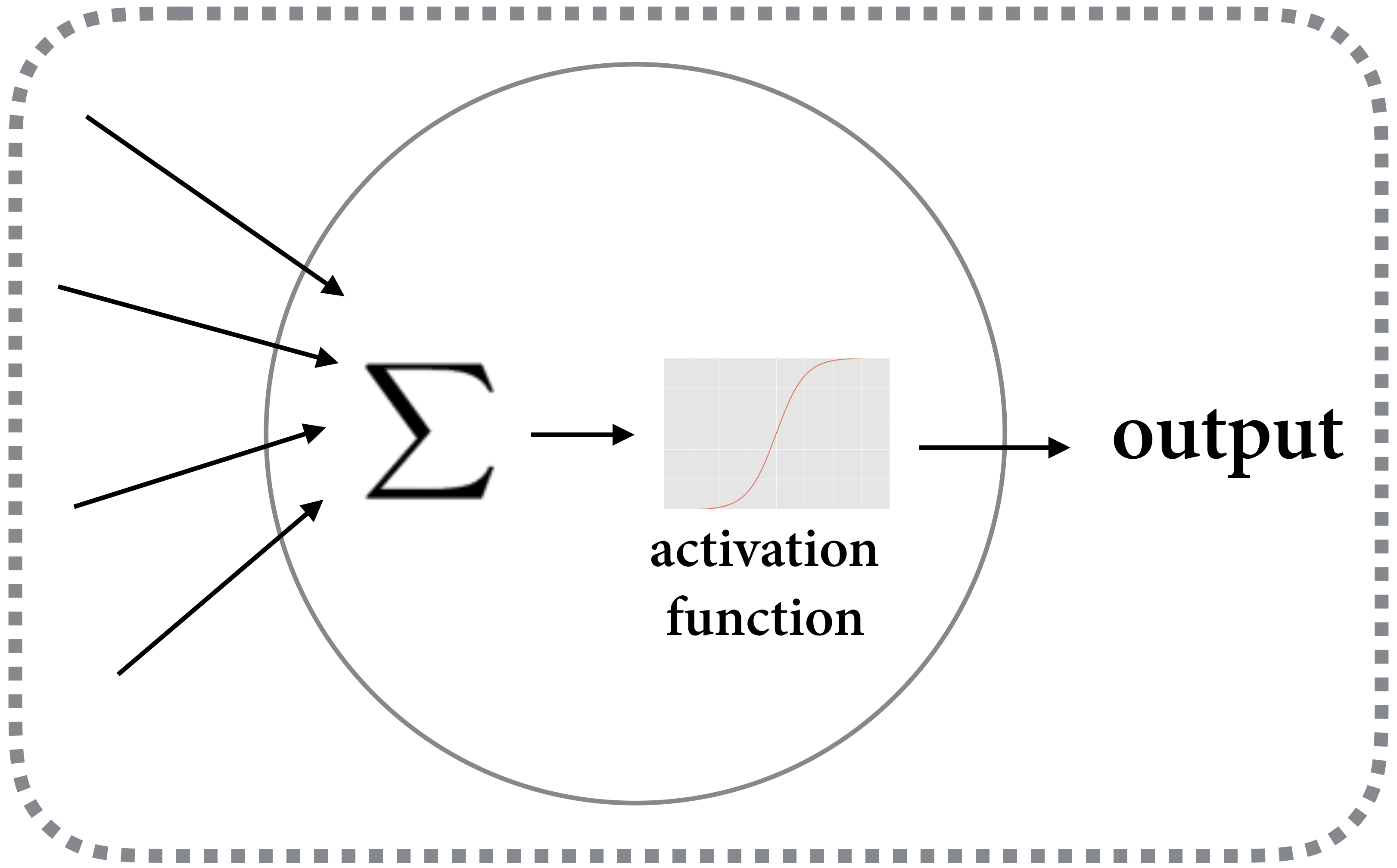
- 羅吉斯迴歸 (Logistic Regression)



神經網路 (Neural Netwok)

