# 梯度下降與反向傳播
# Gradient Descend and Back Propagation

2025/6/17

PRESENTED BY AI Foundation

CREATED FOR

INNOLUX

# 梯度下降與反向傳播
## Gradient Descend and Back Propagation

1 梯度下降

2 反向傳播

# 梯度下降


Media saying AI will take over the world
My Neural Network
Dog
AI will take over soon

# 何為「學習」？

$$Wx + b = y$$

$W = 2$, $b = 0.5$

y = 5

$x = 2$
$b = 0.5$

$W = 0.5$
$y = 1.5$

5 - 1.5 = 3.5

你的答案太小了

$W = 3$
$y = 6.5$

5 - 6.5 = -1.5

你的答案太大了

$W = 2$
$y = 4.5$

5 - 4.5 = 0.5

你的答案有點小

# 何為「學習」？

$$x = 2 \xrightarrow{W} \boxed{Wx + 0.5} \longrightarrow \hat{y}$$

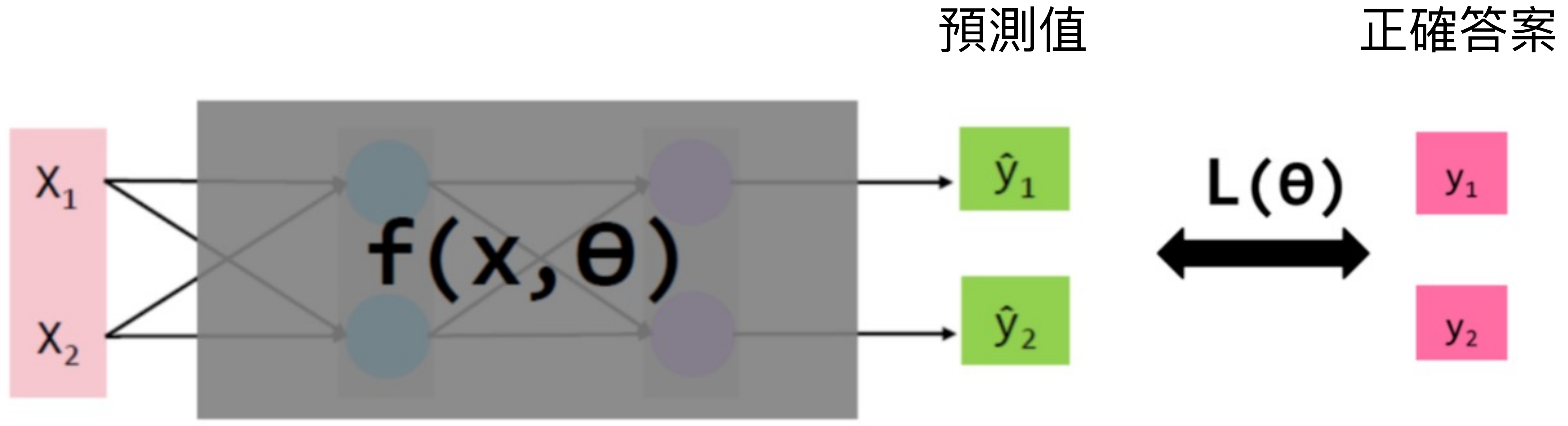
$\boxed{|y - \hat{y}|}$ *Loss*

| | |
|---|---|
| $W = 0.5$ | $\hat{y} = 1.5 \overset{3.5}{\longleftrightarrow} y = 5$ |
| $W = 3$ | $\hat{y} = 6.5 \overset{1.5}{\longleftrightarrow} y = 5$ |
| $W = 2$ | $\hat{y} = 4.5 \overset{0.5}{\longleftrightarrow} y = 5$ |

# 梯度下降 Gradient Descent

目標：$\theta^* = \underset{\theta}{\operatorname{argmin}}\, L(\theta)$

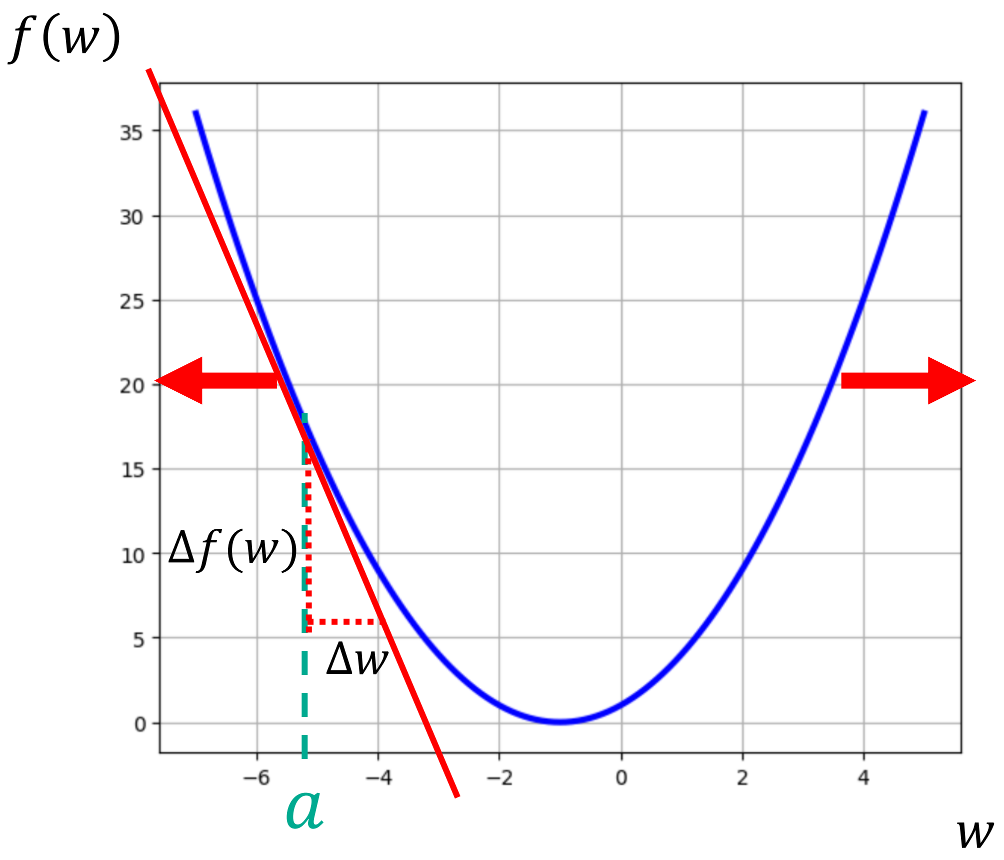
$L$ 為損失函數,
$\theta = \{w, b\}$ 為模型參數



# 梯度 Gradient

假設有一個函數$f(w)$，
梯度就是$f(w)$在$w = a$的切線斜率。

- 正負號會決定函數在$w = a$的增長方向。
- 其數值就是在這個方向上的增長變化量。

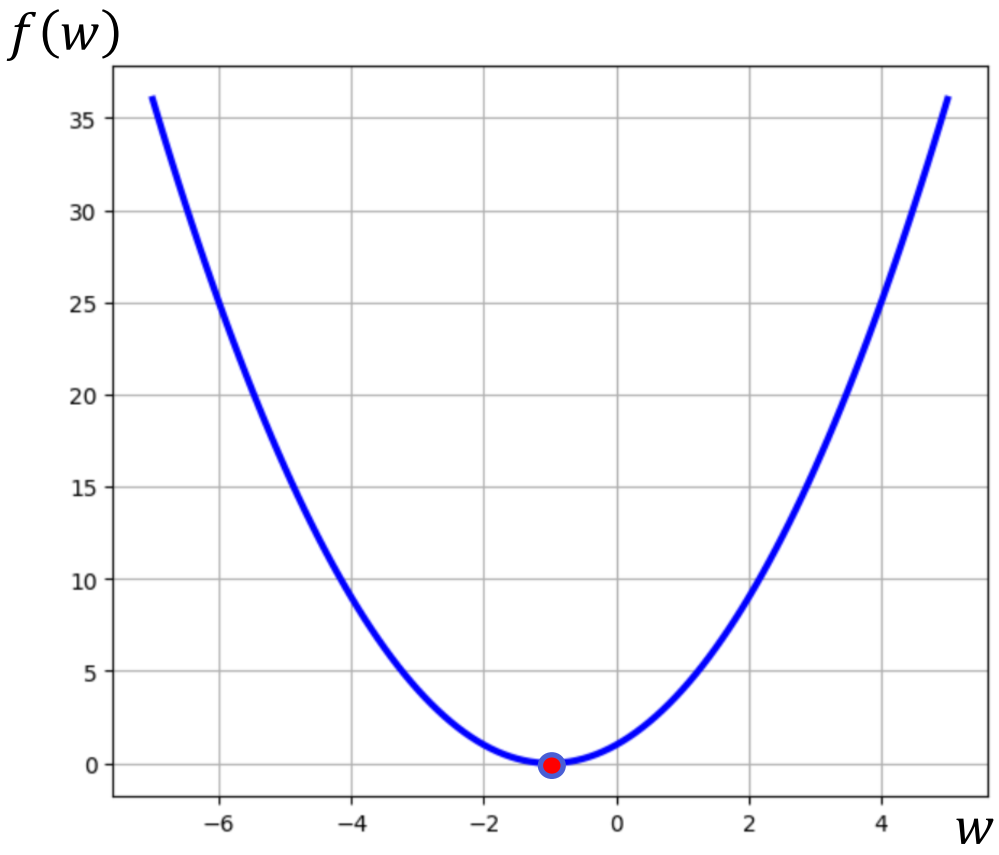


# 梯度 Gradient

- 假設給定的函數是：$f(w) = w^2 + 2w + 1$
- 先對$f(w)$進行微分，找出$f(x)$的導數

$$f'(w) = \frac{df(w)}{dw} = 2w + 2 = 2(w + 1)$$

- 梯度為 0 的時候會出現極值
- 當$w = -1$時， $f(w)$出現極小值（實際上這樣就是公式解，並不適用在神經網路的參數更新上）

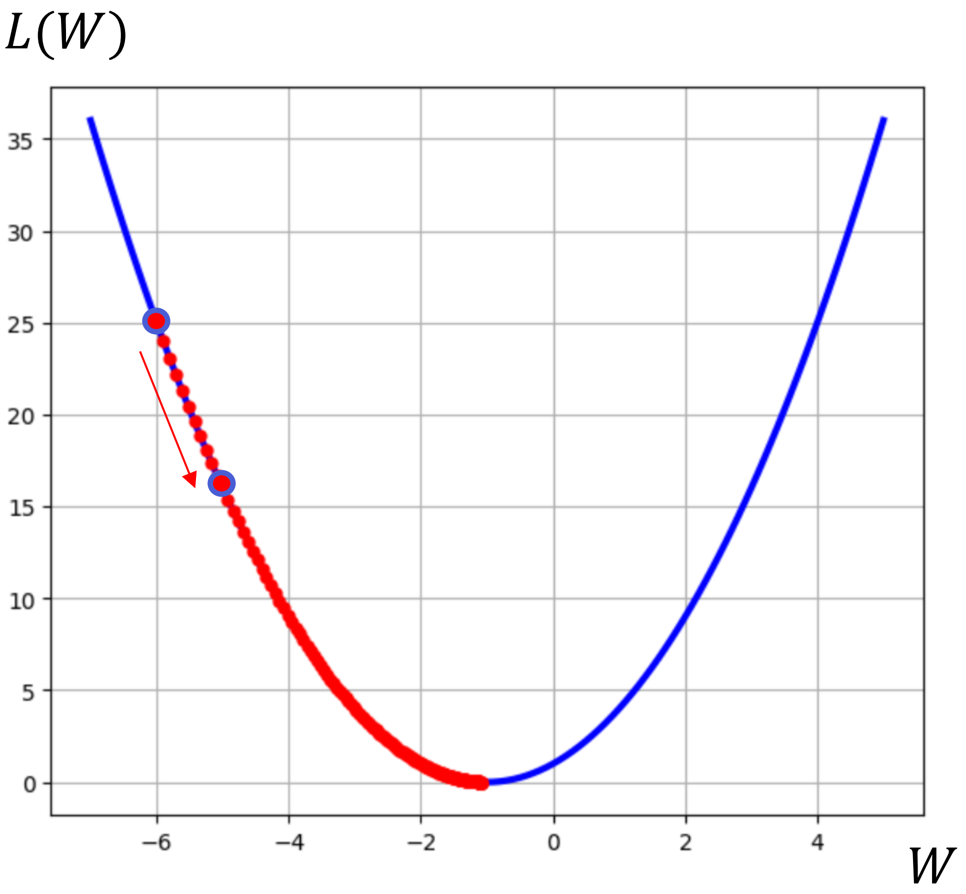


# 梯度下降 Gradient Descent

Loss $L(W) = f(W) = W^2 + 2W + 1$
$\nabla L(W) = f'(W) = 2(W+1)$

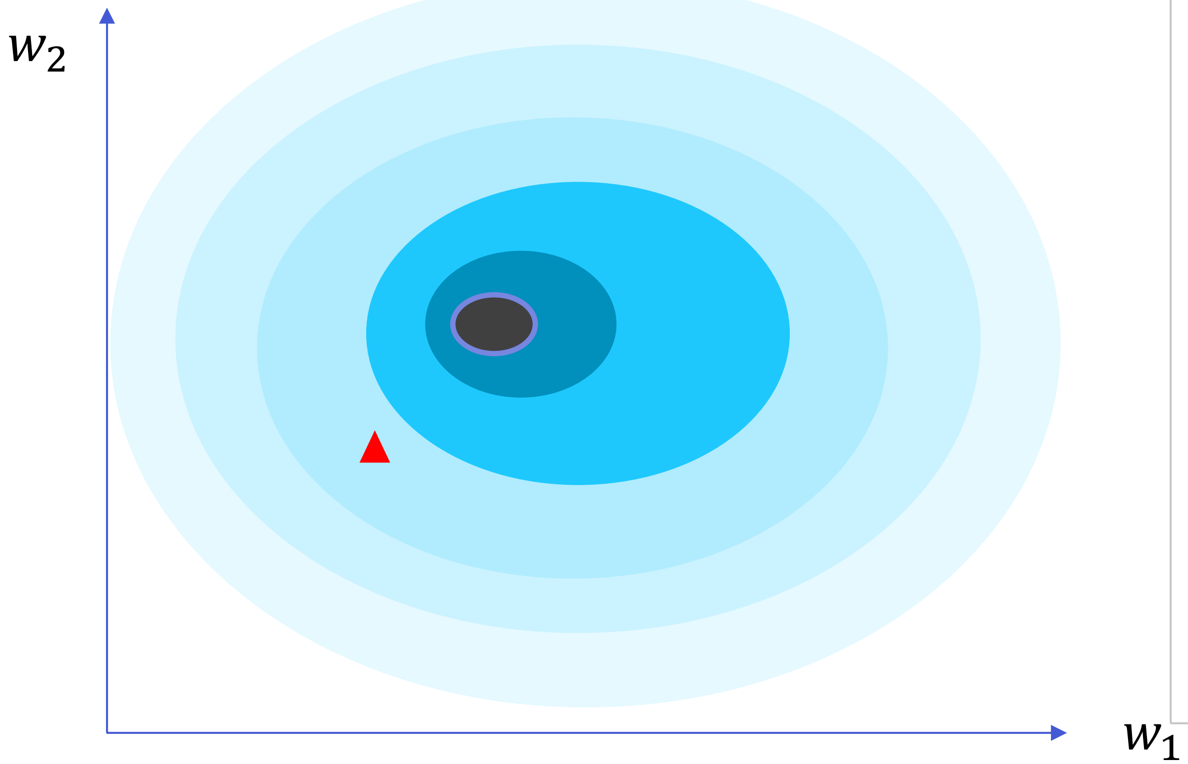
包含學習率調整的參數更新公式：
$W^{t+1} = W^{t} - \eta \cdot \nabla L(W^{t})$

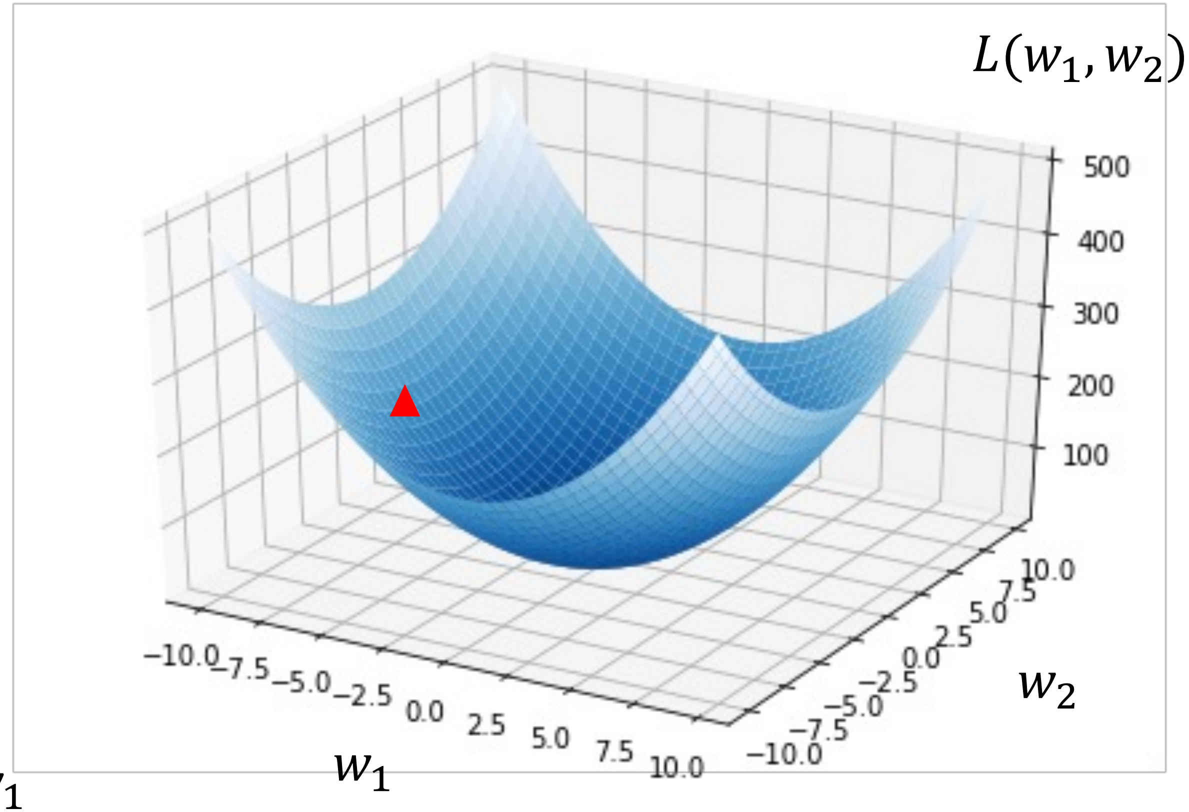
1. 初始化$W^t = -6$ , $Loss = L(-6) = 25$
2. 將$W^t$代入梯度函數得到梯度
$$\nabla L(-6) = f'(-6) = -10$$
3.用梯度更新$W^{t+1} = W^t - \eta \cdot \nabla L(W^t)$
$$W^{t+1} = -6 - 0.1(-10) = -5$$
4.將$W^{t+1}$代入梯度函數得到新的梯度
$$\nabla L(-5) = f'(-5) = -8$$
5.再次用梯度更新$W^{t+1} \rightarrow W^{t+2}$
$$W^{t+2} = -5 - 0.1(-8) = -4.2$$