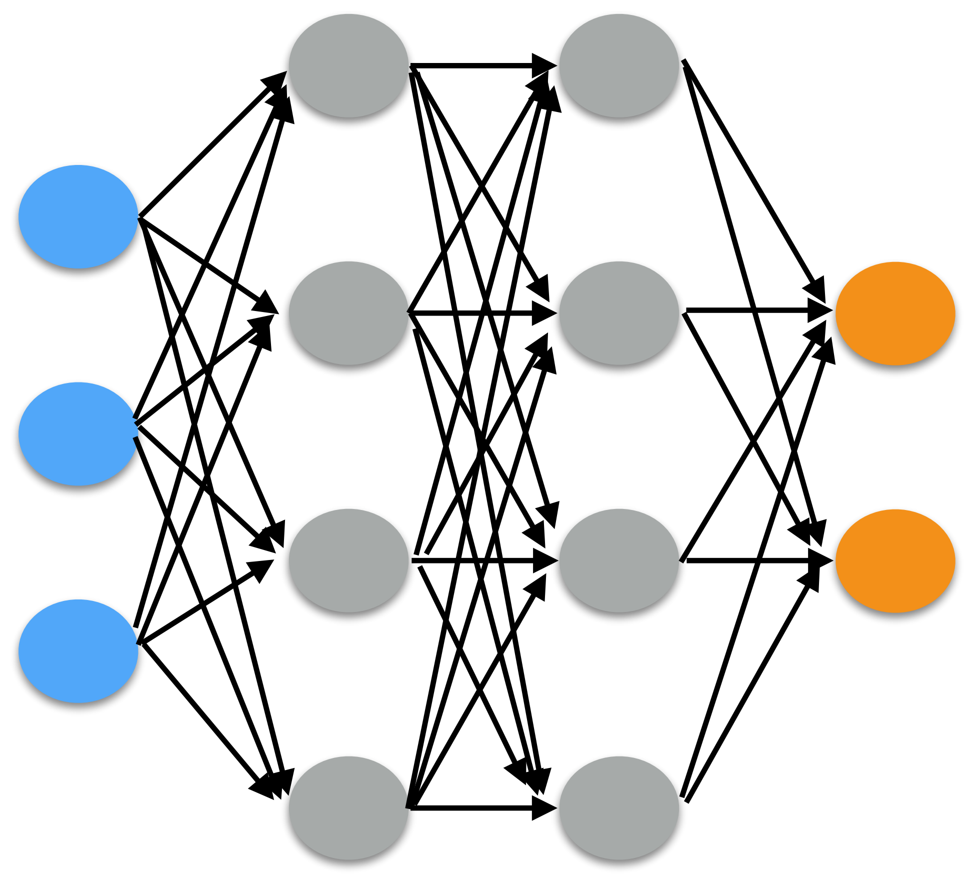


Perceptron/Neuron

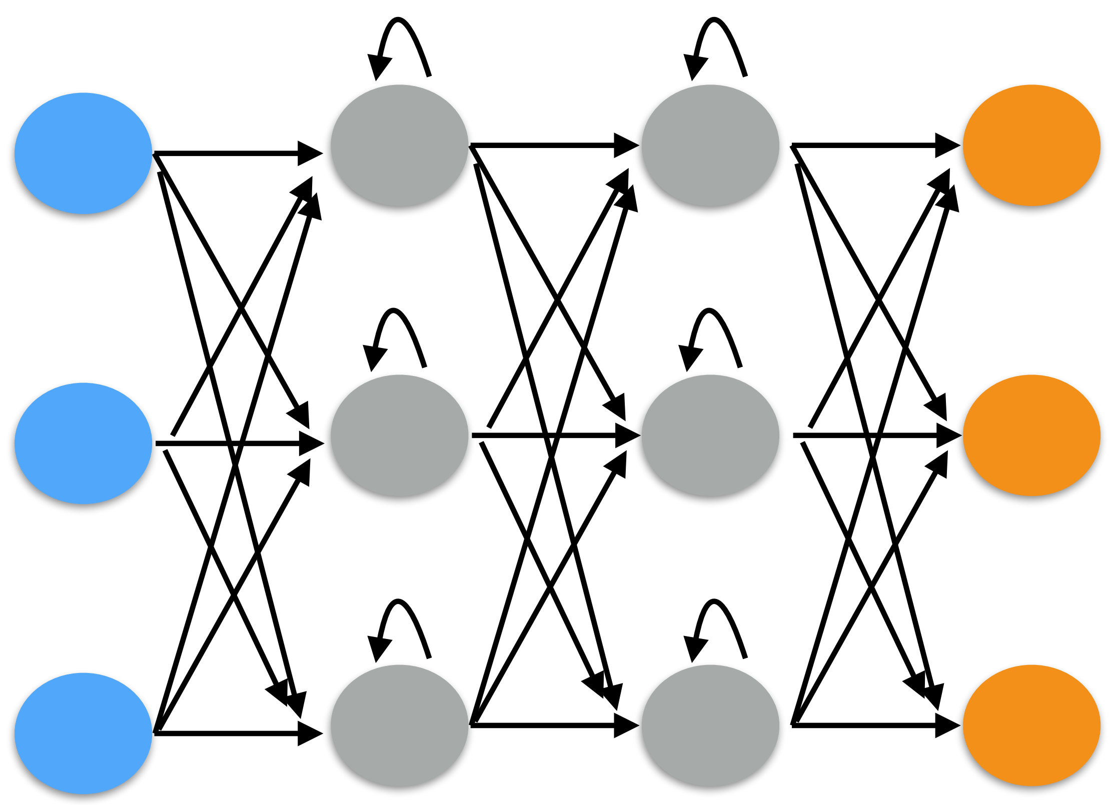


神經網路架構

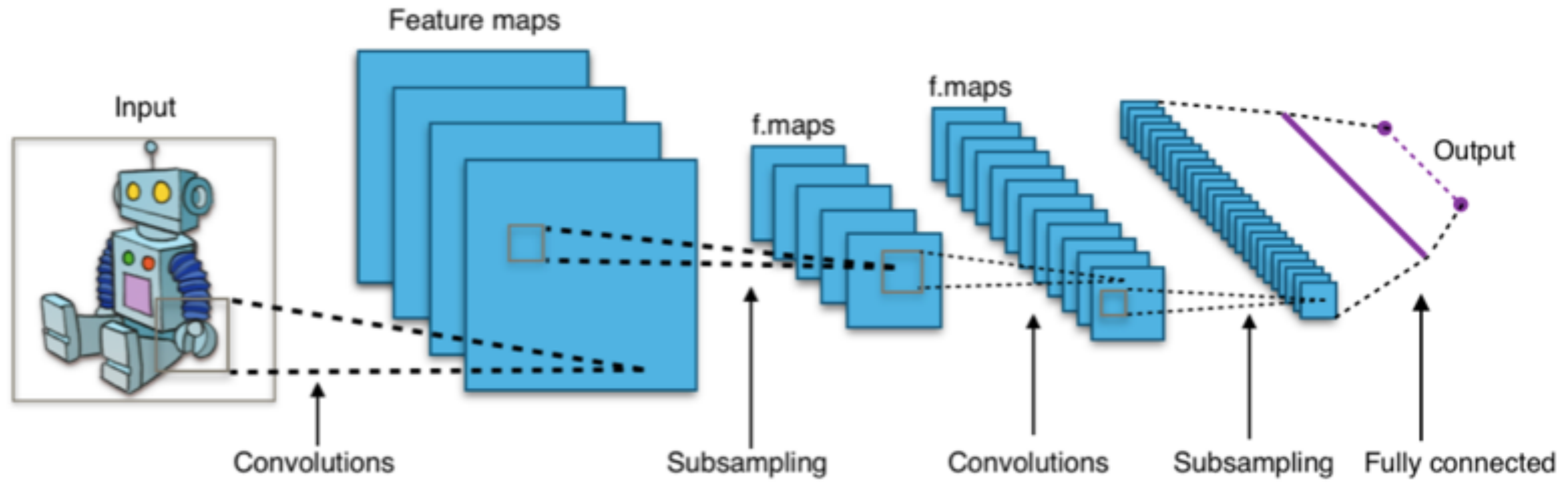
MLP



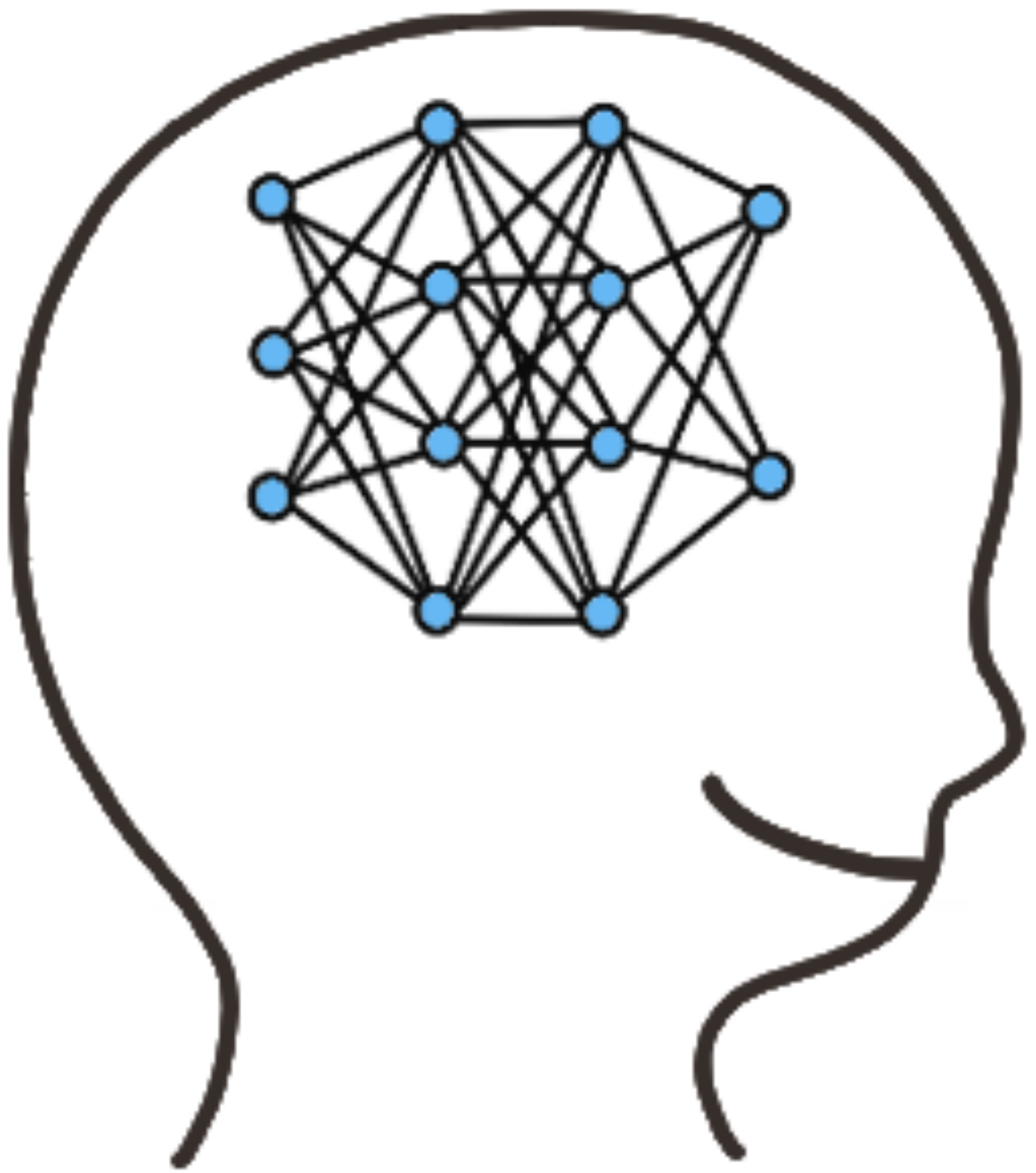
RNN



CNN



(Wikimedia Commons)



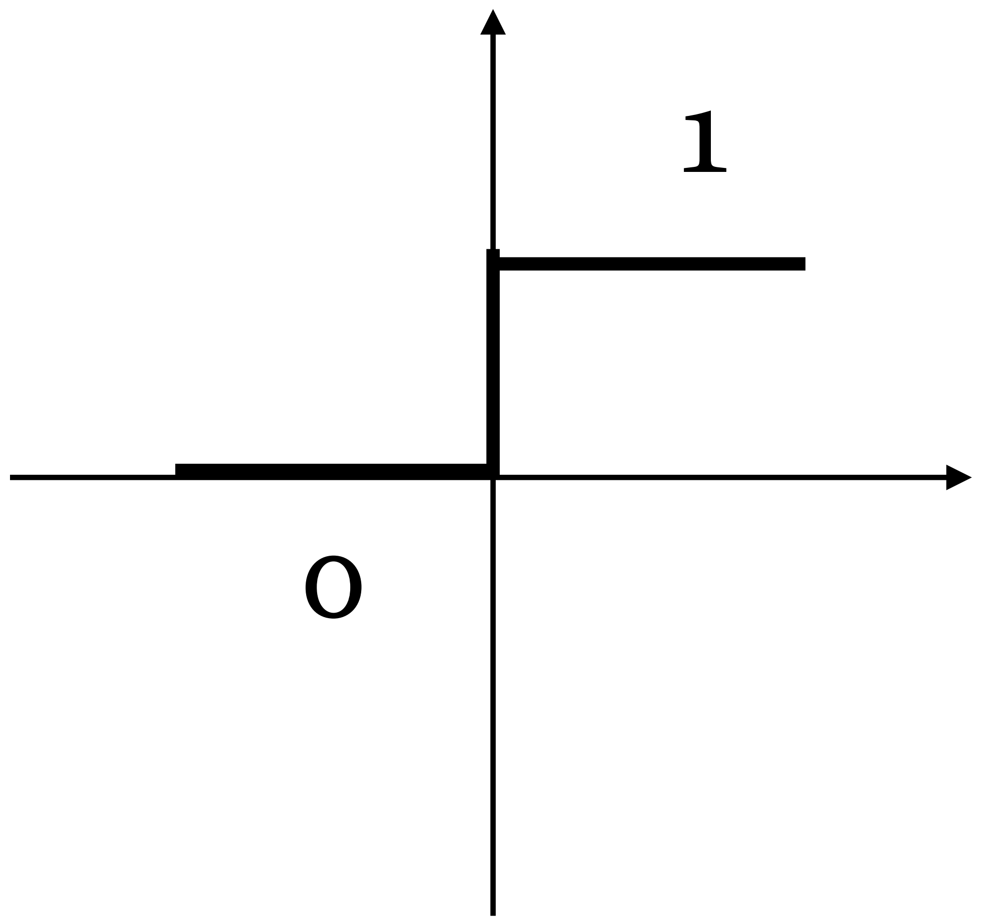
激勵函數

Activation Function

Activation Funciton

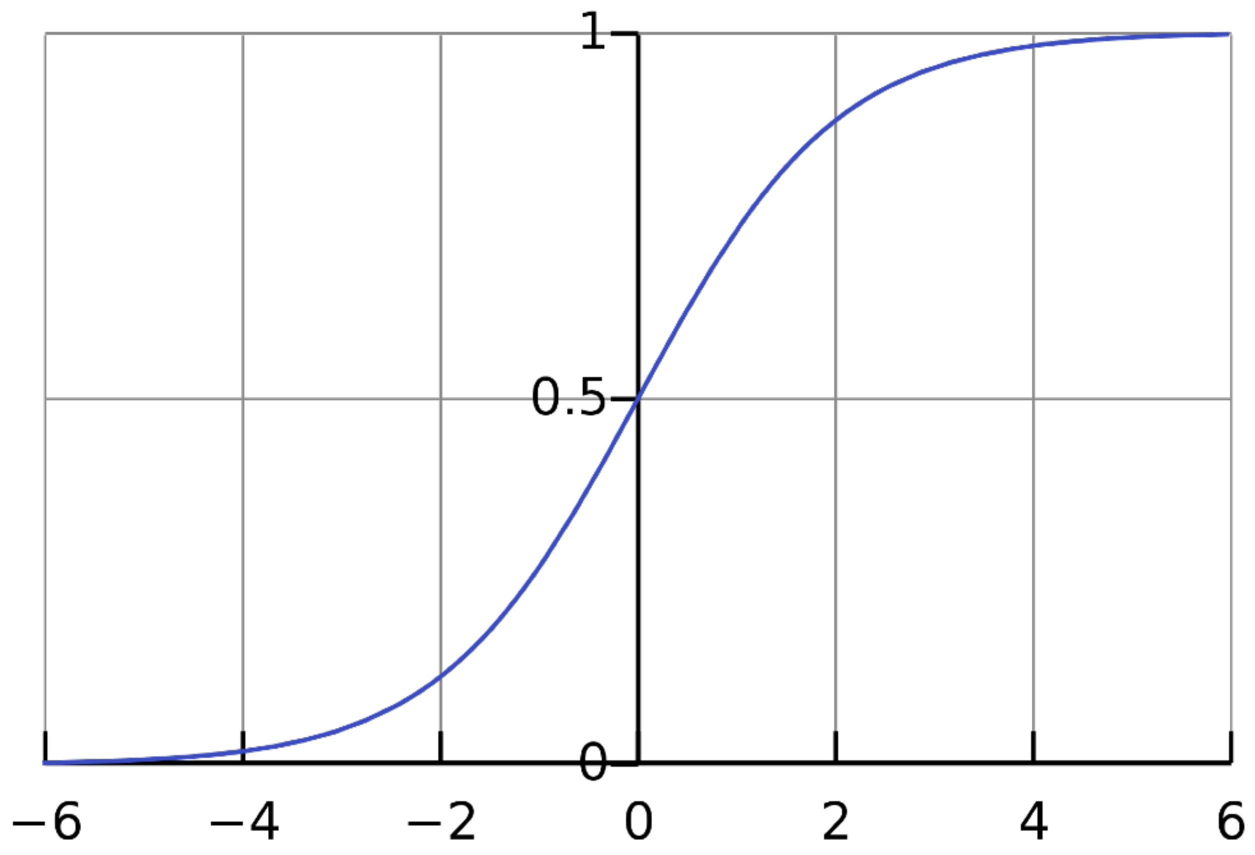
- 激勵/活化函數

Step Function



- 輸出為0/1

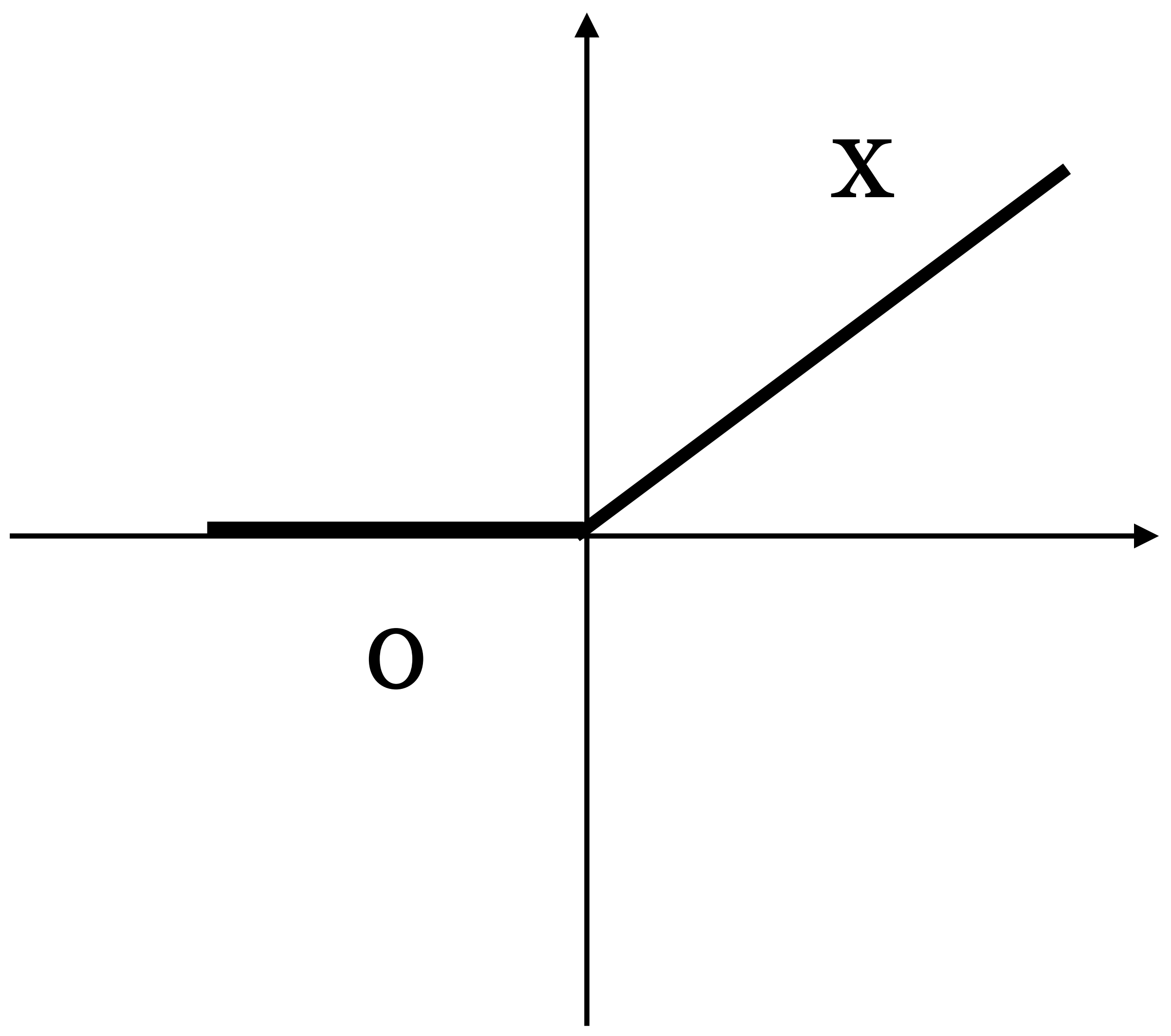
Sigmoid/Logistic Function



- 輸出介於(0,1)
- 可輸出連續型數值(機率)