# 梯度下降 Gradient Descent

畫成等高線圖，越中心 $L(w_1, w_2)$越低， ▲為一組初始的數值

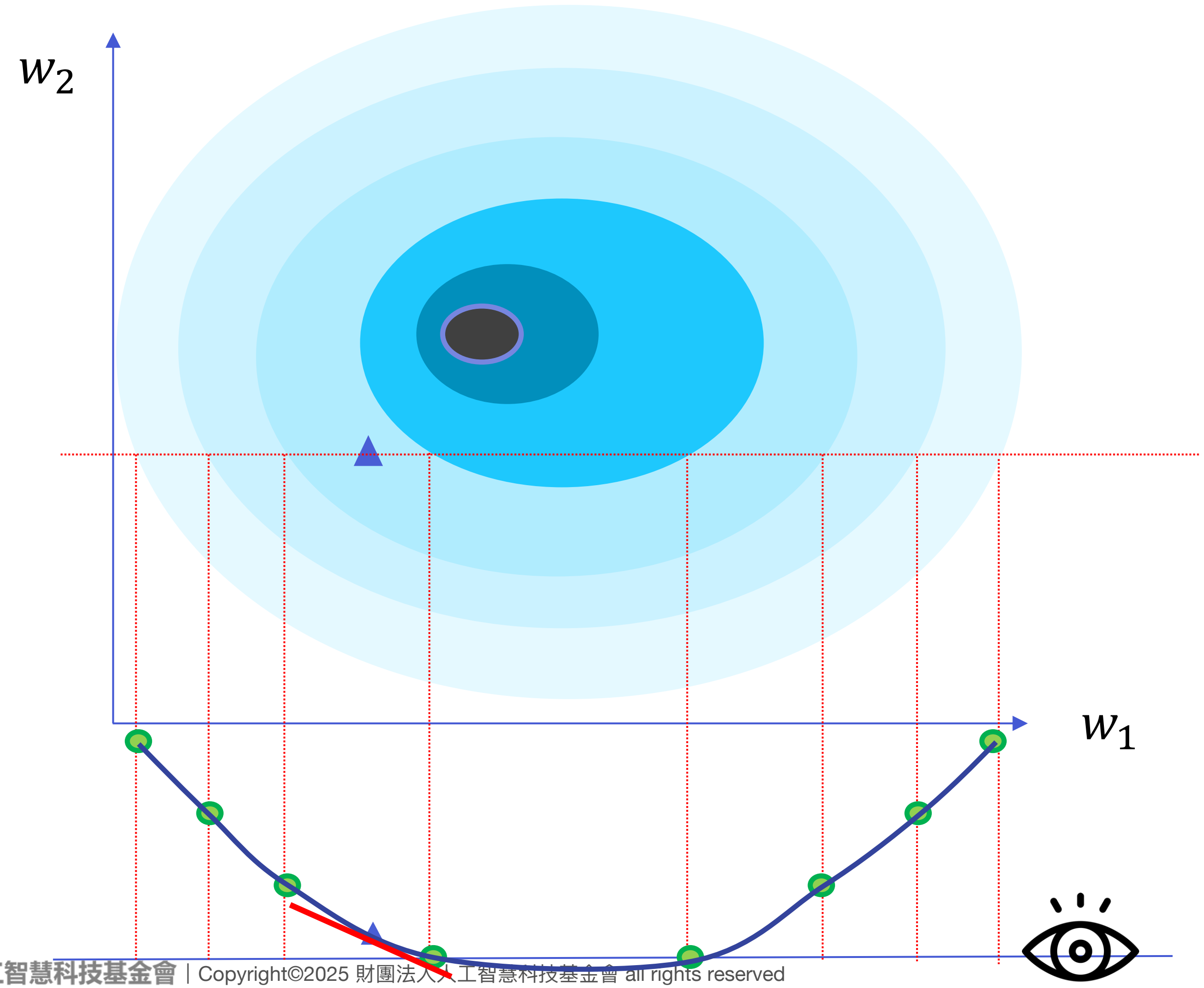


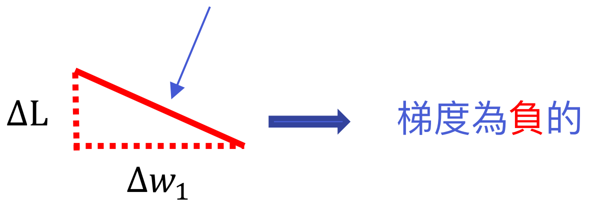

擴展到參數有兩個的狀況下，此時需要求取不同方向的變化量以得到梯度

# 梯度下降 Gradient Descent

畫成等高線圖，越中心 $L(w_1,w_2)$越低，　▲為一組初始的數值

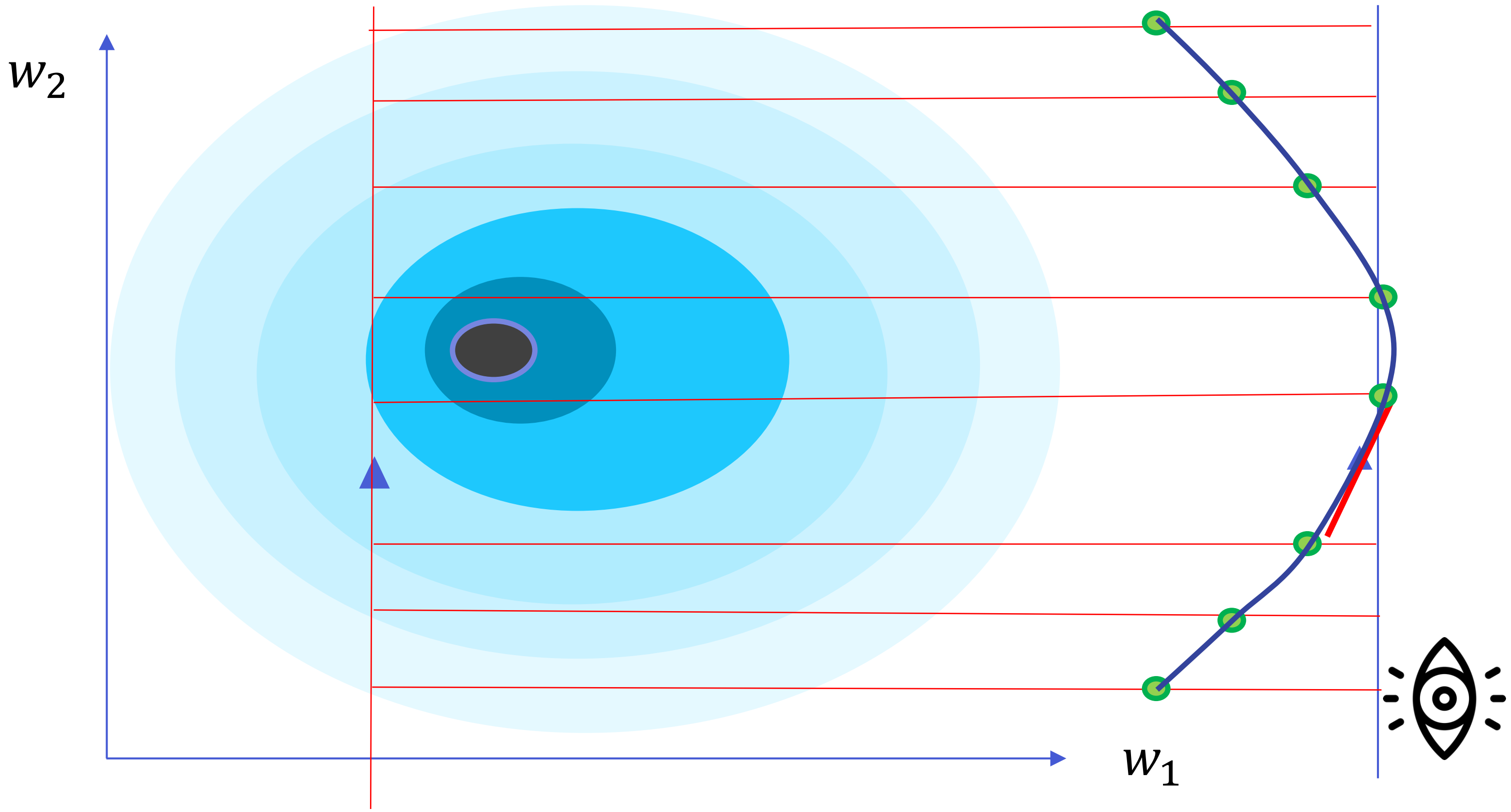



此處▲ 的梯度：$\frac{\partial L(w_1,w_2)}{\partial w_1}$

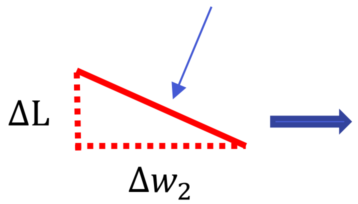


# 梯度下降 Gradient Descent

畫成等高線圖，越中心 $L(w_1, w_2)$越低， ▲為一組初始的數值

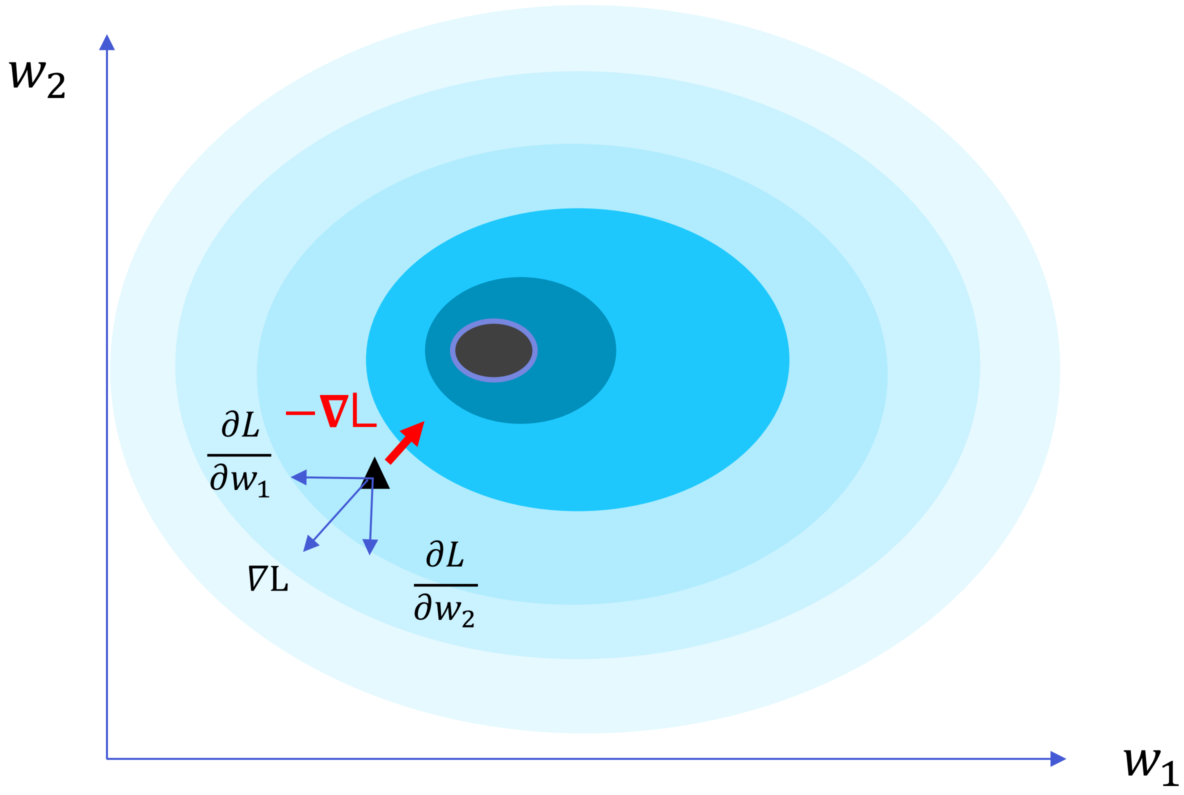



此處 ▲ 的梯度：$\dfrac{\partial L(w_1, w_2)}{\partial w_2}$




梯度為負的

# 梯度下降 Gradient Descent




▲的梯度方向為：$\overrightarrow{\nabla L} = \overrightarrow{\partial L_{w_1}} + \overrightarrow{\partial L_{w_2}}$

⇨ ▲ 下一次位置為 $▲^{t+1} = ▲^{t} - \eta \nabla L$

註：由於參數更新的目標是要Loss越來越小，因此更新方向是梯度的反方向 $-\boldsymbol{\nabla L}$

# 梯度下降 Gradient Descent

目標：$\theta^* = \underset{\theta}{\operatorname{argmin}}\, L(\theta)$

$L$: 損失函數,
$\theta = \{w, b\}$: 參數

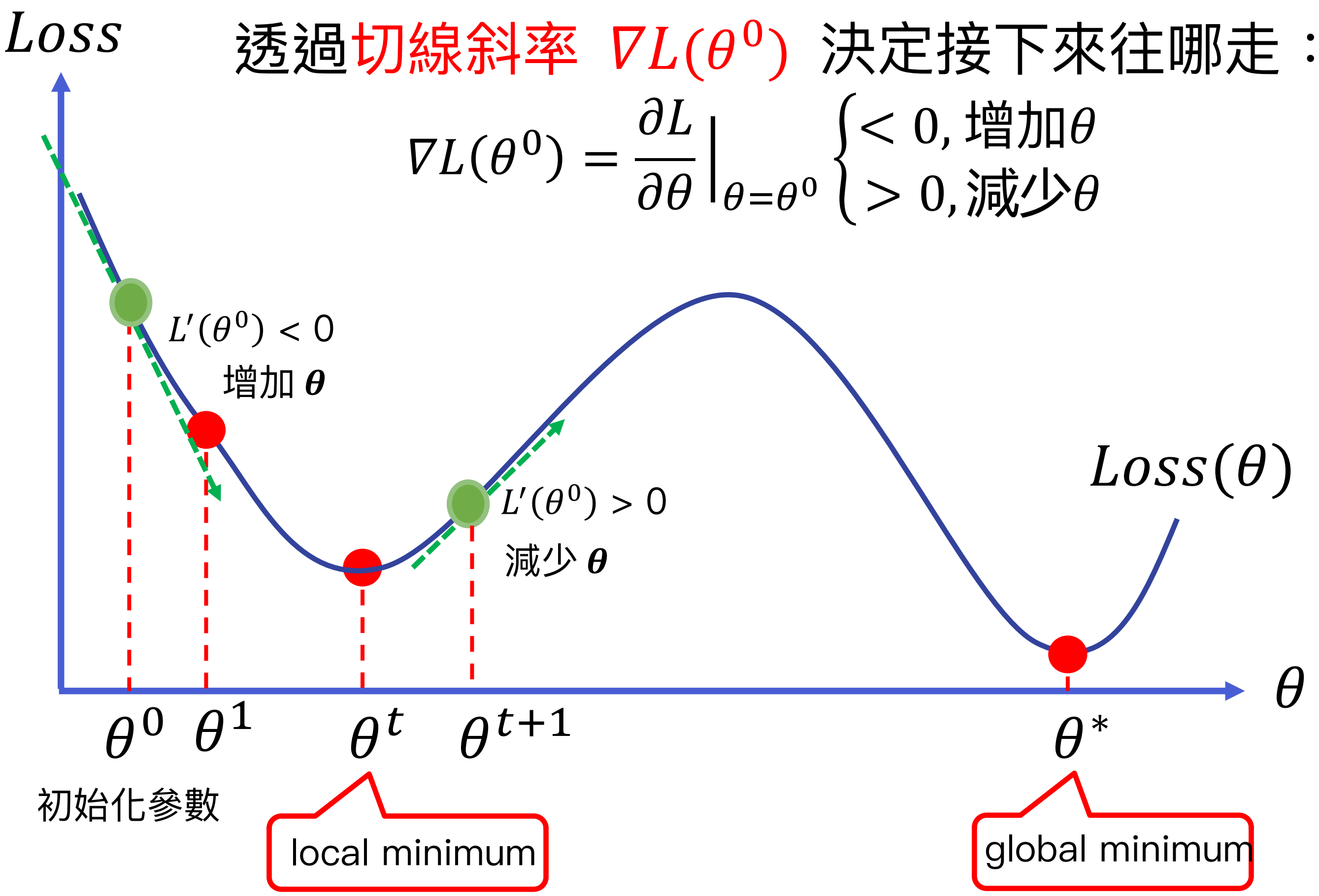
1. 隨機初始化參數$\theta^0$
2. 計算$L'(\theta^0)$決定方向
3. 訂定$\eta$學習率

$$\theta^{t+1} = \theta^t + \eta \cdot (-\nabla L(\theta^t))$$

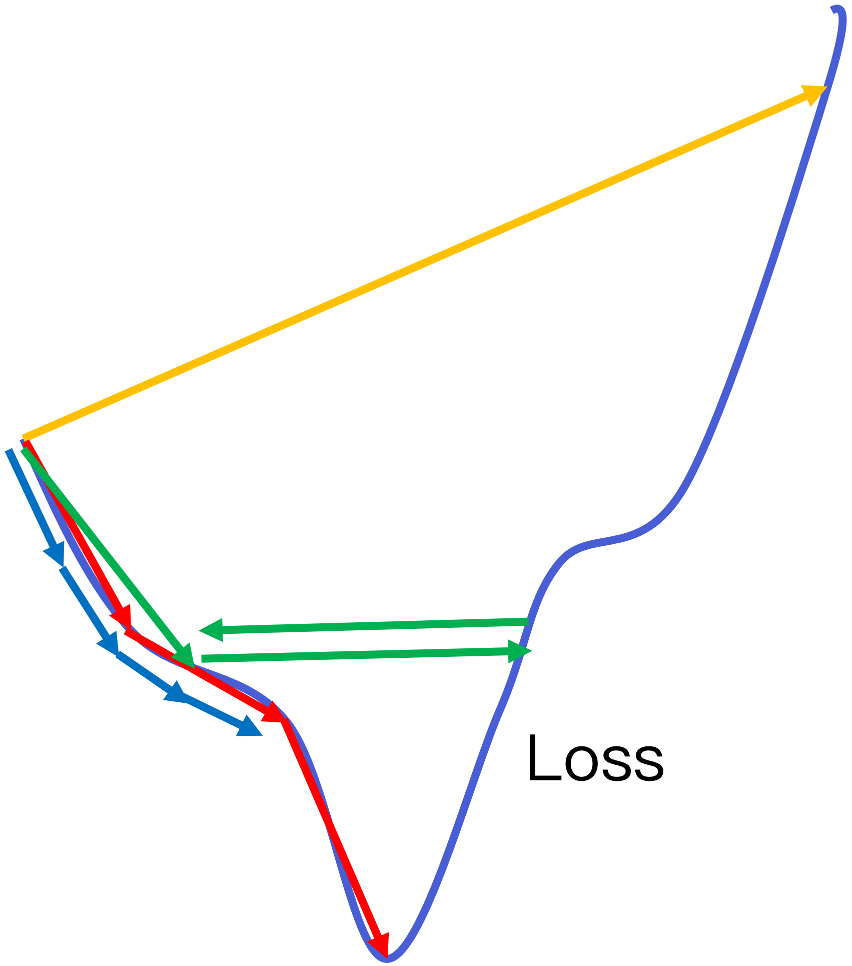
$$\nabla L(\theta) = \frac{\partial L}{\partial \theta} = \begin{bmatrix} \frac{\partial L}{\partial w} \\ \frac{\partial L}{\partial b} \end{bmatrix}$$

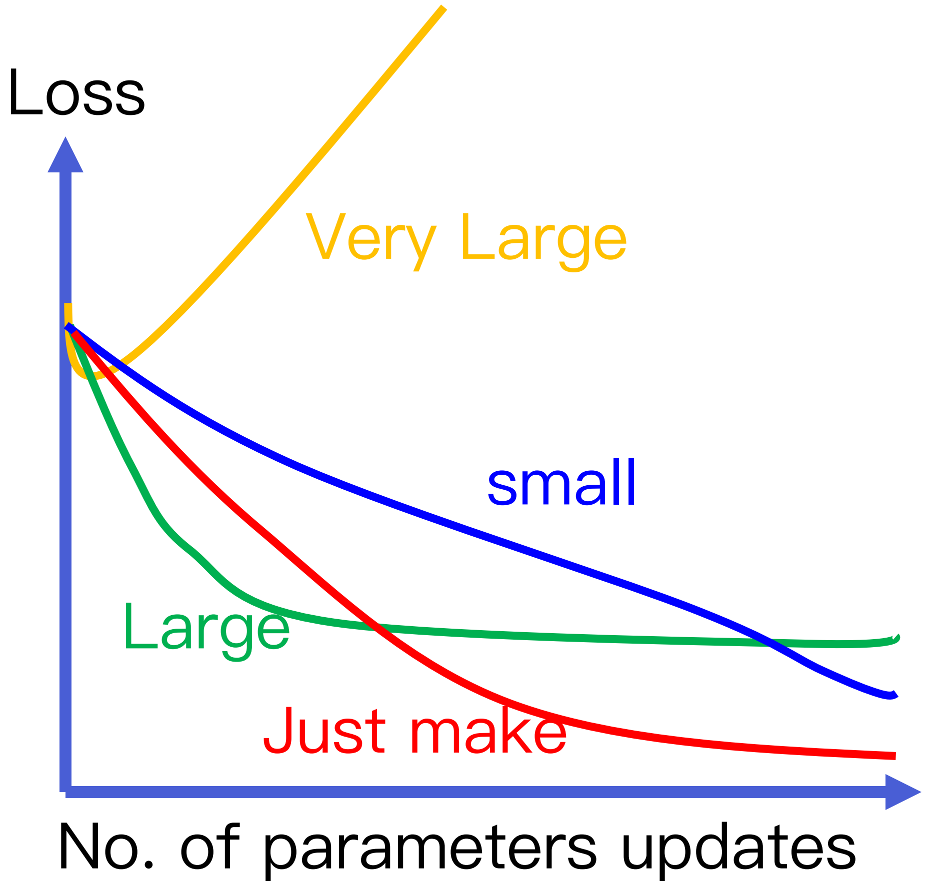
透過切線斜率 $\nabla L(\theta^0)$ 決定接下來往哪走：

$$\nabla L(\theta^0) = \left.\frac{\partial L}{\partial \theta}\right|_{\theta=\theta^0} \begin{cases} < 0, 增加\theta \\ > 0, 減少\theta \end{cases}$$



# 學習率 Learning rate






圖片來源 :李宏毅

# 影響參數更新因素

## 梯度下降 Gradient Descent

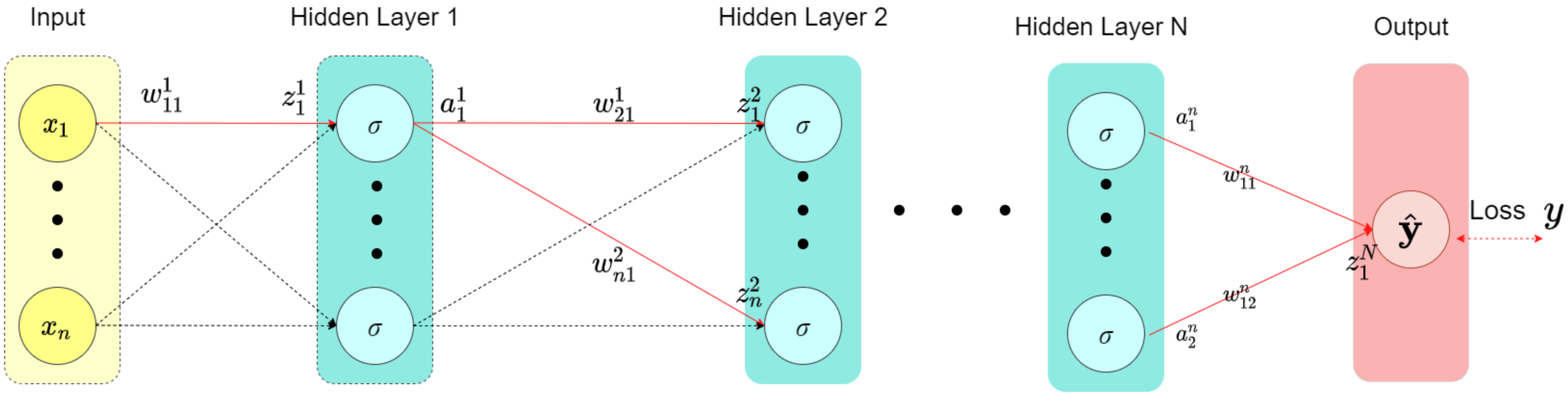
$$\theta^{t+1} = \theta^{t} - \eta \cdot \nabla L(\theta^{t})$$

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$\hat{y} = \sigma(W^{\mathrm{T}}X + b)$$

1. 受 學習率 $\eta$ 影響
2. 受 損失函數 $L(\theta)$ 影響
3. 受 參數初始化 $\theta^{0} = \{w, b\}$ 影響
4. 受 樣本數量 $n$ 影響
5. 受 激勵函數 $\sigma$ 影響