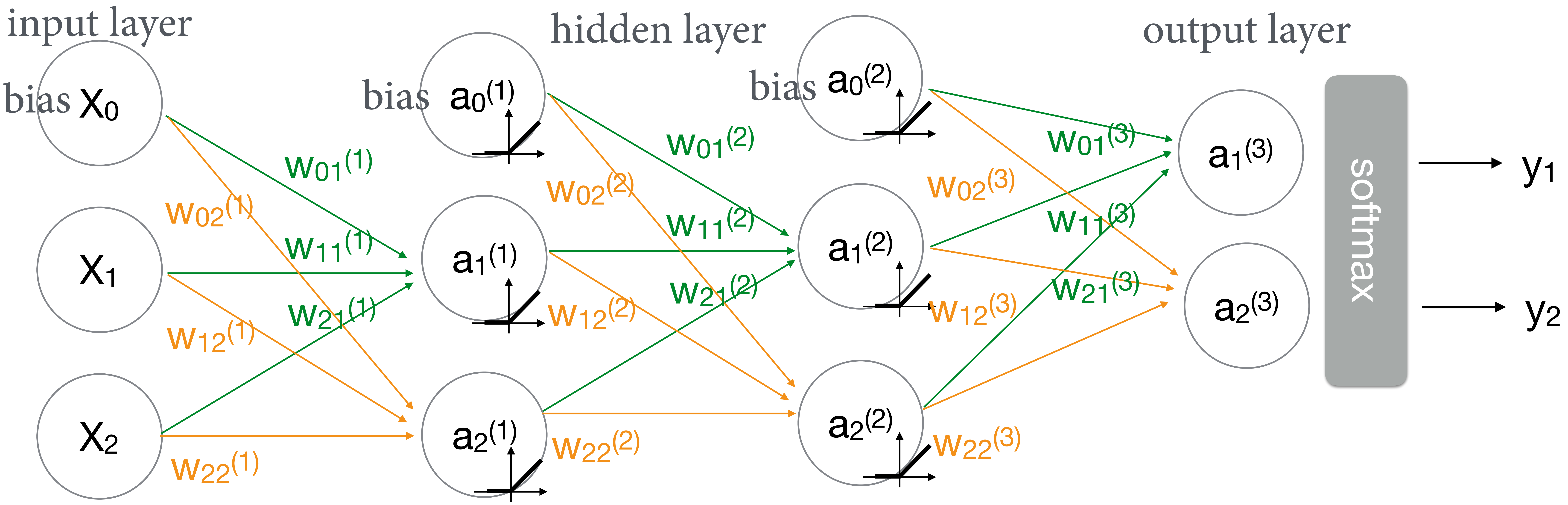
Activation Funciton

- ReLU (Rectified Linear Unit)



- 相較於Logistic Function 訓練時更容易收斂(輸出值較大)
- 目前深度學習較常用的activation function

Propagation Forward



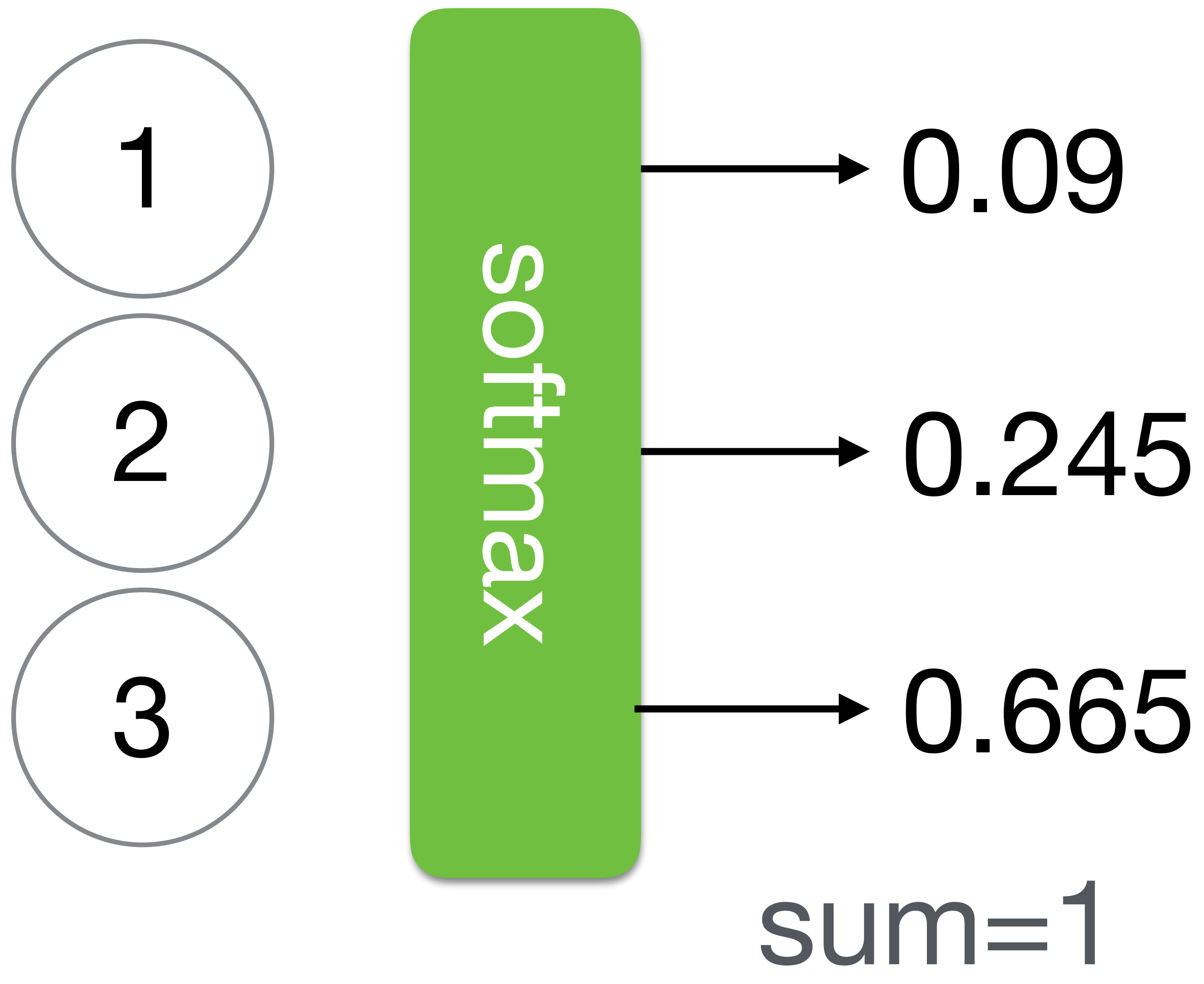
$$a_n^{(i)} = w_{0n}^{(i)} \cdot a_0^{(i-1)} + w_{1n}^{(i)} \cdot a_1^{(i-1)} + w_{2n}^{(i)} \cdot a_2^{(i-1)}$$

Softmax

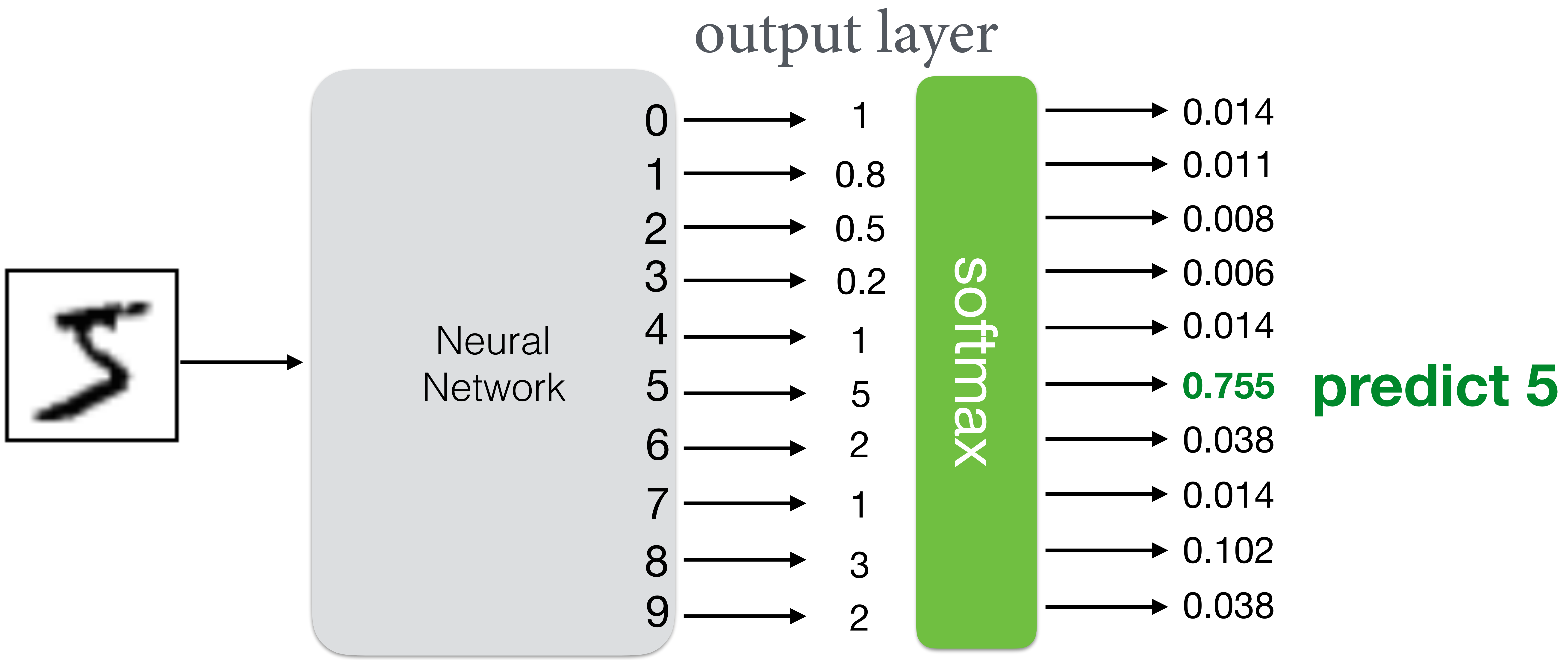
- Softmax函數可以把數值矩陣轉換成介於[0, 1]之間的機率值，且相加為1

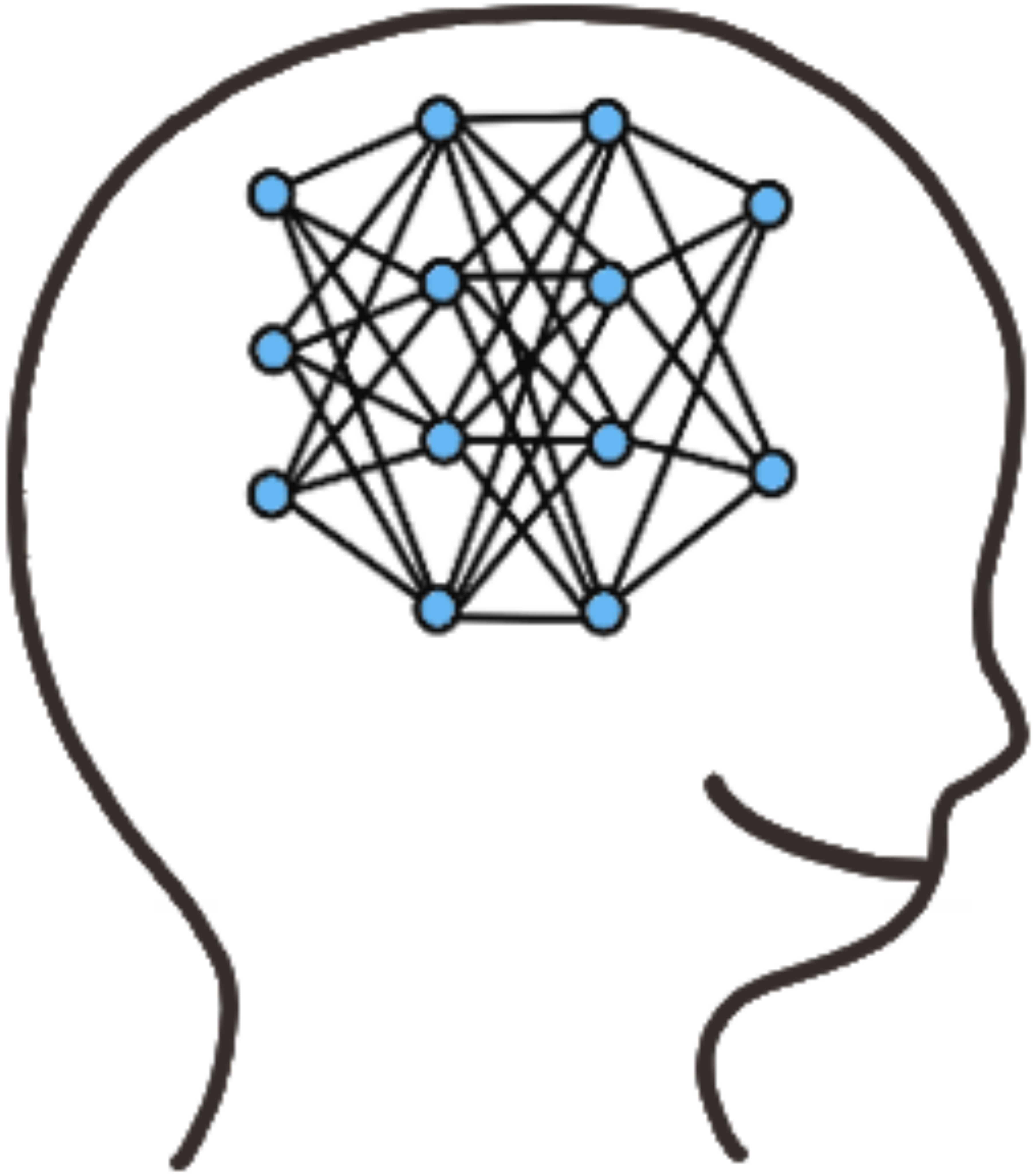
$$\mathrm{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

e.g.



多類別分類 with Softmax





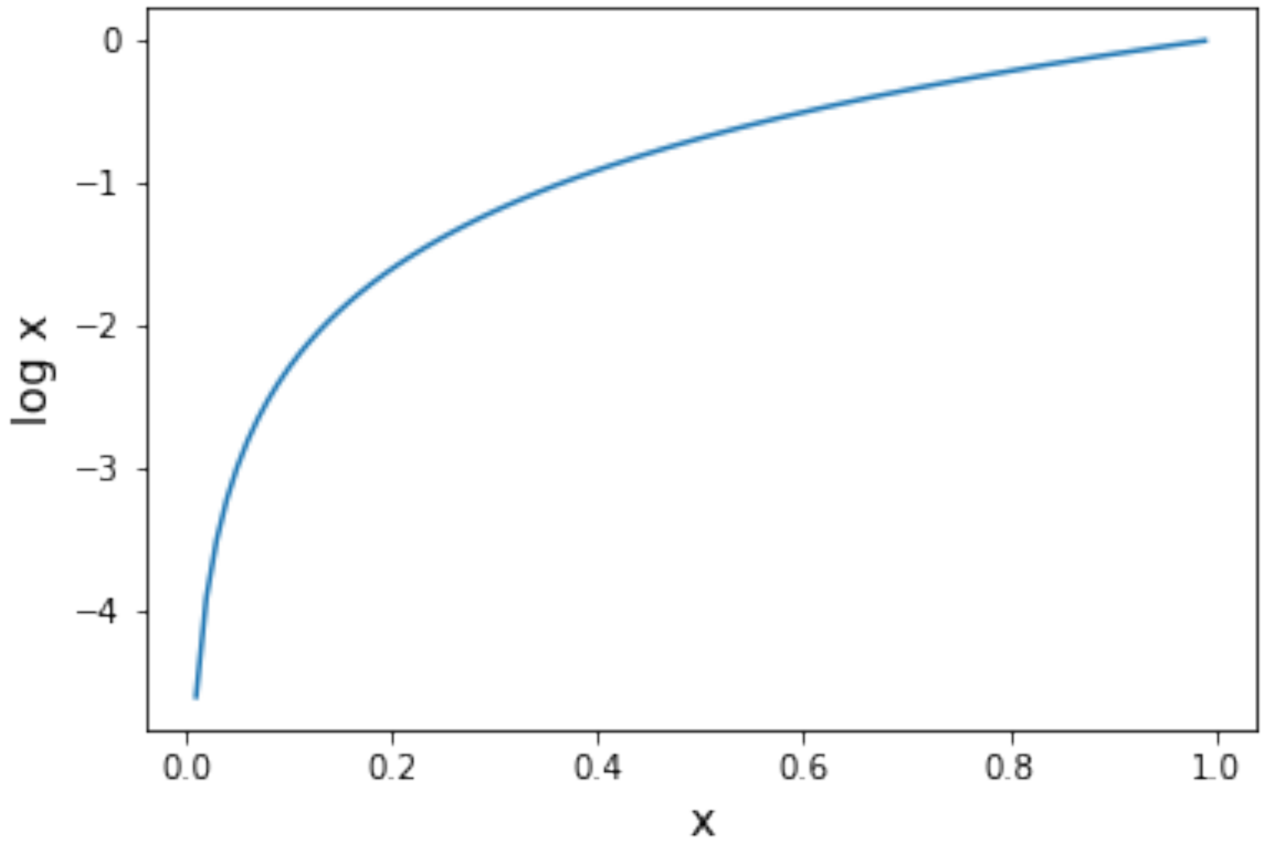
反向傳播演算法
Backpropagation

損失函數 (Loss Function)

- MSE (Mean Squared Error)
- 交叉熵誤差 (Cross Entropy Error)
  - $-\sum_m y_m \log \hat{y}_k$
  - 因為結果是0/1 (one-hot)，所以只會計算到真實值y = 1的Error

Notes

- **代價函數(Cost Function) 和損失函數(Loss Function) 意義相當接近，一般會把單一點實際和預測值的差稱為Loss Function，所有資料誤差加總或是加上正規化稱為Cost Function，另外也有人稱這兩者為Error Function。**



訓練權重方法

- 梯度下降 (Gradient Descent)

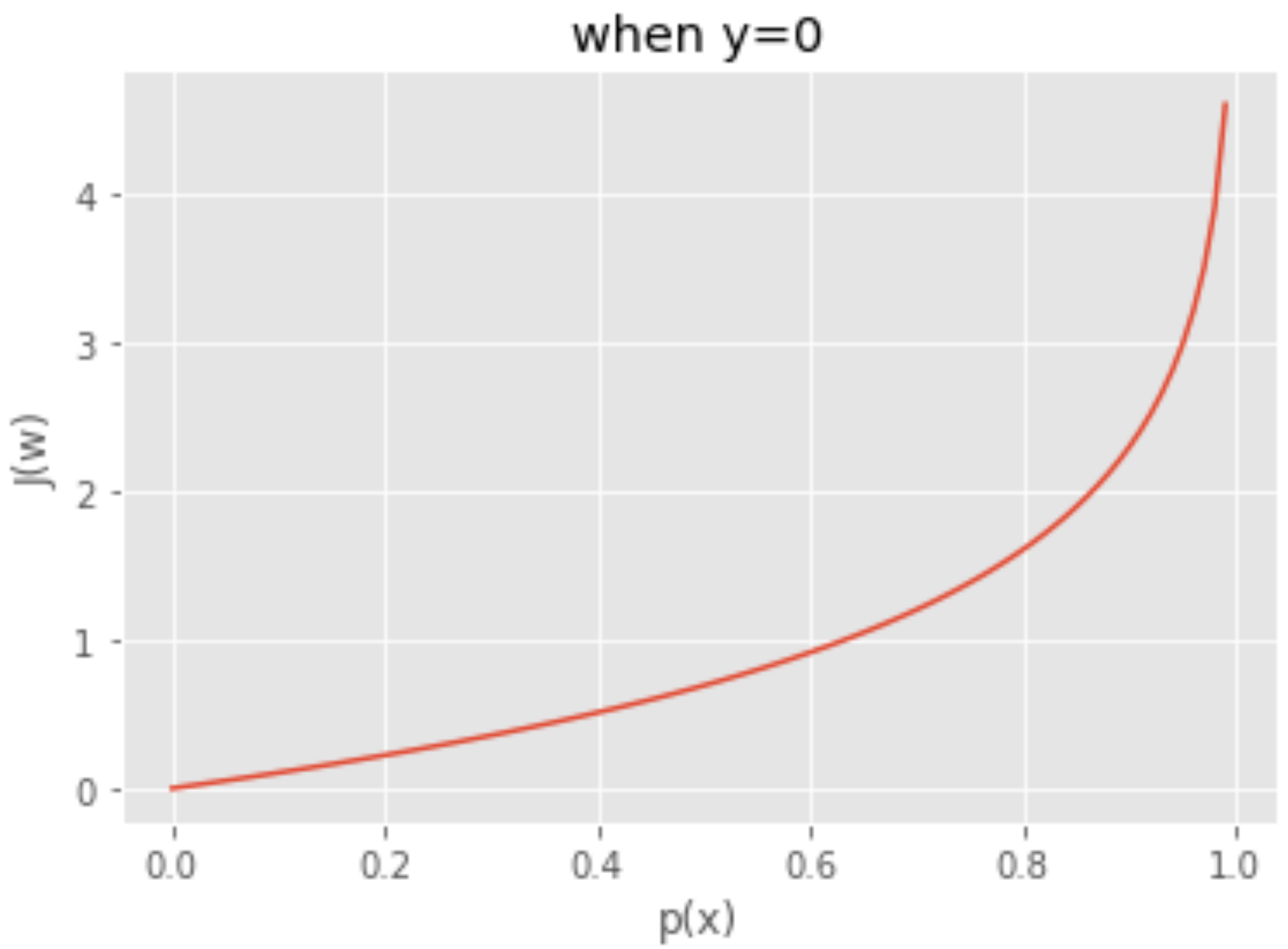
- **小批梯度下降**(Mini-batch Gradient Descent, MBGD)：介於批次梯度下降(Batch Gradient Descent, BGD)和隨機梯度下降(Stochastic Gradient Descent, SGD)之間，每次隨機選擇m筆資料計算。

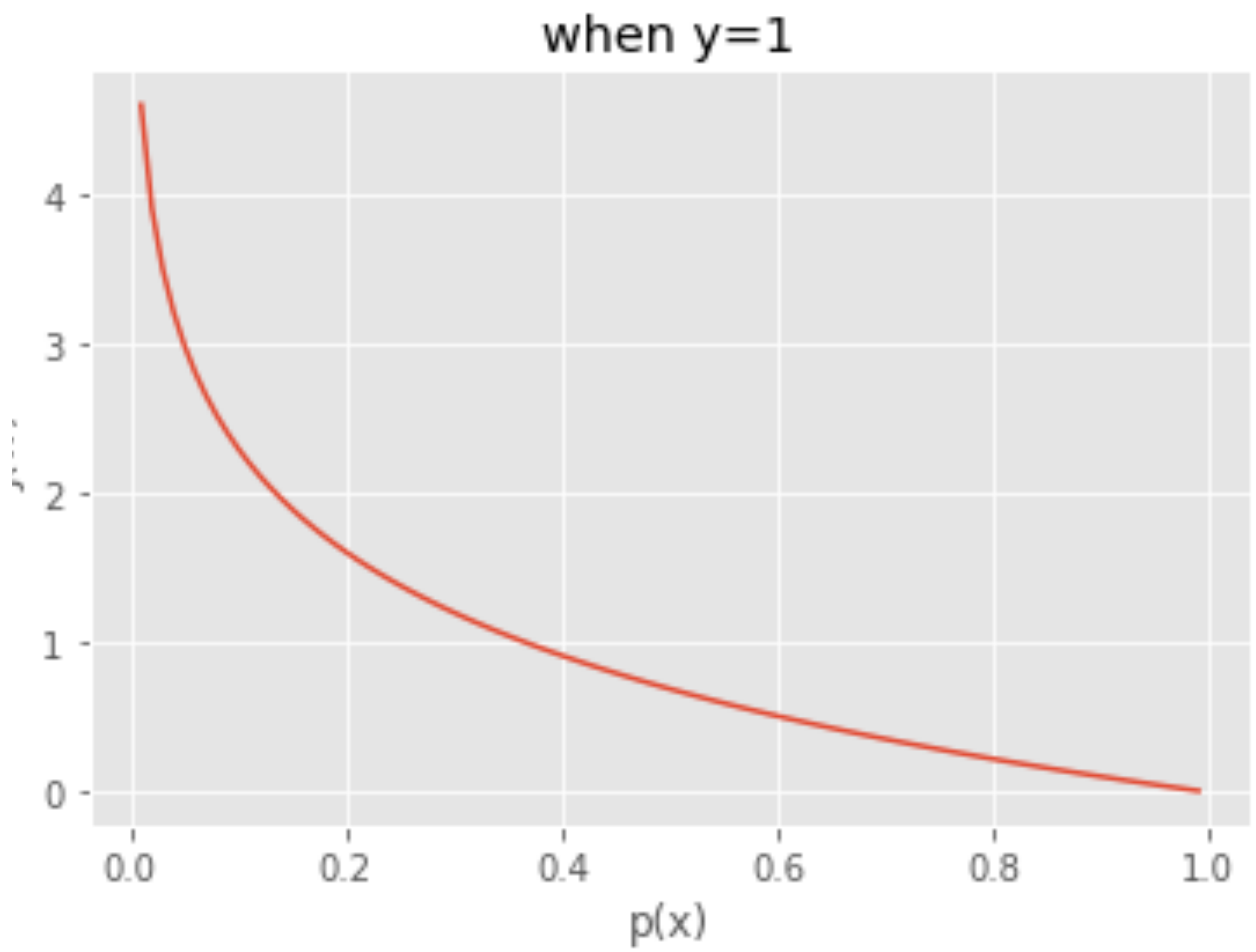
> Notes
>
> - **現在一般泛指的**SGD**也包含** MBGD。