# 梯度下降的缺點

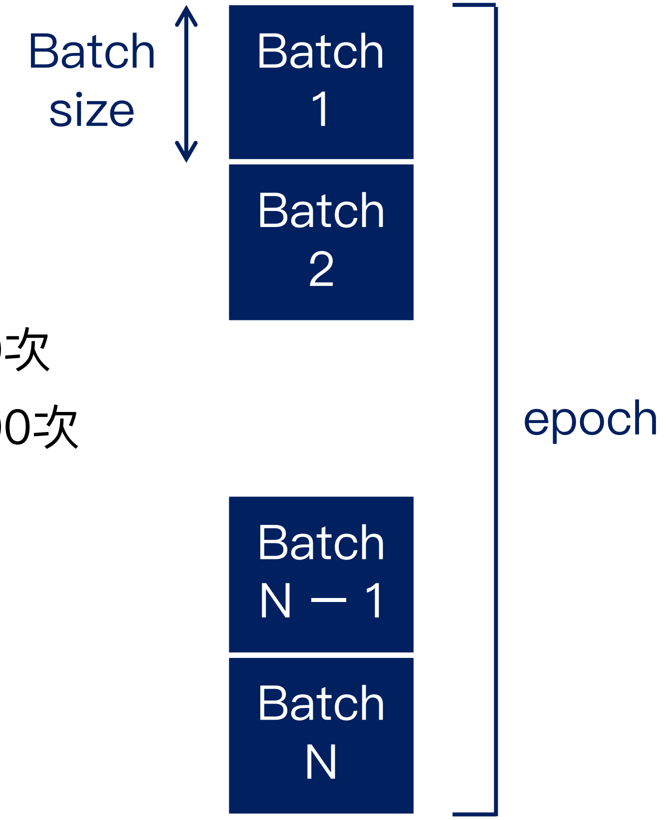



- 梯度下降看完全部的資料集（即1個epoch）才更新一次，收斂速度很慢，有辦法加速嗎？
  → 使用「Mini–Batch」來解決

- 梯度下降法不保證找到全域最佳解，該怎麼解決？
  → 利用「動量 Momentum 」的概念，來突破區域最小值

# Mini–batch & Epoch

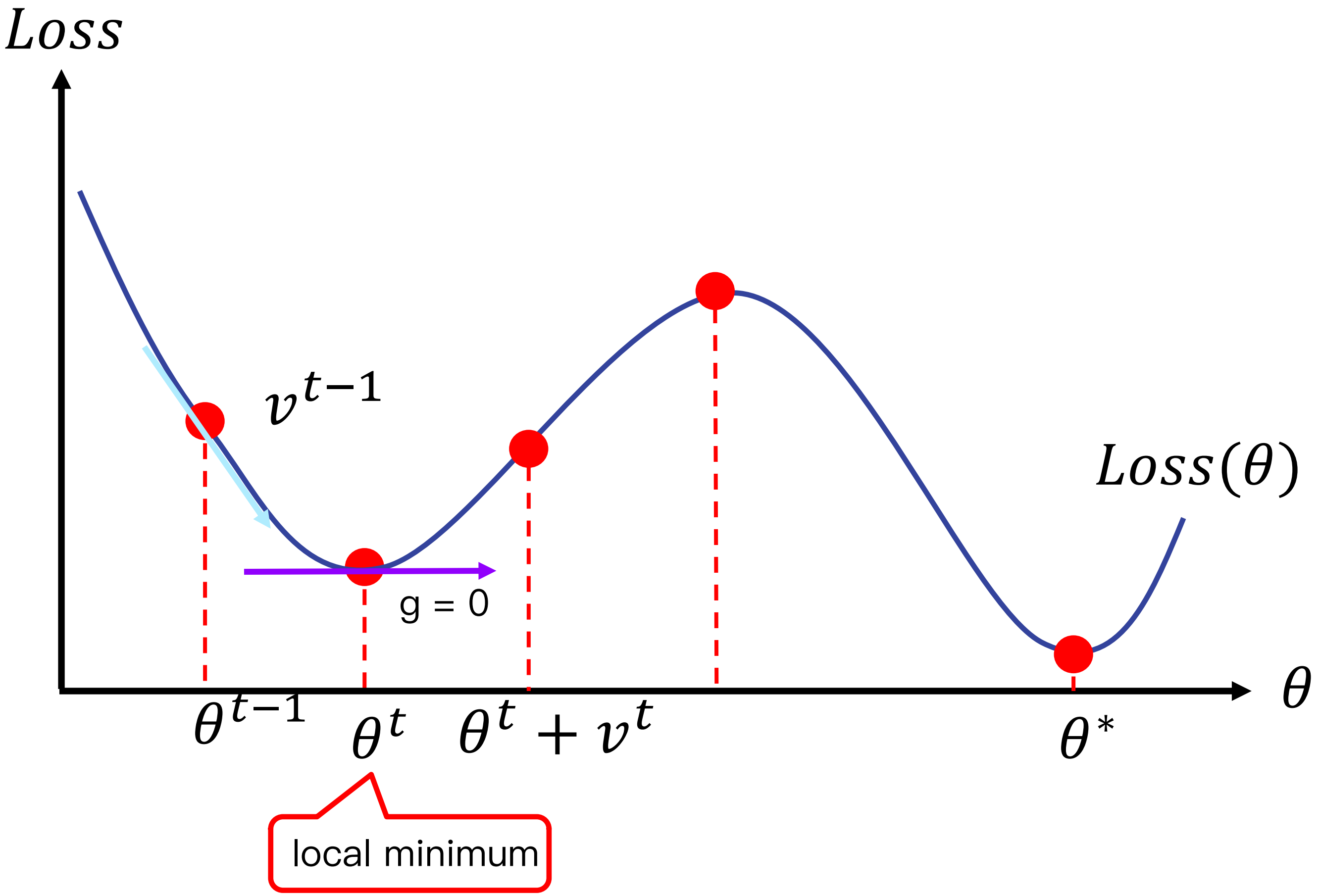
- 1個epoch = 一輪的訓練 = 模型看完整個訓練資料集1次
- Mini–batch：把所有的資料拆分成多份
  - 假設資料集有一1000筆資料
  - Batch–size設定為100，則可以拆分成10份 → 1個 epoch內會更新10次
  - Batch–size設定為10，則可以拆分成100份 → 1個 epoch內會更新100次
- 如何設定 Batch size
  - 常見的設定值 2 的次方：8、16、32、64、128、256 .....
  - 太小 ➔ 無法善用 GPU 平行運算的優勢
  - 太大 ➔ 塞不進 GPU 記憶體



# 不同樣本數量的梯度下降法

- 用 一個 epoch 來求梯度 ➔ 稱為 (Batch) Gradient Descent, (B)GD
- 用 一筆資料 來求梯度 ➔ 稱為 Stochastic Gradient Descent, SGD
- 用 一個 Mini-batch 來求梯度 ➔ 稱為 Mini-batch Gradient Descent

# 動量 Momentum




SGD：

$$\theta^{t+1} = \theta^{t} - \eta(\nabla L(\theta^{t}))$$

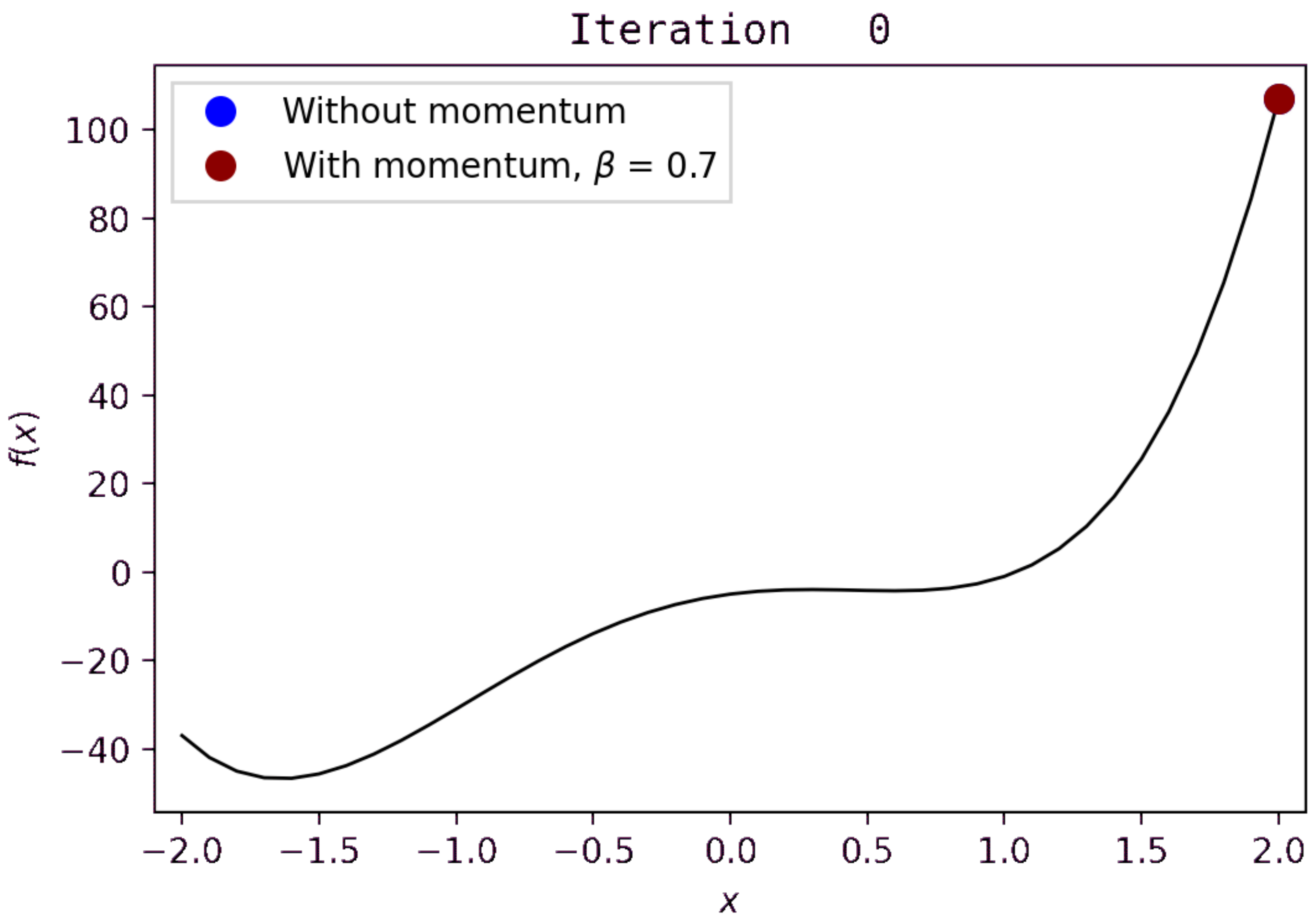
SGD + momentum：

$$v^{t} = \begin{cases} \eta \nabla L(\theta^{t}), & t = 0 \\ \beta v^{t-1} + \eta(\nabla L(\theta^{t})), & t \geq 1 \end{cases}$$

$$\theta^{t+1} = \theta^{t} - v^{t}$$

$\beta$為動量項係數，一般設為0.9。

# 動量 Momentum




SGD：

$$\theta^{t+1} = \theta^{t} - \eta(\nabla L(\theta^{t}))$$

SGD + momentum：

$$v^{t} = \begin{cases} \eta\nabla L(\theta^{t}), & t = 0 \\ \beta v^{t-1} + \eta(\nabla L(\theta^{t})), & t \geq 1 \end{cases}$$

$$\theta^{t+1} = \theta^{t} - v^{t}$$

$\beta$為動量項係數，一般設為0.9。

# AdaGrad

- 在前面的優化中，有一個超參數一直都是固定值，但它也是極為重要的數值，那就是Learning rate。
- 我們應該要試著讓此超參也隨著梯度去改變。當梯度趨近於最小值時，Learning rate也跟著變小。
- $\epsilon$為平滑值，加上$\epsilon$的原因是為了不讓分母為0
- $n$為前面所有梯度值的平方和，利用前面學習的梯度值平方和來調整Learning rate

$$W \leftarrow W - \eta \frac{1}{\sqrt{n+\epsilon}} \frac{\partial L}{\partial W}$$

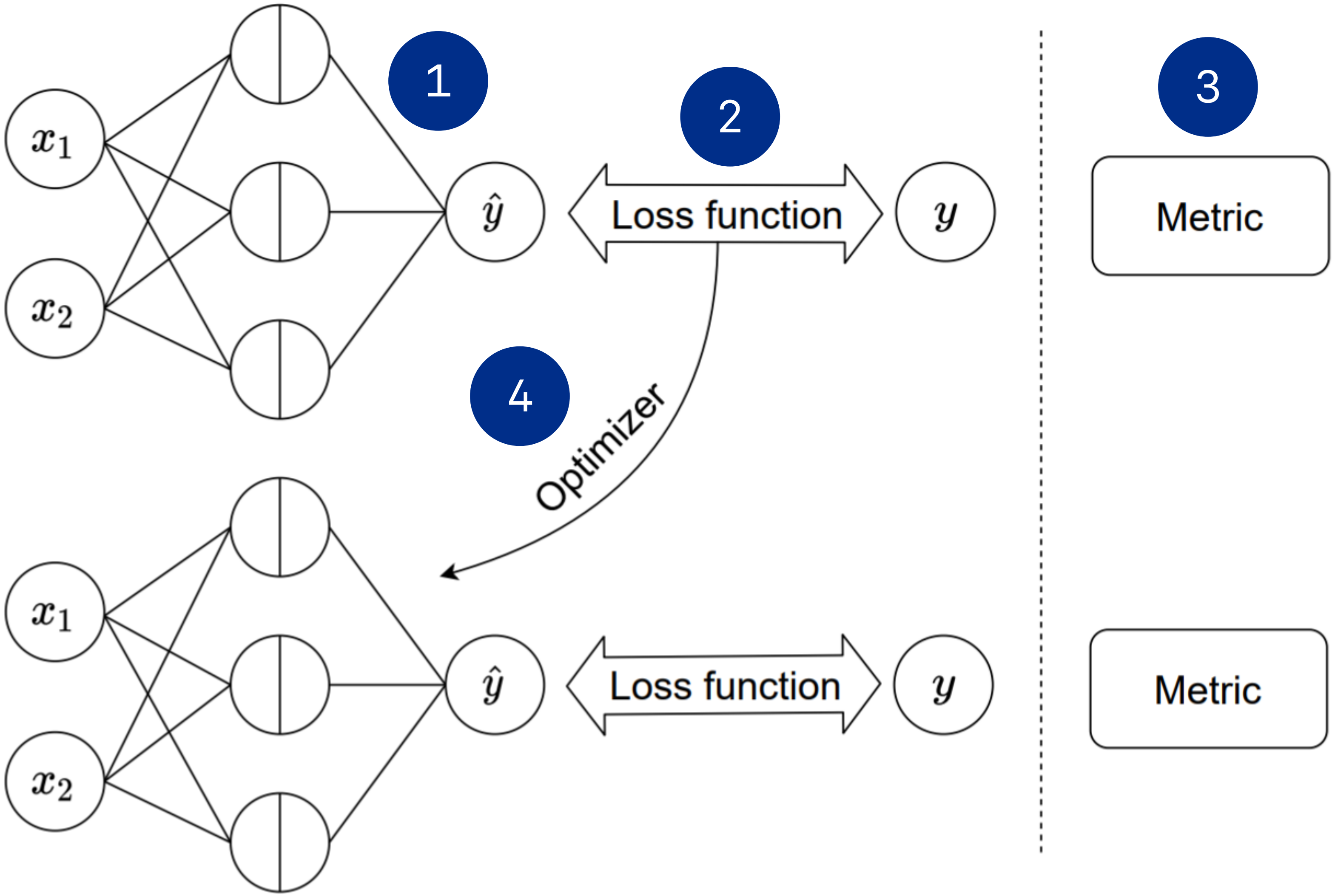
$$n = \sum_{r=1}^{t} \left(\frac{\partial L_r}{\partial W_r}\right)^2$$

$$W \leftarrow W - \boxed{\eta \frac{1}{\sqrt{\sum_{r=1}^{t} \left(\frac{\partial L_r}{\partial W_r}\right)^2 + \epsilon}}} \frac{\partial L}{\partial W}$$