Cost Function of Logistic Regression

$$J(w) = -\frac{1}{m}\left(\sum_{i=1}^{m} y^{(i)} \log p(x^{(i)}) + (1-y^{(i)}) \log(1-p(x^{(i)}))\right)$$

$$J(w) = -\log(1-p(x)) \qquad J(w) = -\log(p(x))$$





Cost Function & Gradient Descent

Logistic Regression: $J(w) = -\frac{1}{m}(\sum_{i=1}^{m} y^{(i)} \log p(x^{(i)}) + (1-y^{(i)})\log(1-p(x^{(i)})))$

Neural Network: $J(w) = -\frac{1}{m}(\sum_{i=1}^{m}\sum_{j=1}^{k} y_j^{(i)} \log \hat{y}_j^{(i)} + (1-y_j^{(i)})\log(1-\hat{y}_j^{(i)}))$

- 計算 J(w)的偏微分： $\frac{\partial}{\partial w} J(w)$
- 更新權重： $w_n = w_n - \alpha \cdot \frac{\partial}{\partial w} J(w)$

Notes

- Logistic Regression和Neural Network 通常都是採用Cross Entropy作為Loss Function

反向傳播法 (Backpropagation)

- 計算與hidden layer相連接的weights
- Steps:

1. 隨機初始weights
2. Propagation forward
3. 計算Loss/Cost
4. Backpropagation(1) - update the weights from *output layer* to *hidden layer*
5. Backpropagation(2) - update the weights from *hidden layer* to *input layer*

Notes

- **若神經網路中的weights初始值皆設為0(或相同值)，經Propagation後所有neuron的值都還會是0(或相同值)，導致無法正常訓練weights。**

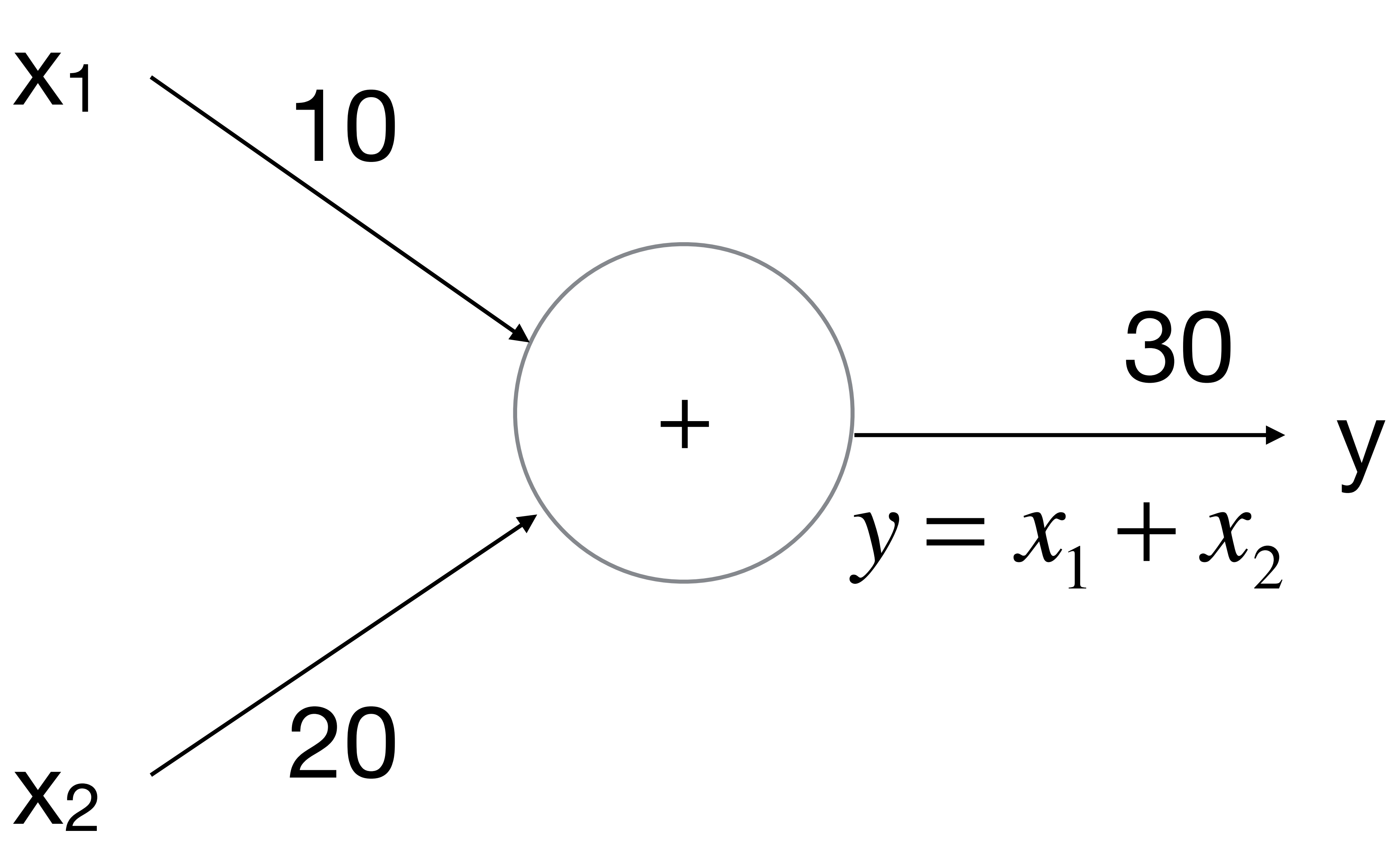
反向傳播法 (Backpropagation)

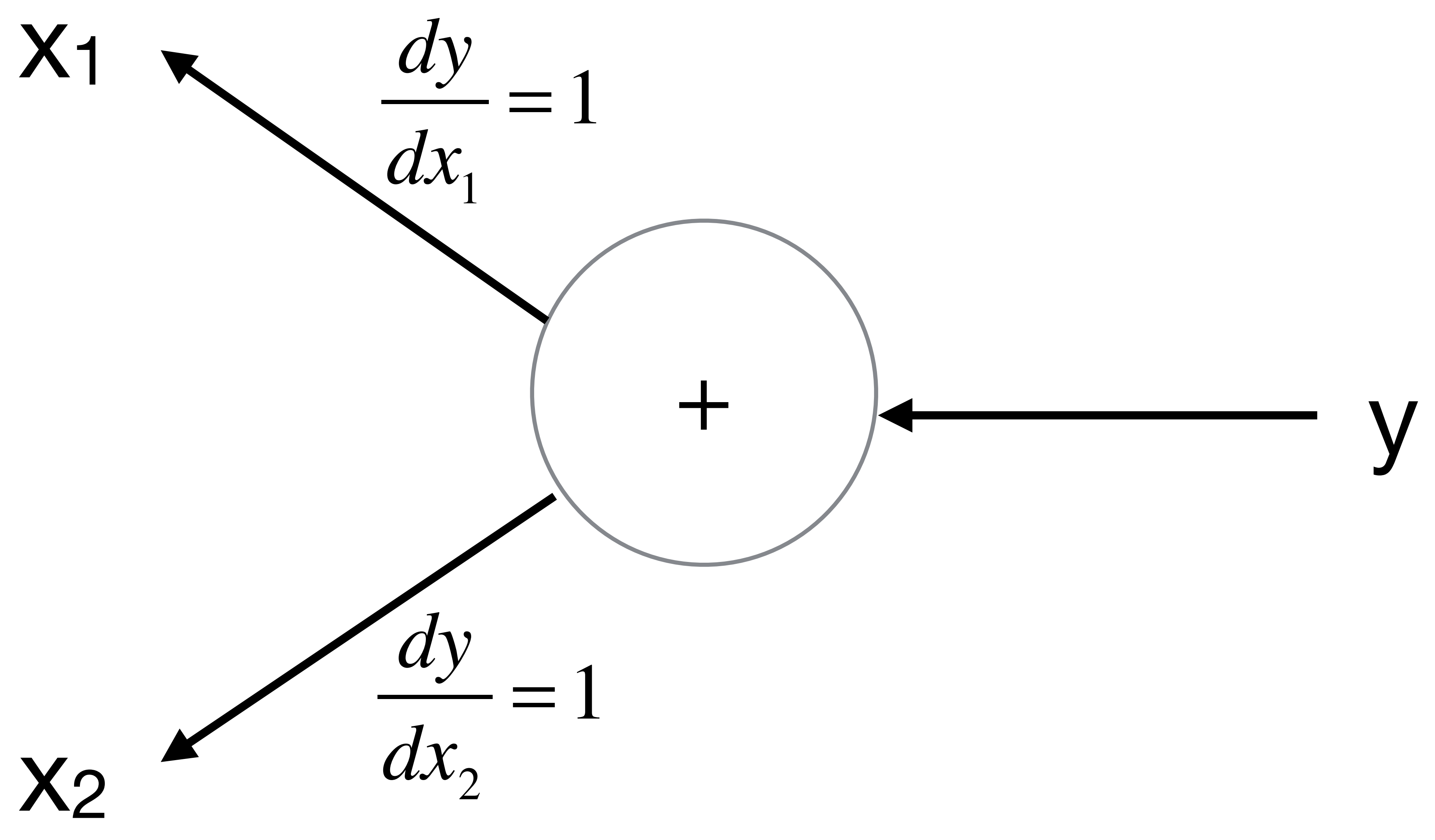
- Rumelhart et al. (1986)
- 搭配梯度下降的優化計算方法，至今仍是訓練深度學習模型最常用的方法
- 利用計算圖(Computational Graph)來理解

Notes

- **若對Backpropagation 完整數學及演算法推導過程有興趣，可參考References。**

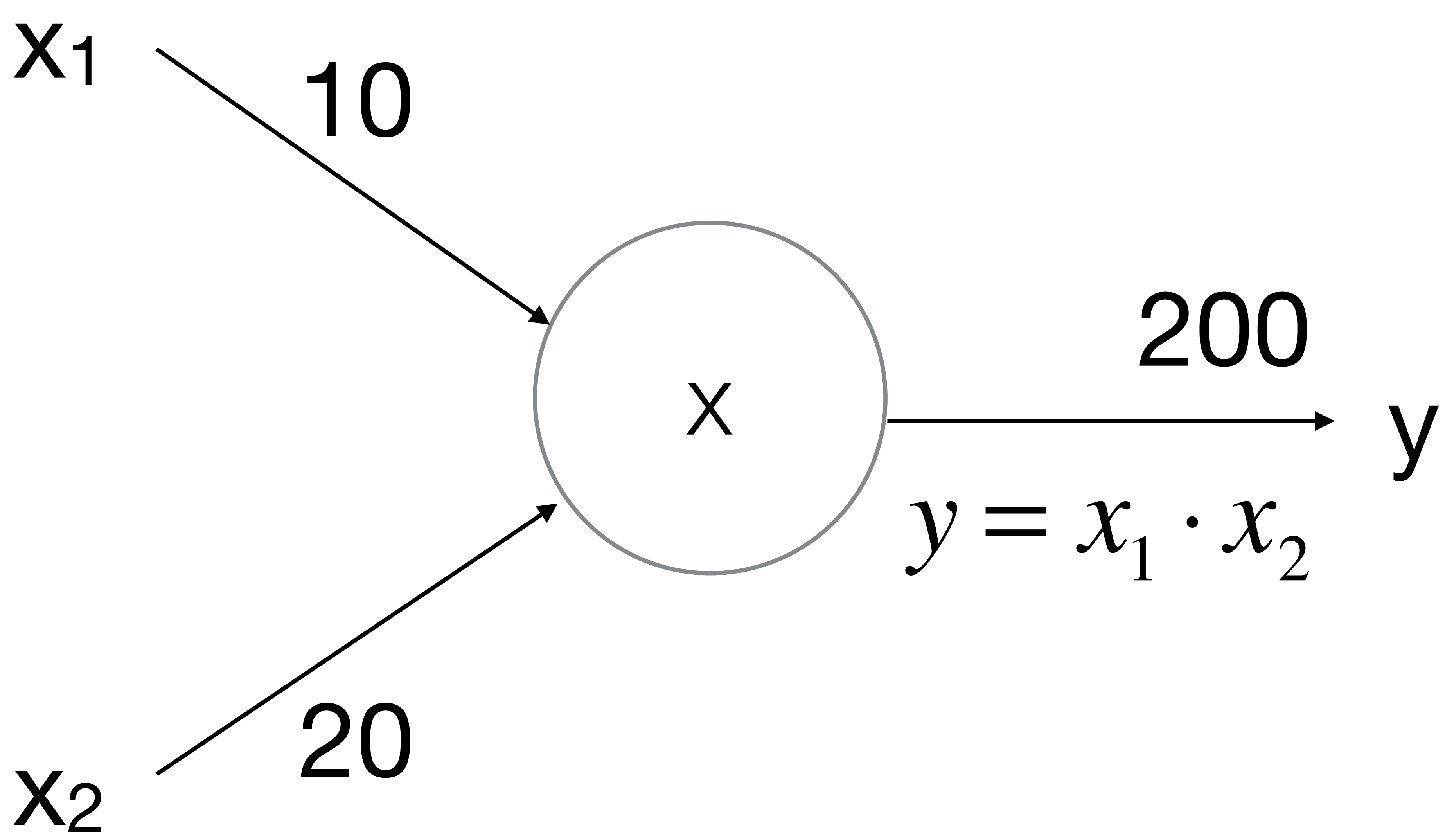
計算圖 - 加法

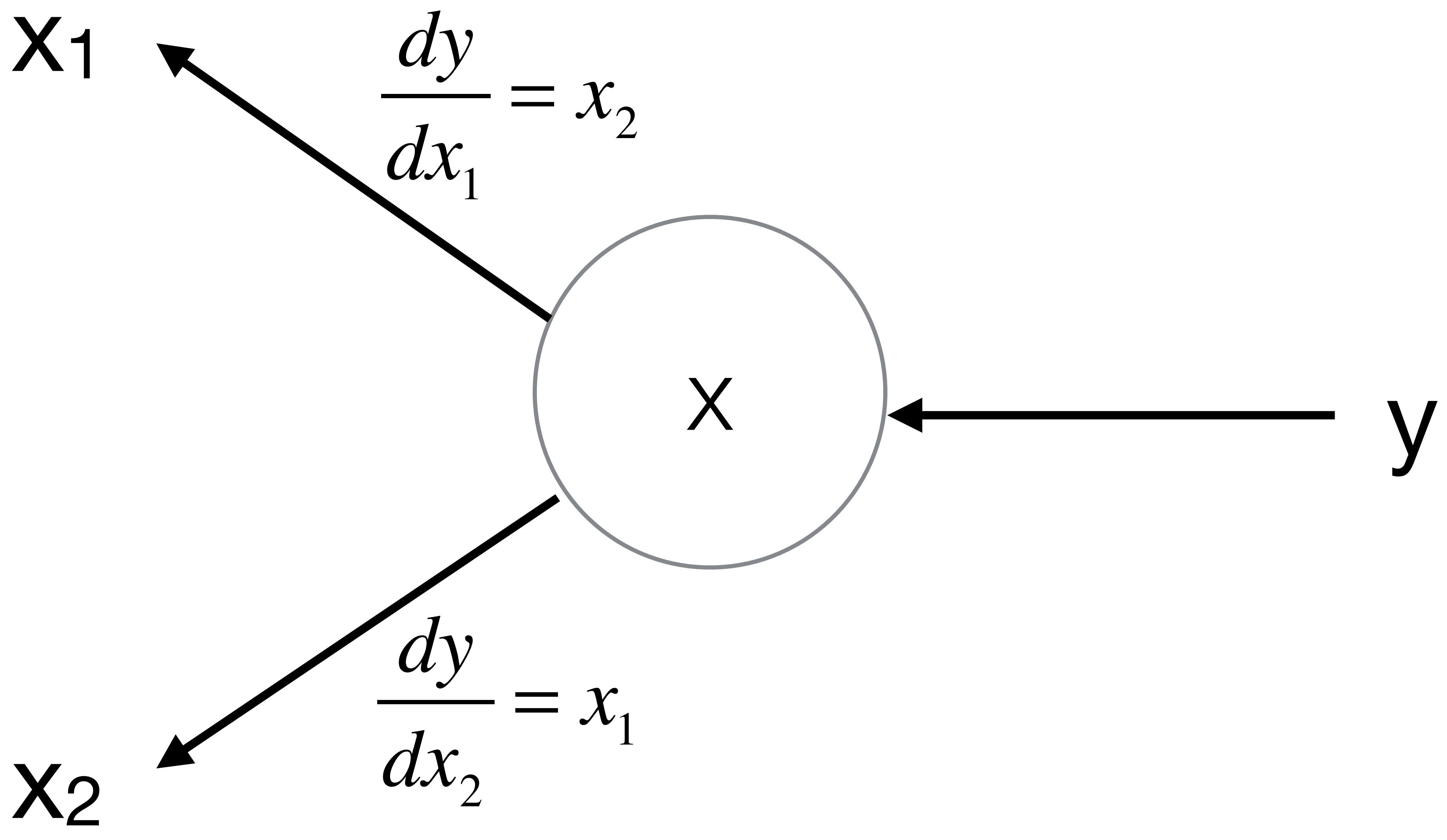




- 1代表x_1或x_2增加1，y就會增加1

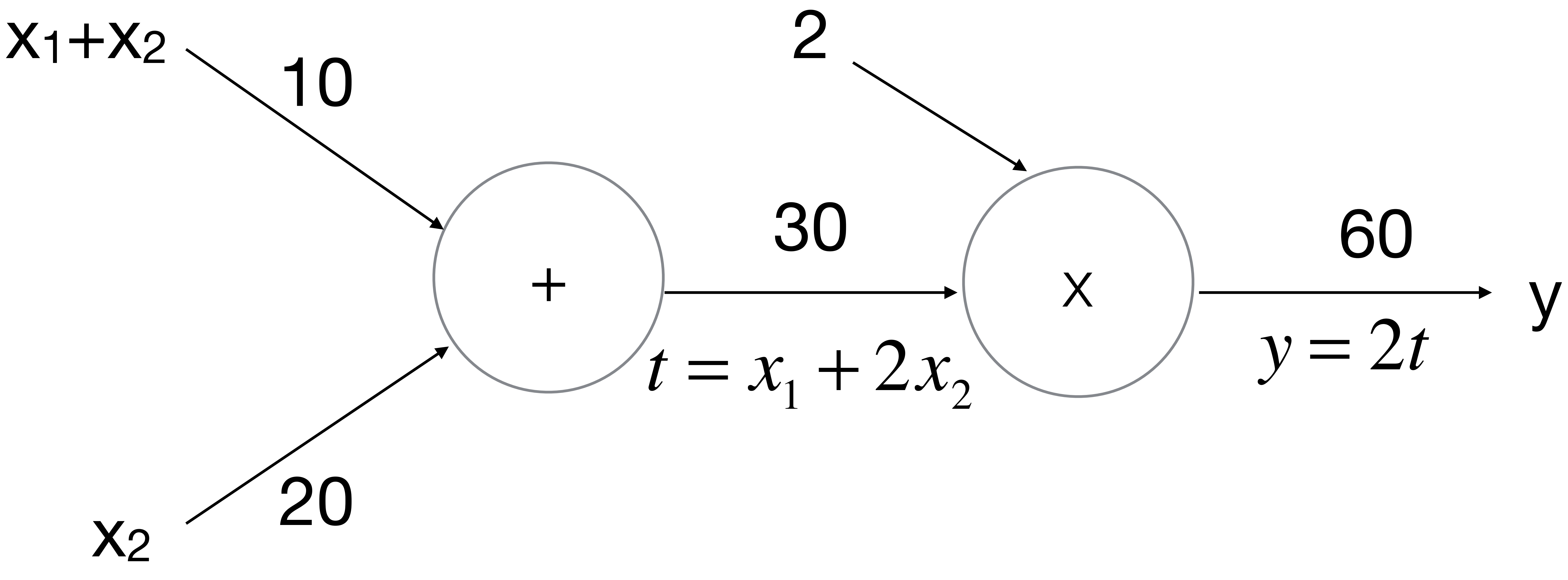
計算圖 - 乘法





- x_1增加1，y就會增加x_2
- x_2增加1，y就會增加x_1

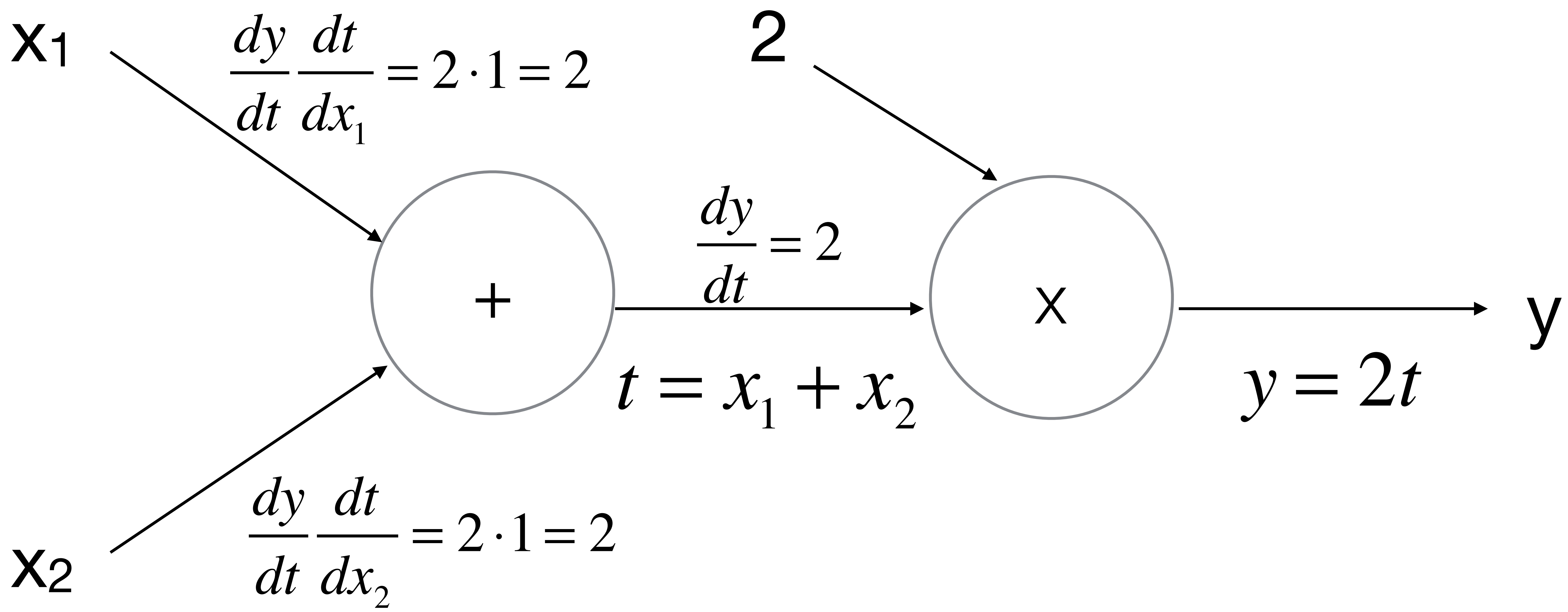
計算圖 - 多層&連鎖律(Chain Rule)



Notes

- 連鎖律(Chain Rule)
  - $\frac{dy}{dx} = \frac{dy}{dt}\frac{dt}{dx}$

計算圖 - 多層&連鎖律(Chain Rule)