# 模型訓練流程

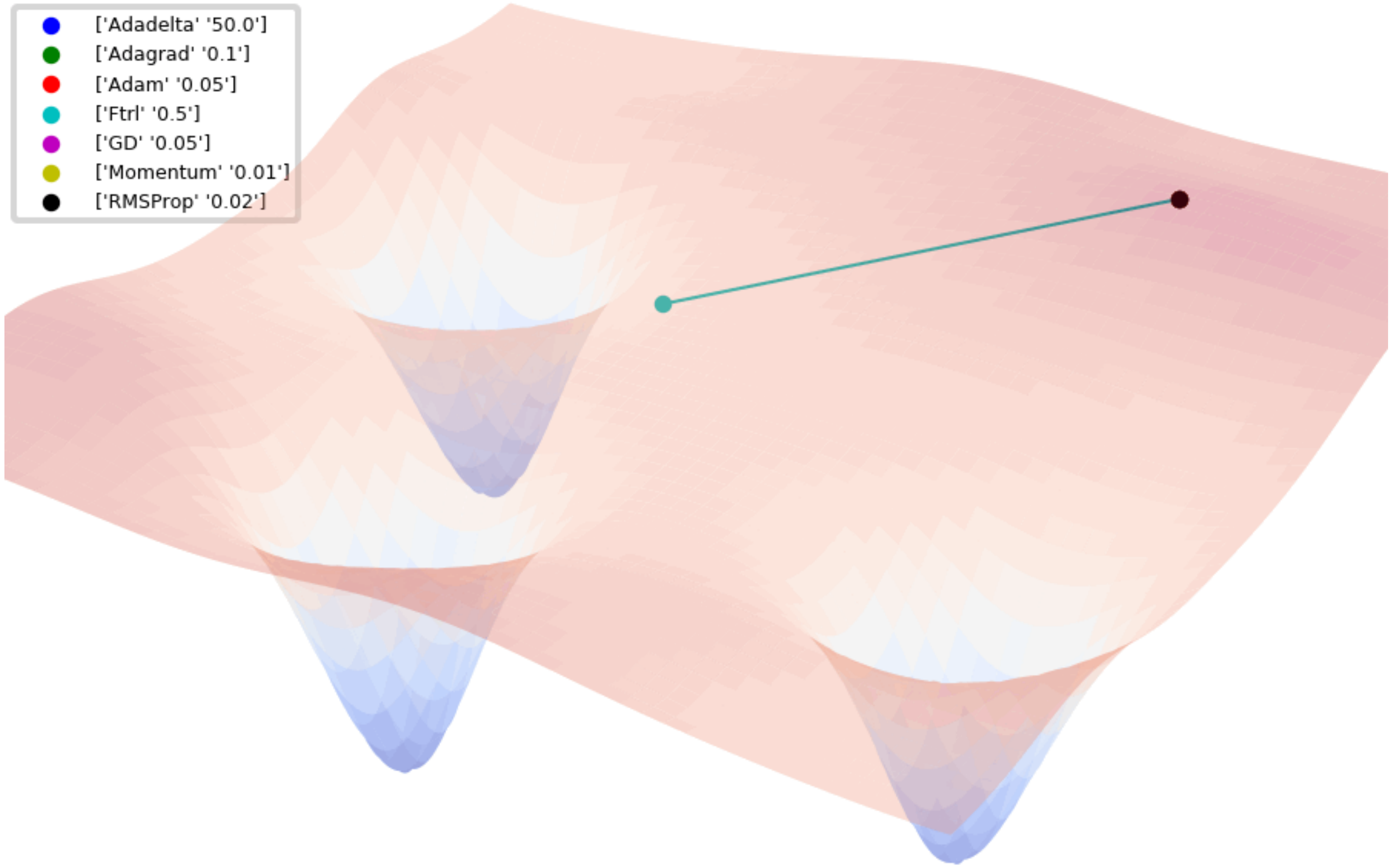
第 1 次訓練

第 2 次訓練



# 最佳化器 Optimizer

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



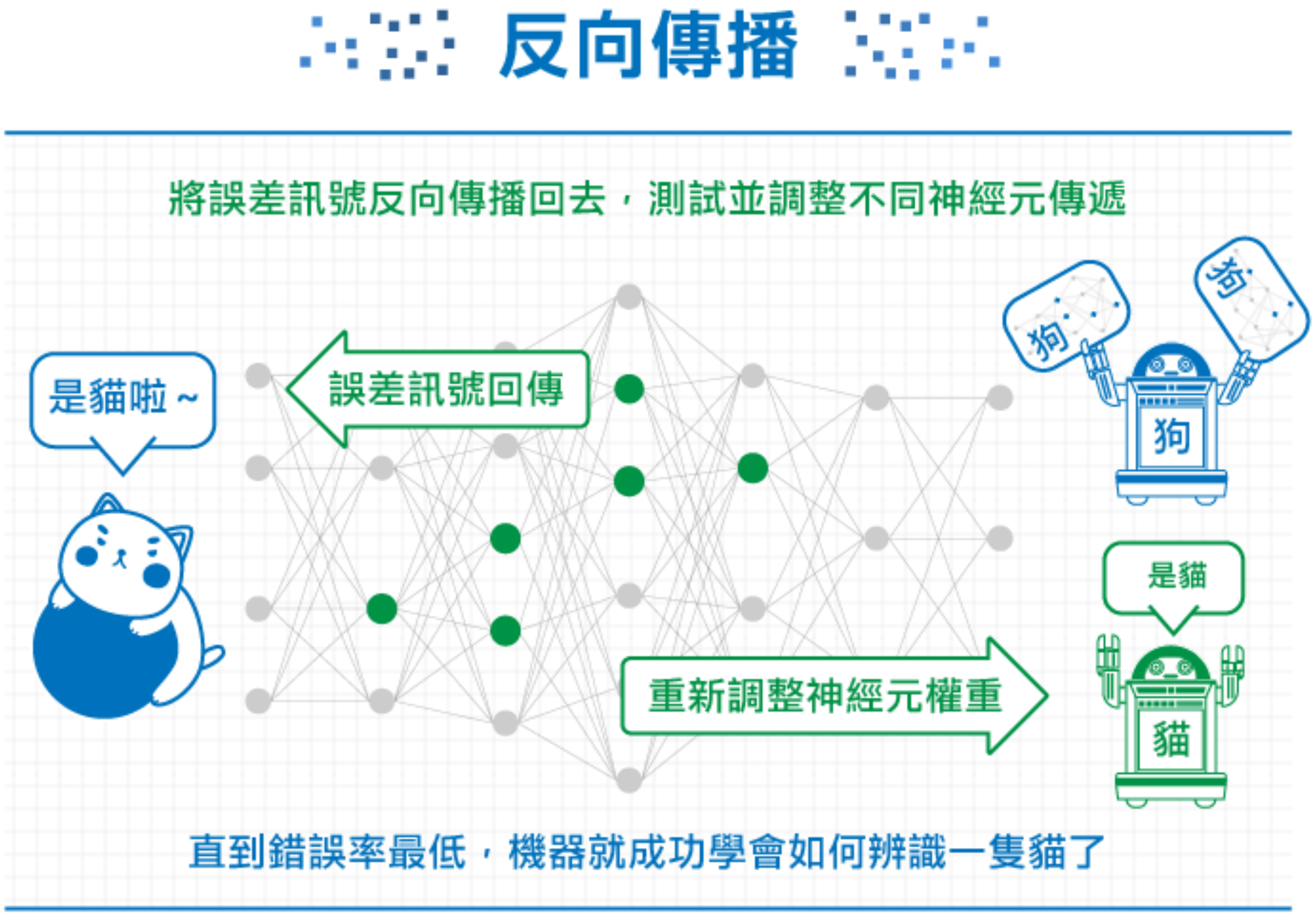



資料來源: https://keras.io/api/optimizers/

Back propagation

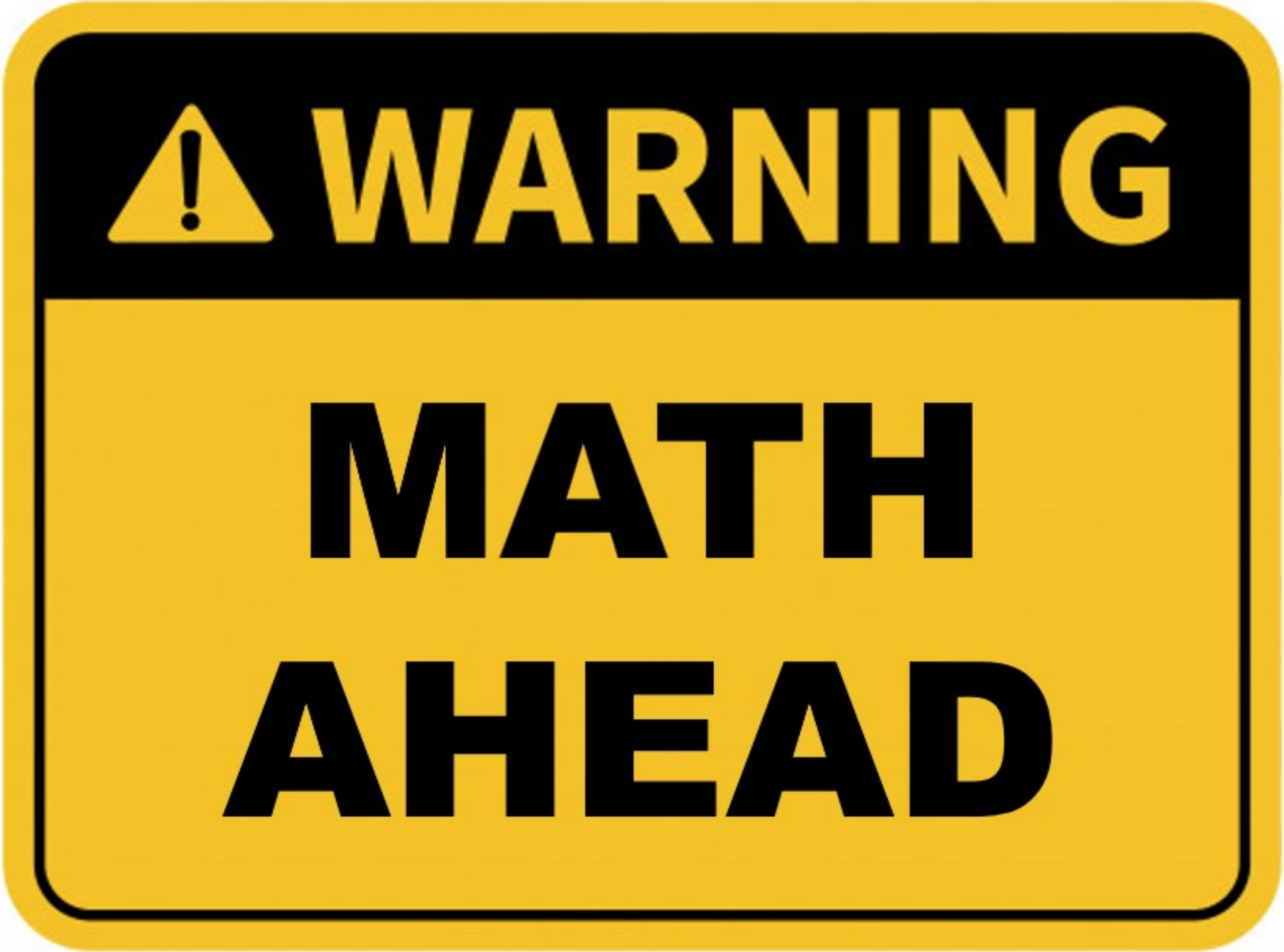
# 反向傳播

# 反向傳播 Back propagation






資料來源：stockfeel


WARNING
MATH
AHEAD

# 反向傳播使用到的微積分：連鎖律

- $y = f(x) \qquad z = g(y)$
- $z = g\big(f(x)\big) = (g \circ f)(x)$
- 如果給 $x$ 變化，會影響到 $y$，$y$ 會影響到 $z$
- 此時 $z$ 對 $x$ 的微分

$$\nabla z = \frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

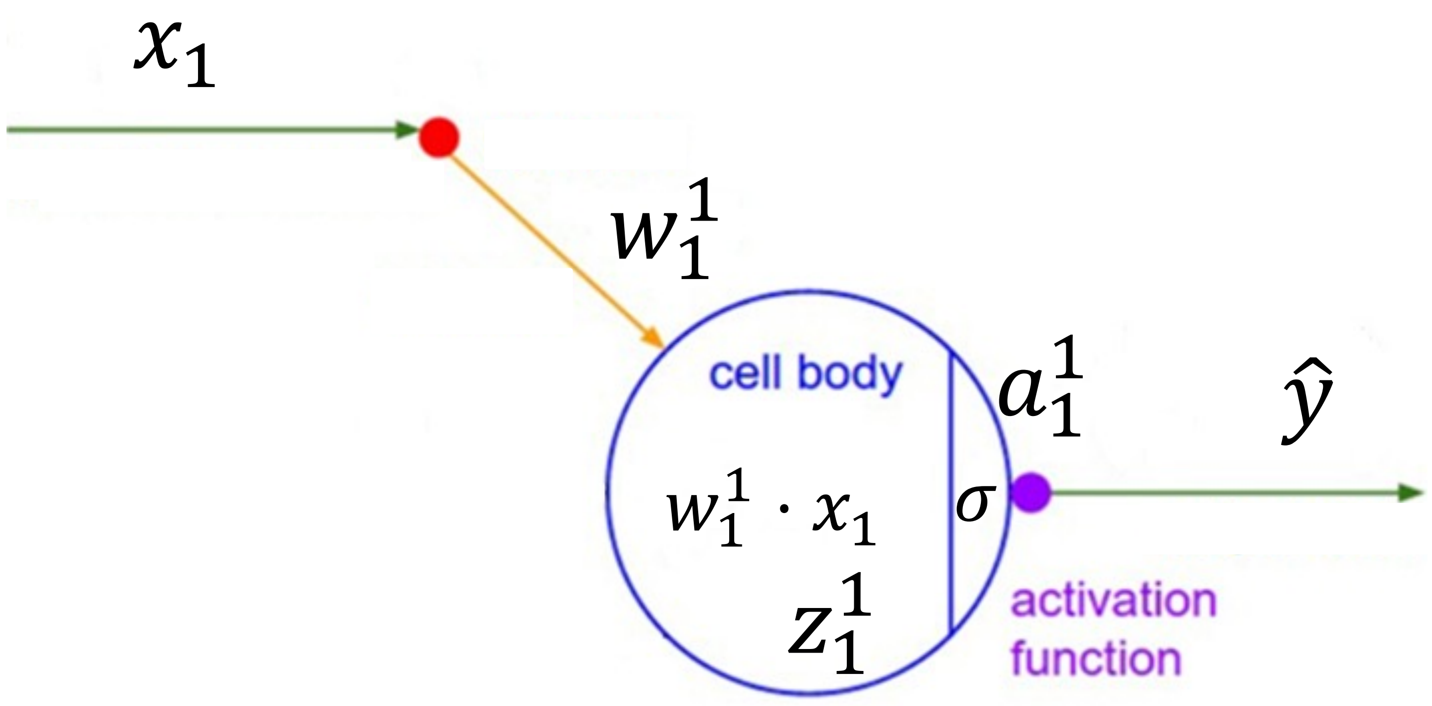
- $s = g(x),\ t = h(x),\ z = f(s,t)$

$$\begin{tikzcd}
 & \Delta s \arrow[dr] & \\
\Delta x \arrow[ur] \arrow[dr] & & \Delta z \\
 & \Delta t \arrow[ur] &
\end{tikzcd}$$

- 此時 $z$ 對 $x$ 的微分

$$\nabla z = \frac{dz}{dx} = \frac{\partial z}{\partial s}\frac{ds}{dx} + \frac{\partial z}{\partial t}\frac{dt}{dx}$$

# 反向傳播 Back propagation




$$
\begin{aligned}
z_1^1 &= w_1^1 \cdot x_1 \\
a_1^1 &= \sigma(z_1^1) \\
\hat{y} &= a_1^1 \\
L &= loss(\hat{y}, y)
\end{aligned}
$$

$$\widehat{w_1^1} = w_1^1 + \eta \cdot \left(-\frac{\partial L}{\partial w_1^1}\right)$$

# 反向傳播 Back propagation

$$\widehat{w_1^1} = w_1^1 - \eta \cdot \left(\frac{\partial L}{\partial w_1^1}\right)$$

根據連鎖律

$$\frac{\partial L}{\partial w_1^1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_1^1} \frac{\partial a_1^1}{\partial z_1^1} \frac{\partial z_1^1}{\partial w_1^1}$$

$$L = loss(\hat{y}, y)$$

$$\hat{y} = a_1^1 = \sigma(z_1^1)$$

$$z_1^1 = w_1^1 \cdot x_1$$

**Chain rule:**

- $y = f(x), z = g(y), \; z = g(f(x)) = (g \circ f)(x)$

$$(g \circ f)'(x) = \frac{dg}{dx} = \frac{dg}{df}\frac{df}{dx}$$

- $z = f(x, y)$, where $x = g(t), y = h(t)$

$$\frac{df}{dt} = \frac{\partial f}{\partial g}\frac{dg}{dt} + \frac{\partial f}{\partial h}\frac{dh}{dt}$$

# 反向傳播 Back propagation

$$\hat{y} = a_1^1 = \sigma(z_1^1) \qquad z_1^1 = w_1^1 \cdot x_1$$

根據連鎖律

$$\frac{\partial L}{\partial w_1^1} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial a_1^1}\frac{\partial a_1^1}{\partial z_1^1}\frac{\partial z_1^1}{\partial w_1^1} = \frac{\partial}{\partial \hat{y}}\underbrace{\boxed{\frac{1}{2n}(\hat{y}-y)^2}}_{L}\frac{\partial}{\partial a_1^1}\underbrace{\boxed{a_1^1}}_{\hat{y}}\frac{\partial}{\partial z_1^1}\underbrace{\boxed{\sigma(z_1^1)}}_{a_1^1}\frac{\partial}{\partial w_1^1}\underbrace{\boxed{(w_1^1 \cdot x_1)}}_{z_1^1}$$