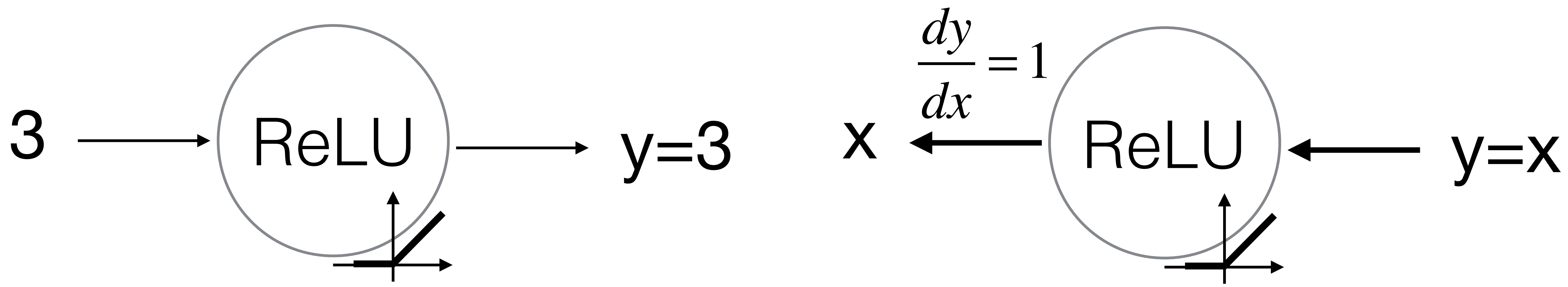
Notes

- 透過Backpropagation可以計算出neuron中的值對output y值的影響。

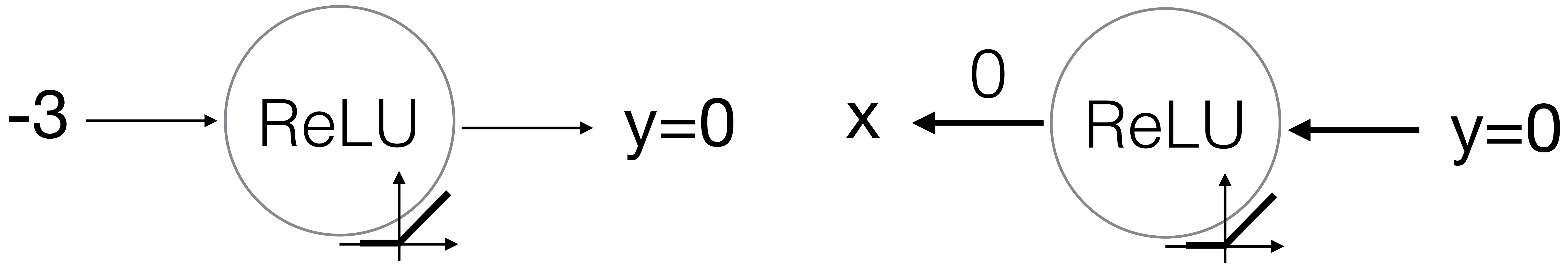
- x_1增加1，y就會增加2
- x_2增加1，y就會增加2

計算圖 - ReLU

when x>0



when x≤0

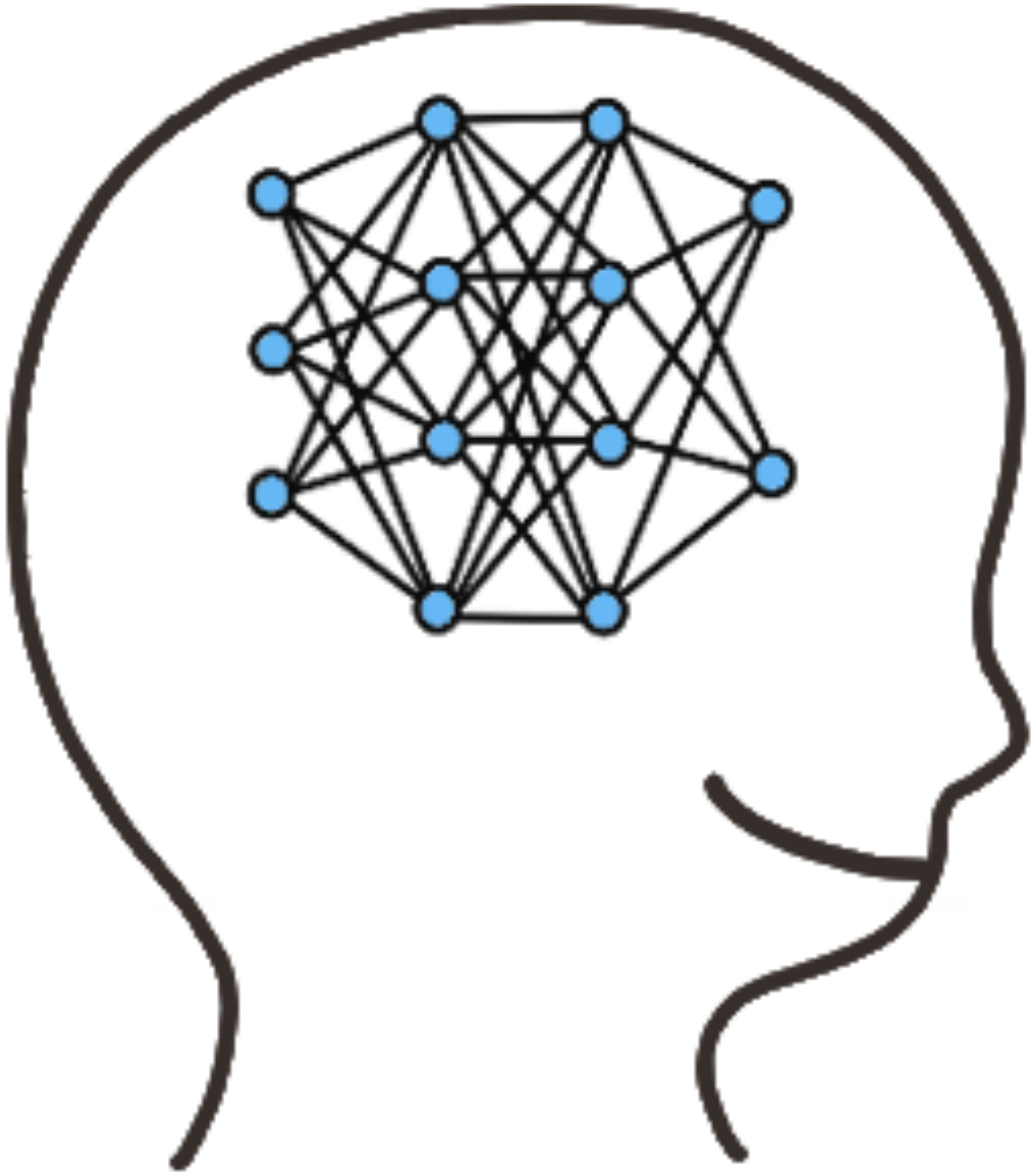


Notes

- 當neuron的值變成0時，Backpropagation也是0，等於這個neuron不運作了。

References

- Artificial Neural Networks: Mathematics of Backpropagation (Part 4)
    - http://briandolhansky.com/blog/2013/9/27/artificial-neural-networks-backpropagation-part-4
- The Backpropagation Algorithm
    - https://page.mi.fu-berlin.de/rojas/neural/chapter/K7.pdf

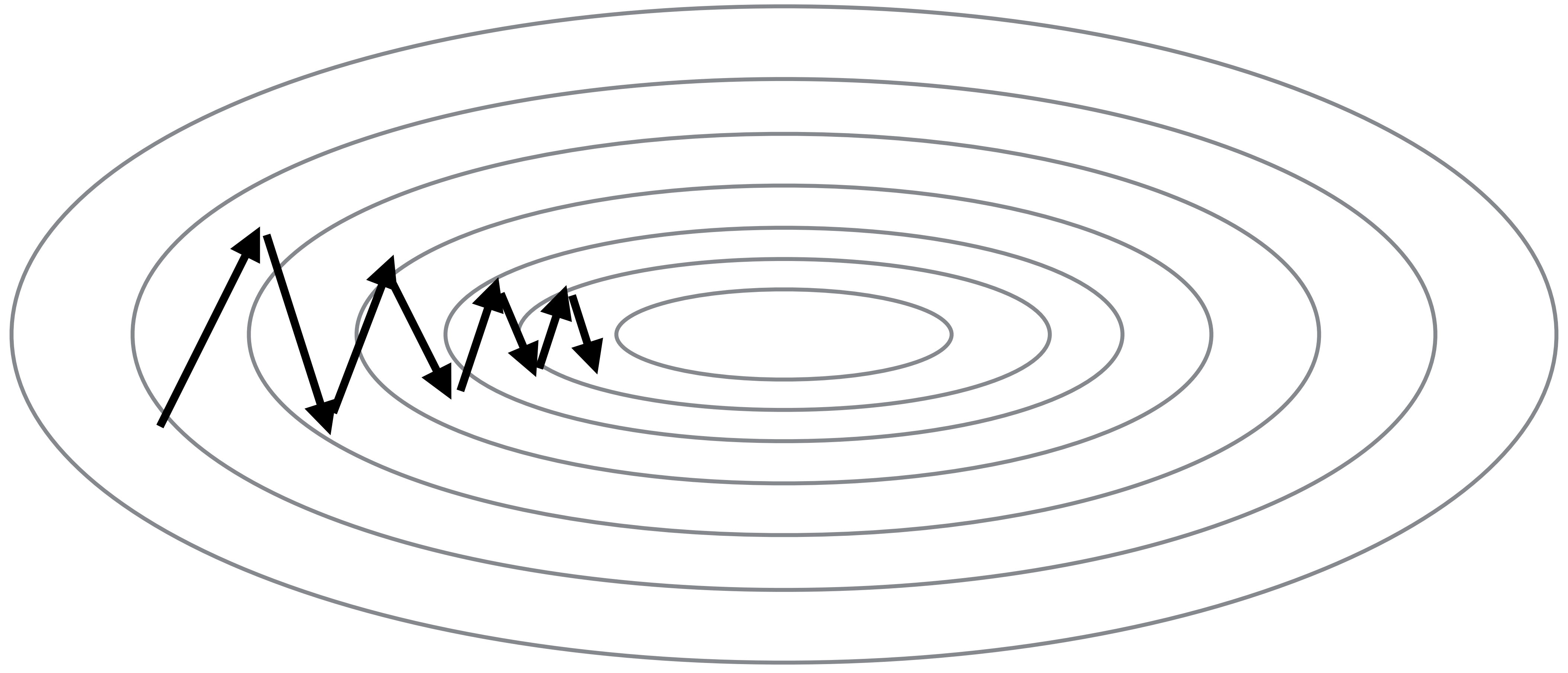


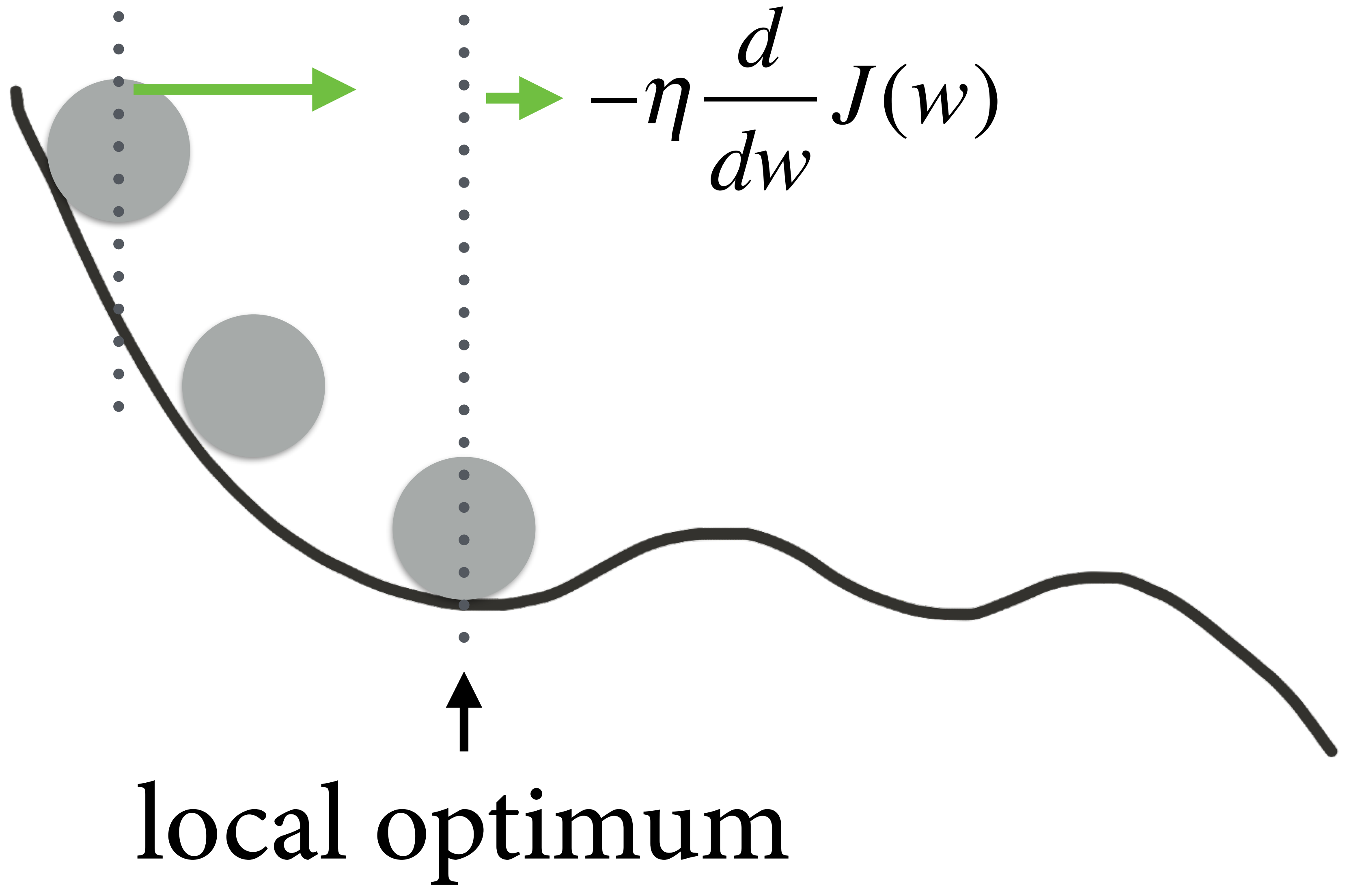
最佳化

Optimization

隨機梯度下降 (SGD)

- 前進方向晃動大，收斂速度較慢
- 步伐越來越小，可能卡在local optimum



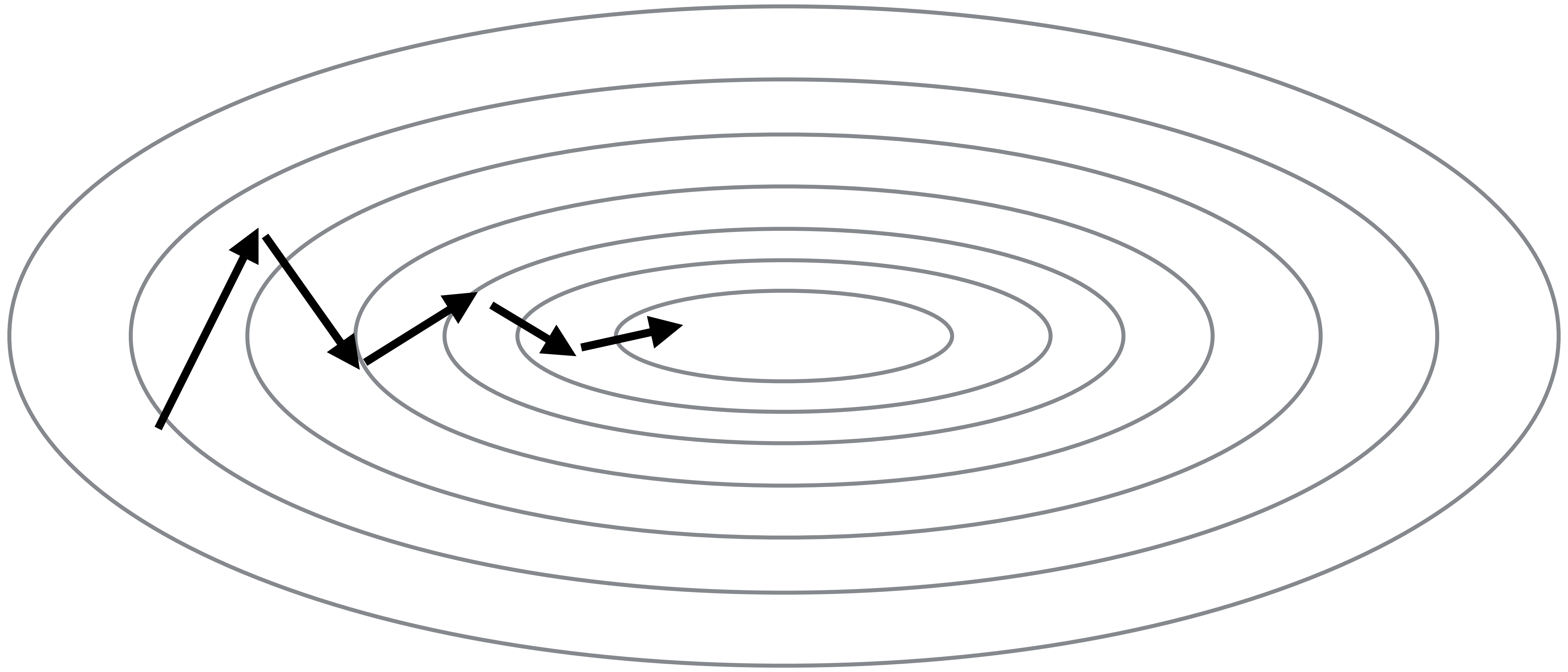


SGD with Momentum

- 物理原理：球越往低處滾，動量越大且越往穩定方向前進，可加速SGD收斂。
- $v = \boxed{rv} - \eta \frac{d}{dw} J(w)$

減速(阻力)=0.9 學習速率(eta)

$$w = w + v$$

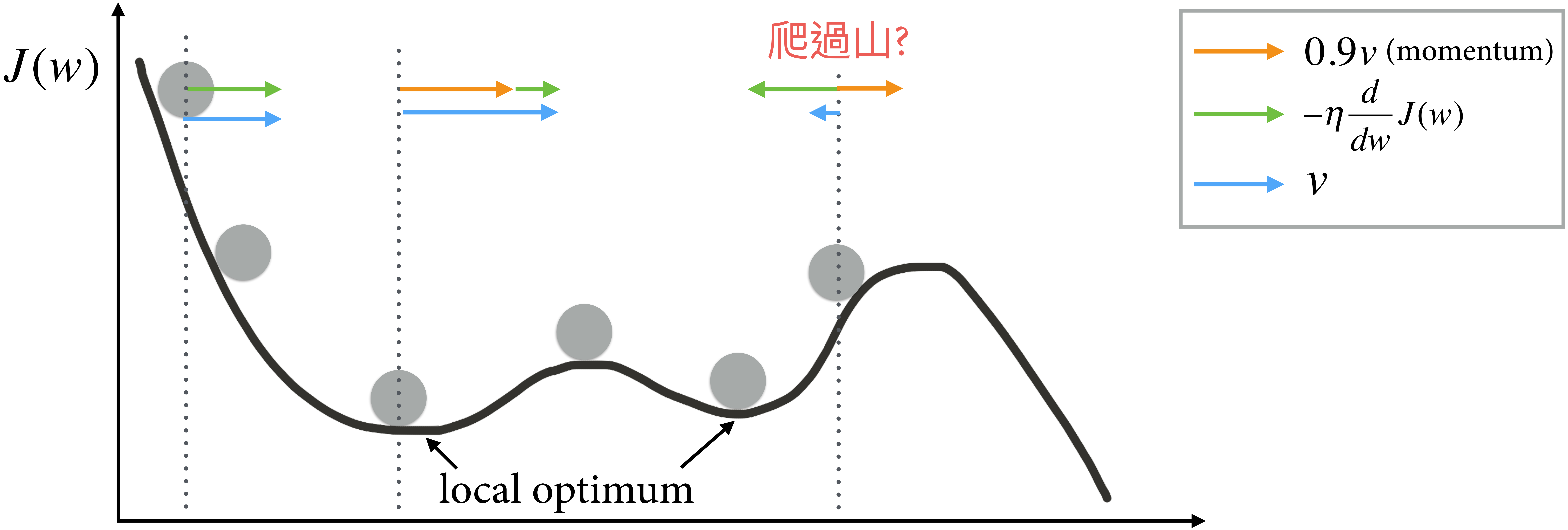


Notes

- 動量 (向量)

$$\vec{p} = m\vec{v}$$

SGD with Momentum



$w = w + v$

SGD with Keras

- keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=**False**)
    - **lr**: float >= 0. Learning rate.
    - **momentum**: float >= 0. Parameter updates momentum.
    - **decay**: float >= 0. Learning rate decay over each update.
    - **nesterov**: boolean. Whether to apply Nesterov momentum.

AdaGrad

- 不同特徵使用不同的學習速率，而**非定值**或**統一降低**的學習速率

$$h = h + \frac{d}{dx}J(w) \odot \frac{d}{dx}J(w)$$

J(w)微分值越大(坡越陡) h越大，學習速率越慢

$$w = w - \boxed{\eta \frac{1}{\sqrt{h+\varepsilon}}} \frac{d}{dw}J(w)$$

學習速率
不同w_n有不同的h

epsilon: 設定很小的值
避免根號內為o

AdaGrad with Keras

- keras.optimizers.Adagrad(lr=0.01, epsilon=1e-08, decay=0.0)
    - **lr**: float >= 0. Learning rate.
    - **epsilon**: float >= 0.
    - **decay**: float >= 0. Learning rate decay over each update.
- *It is recommended to leave the parameters of this optimizer at their default values.*

RMSProp

- 速率降低到最後可能變成0，所以用RMSProp改善此問題，建議值為設定保留過去0.9的h (遺忘掉0.1來更新)

$$h_t = 0.9h_{t-1} + 0.1\frac{d}{dx}J(w) \odot \frac{d}{dx}J(w)$$

$$w = w - \eta\frac{1}{\sqrt{h+\varepsilon}}\frac{d}{dw}J(w)$$

RMSProp

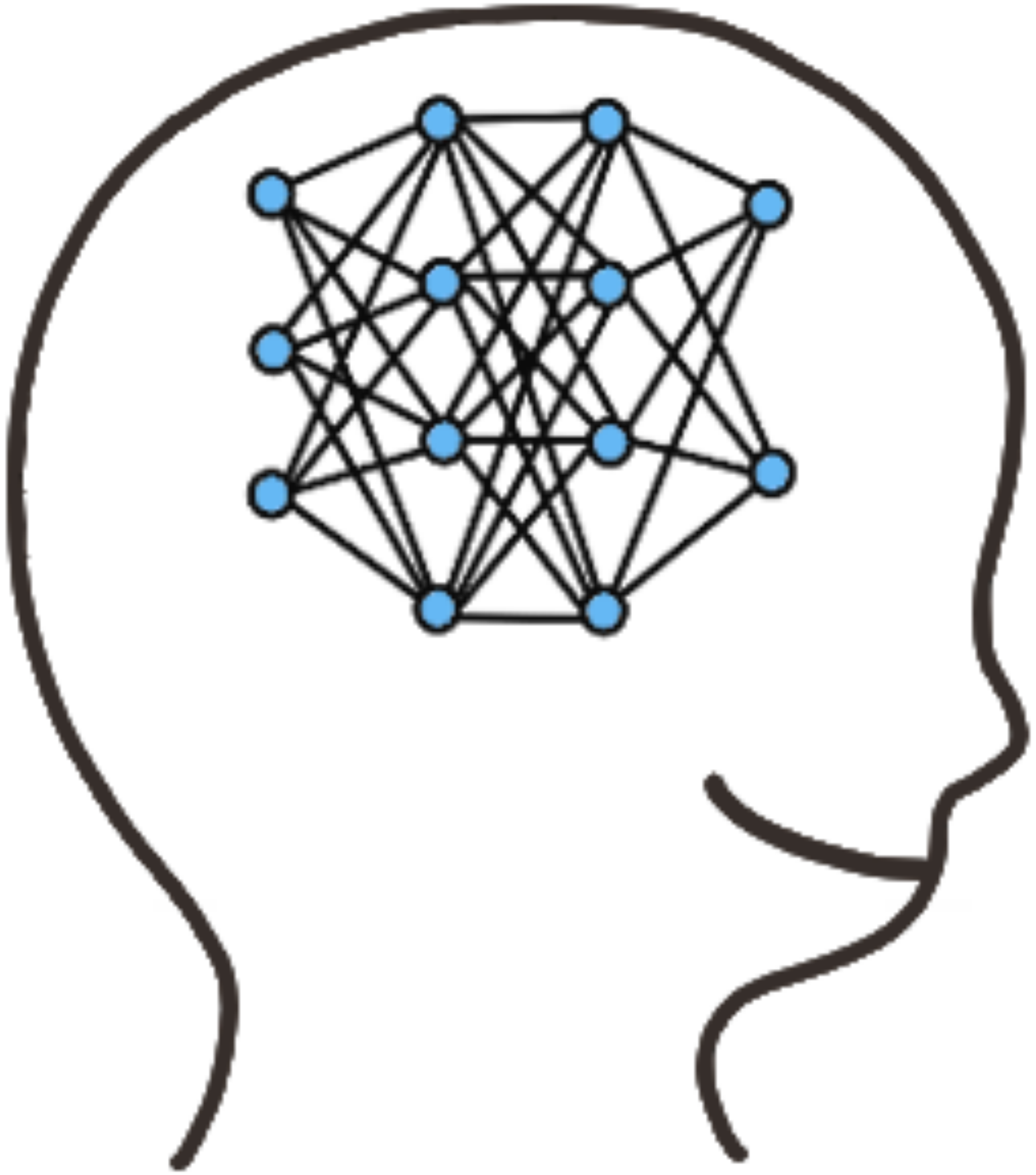
- keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
    - **lr**: float >= 0. Learning rate.
    - **rho**: float >= 0.
    - **epsilon**: float >= 0. Fuzz factor.
    - **decay**: float >= 0. Learning rate decay over each update.
- It is recommended to leave the parameters of this optimizer at their default values (except the **learning rate**, which can be freely tuned).
- This optimizer is usually a good choice for **recurrent neural networks (RNN)**.

Adam

- SGD with Momentum + AdaGrad
- 表現通常不錯
- keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
  - **lr**: float >= 0. Learning rate.
  - **beta_1**: float, 0 < beta < 1. Generally close to 1.
  - **beta_2**: float, 0 < beta < 1. Generally close to 1.
  - **epsilon**: float >= 0. Fuzz factor.
  - **decay**: float >= 0. Learning rate decay over each update.

Notes

- **論文原著:**〈ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION〉 https://arxiv.org/pdf/1412.6980.pdf



用Keras訓練第一個神經網路

Training your first neural network with Keras