損失函數 MSE

$$L = \frac{1}{2n}(\hat{y}-y)^2$$

$$L' = \frac{1}{n}(\hat{y}-y)$$

激勵函數 Sigmoid

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

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

$$= \sigma(z)(1-\sigma(z))$$

# 反向傳播 Back propagation

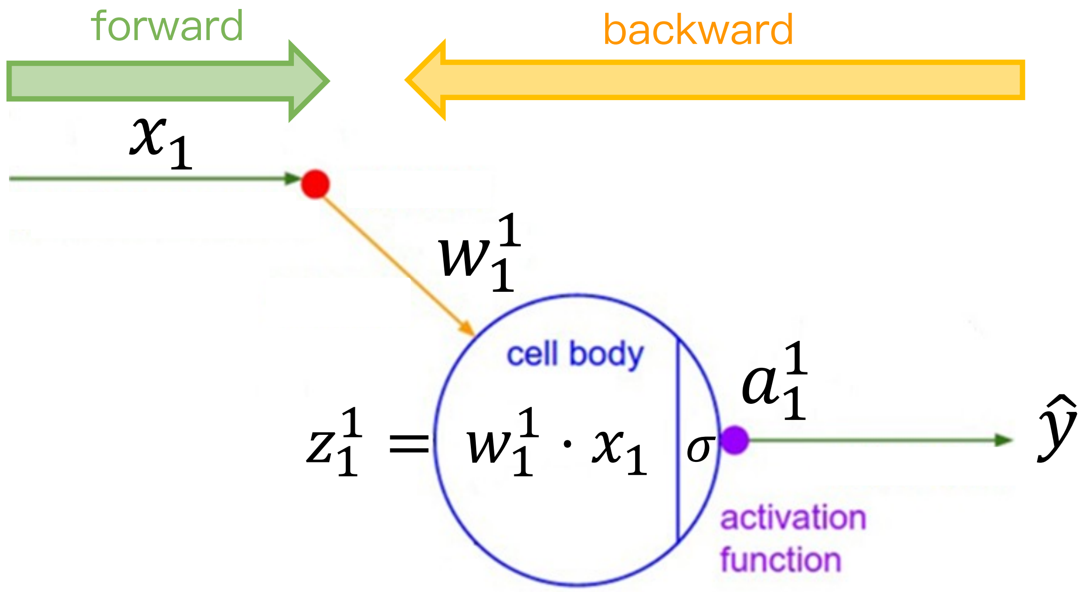
$$\frac{\partial L}{\partial w_1^1} = \underbrace{\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial a_1^1} \cdot \frac{\partial a_1^1}{\partial z_1^1}}_{\text{backward}} \cdot \underbrace{\frac{\partial z_1^1}{\partial w_1^1}}_{\text{forward}} = \frac{1}{n}(\hat{y} - y) \cdot 1 \cdot \sigma(z_1^1)(1 - \sigma(z_1^1)) \cdot x_1$$

$$z_1^1 = w_1^1 \cdot x_1$$

$$\hat{y} = a_1^1 = \sigma(z_1^1)$$

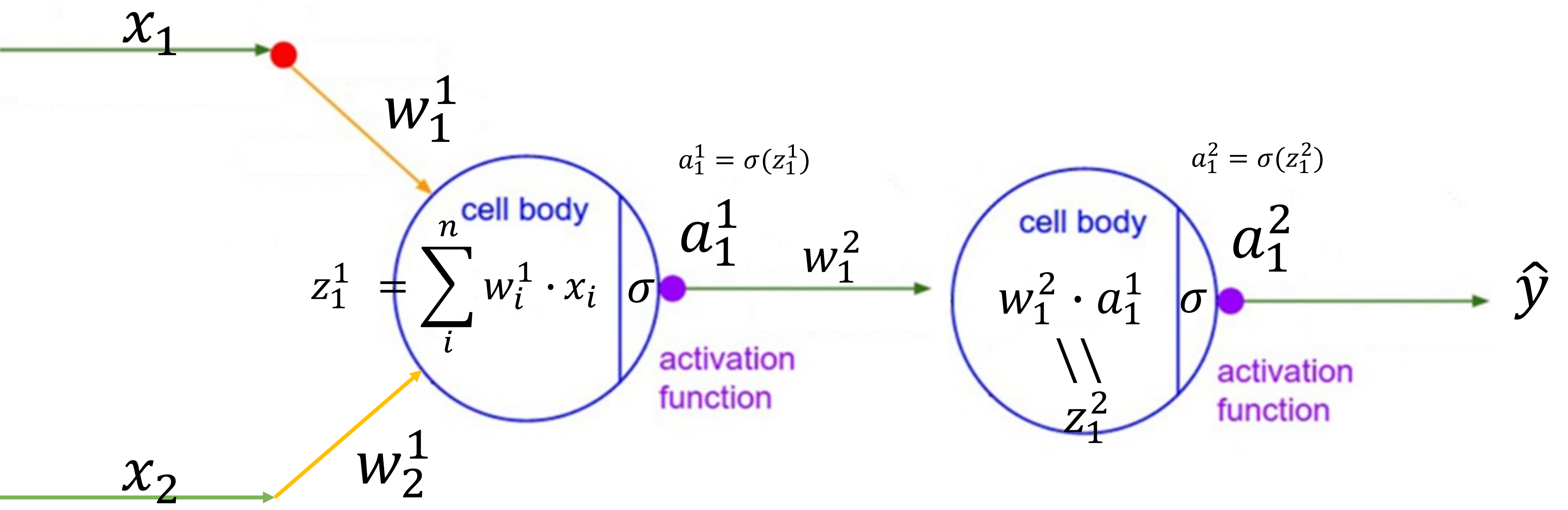
$$L = loss(\hat{y}, y)$$

$$\widehat{w_1^1} = w_1^1 + \eta \cdot (-\frac{\partial L}{\partial w_1^1})$$



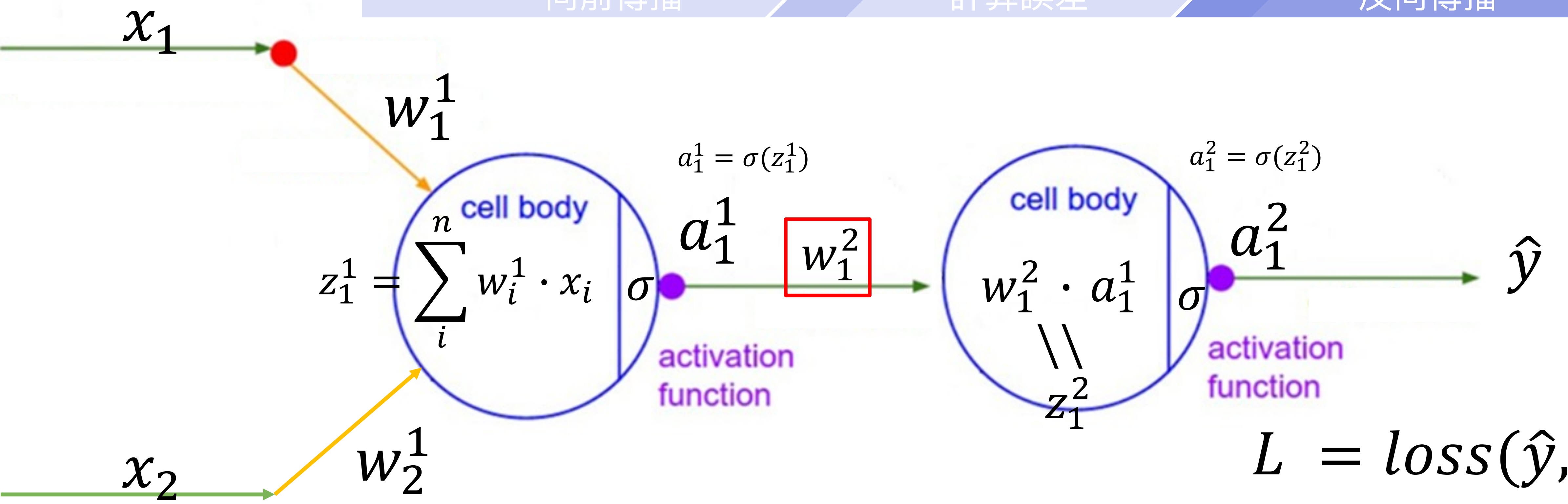
# 反向傳播 Back propagation

從單層神經元延伸到多層神經元




$$L = loss(\hat{y}, y)$$

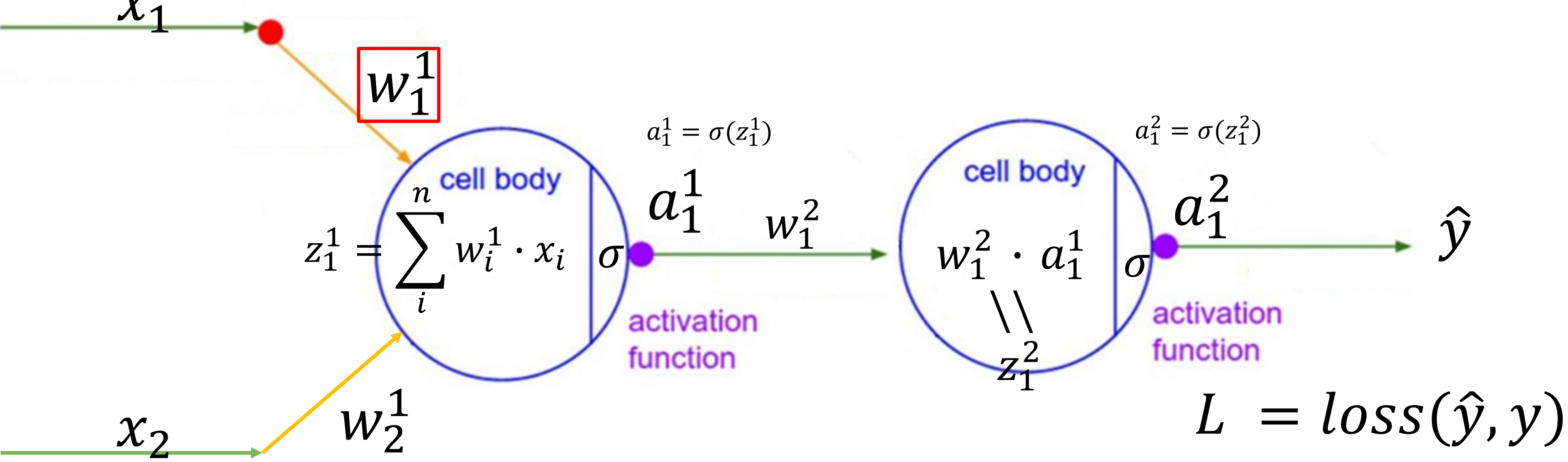



$$L = loss(\hat{y}, y)$$

$$\widehat{w_1^2} = w_1^2 - \eta \cdot \frac{\partial L}{\partial w_1^2}$$

$$\frac{\partial L}{\partial w_1^2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_1^2} \frac{\partial a_1^2}{\partial z_1^2} \frac{\partial z_1^2}{\partial w_1^2}$$

$$\widehat{w_1^2} = w_1^2 - \eta \cdot \boxed{\frac{1}{n}(\hat{y} - y) \cdot \sigma(z_1^1)(1 - \sigma(z_1^1))} \cdot \boxed{a_1^1}$$




$$\widehat{w_1^2} = w_1^2 - \eta \cdot \frac{\partial L}{\partial w_1^2}$$

$$\frac{\partial L}{\partial w_1^2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_1^2} \frac{\partial a_1^2}{\partial z_1^2} \frac{\partial z_1^2}{\partial w_1^2}$$

$$\widehat{w_1^1} = w_1^1 + \eta \cdot (- \qquad )$$

$$\widehat{w_1^1} = w_1^1 - \eta \cdot \frac{\partial L}{\partial w_1^1} \qquad \frac{\partial L}{\partial w_1^1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_1^2} \frac{\partial a_1^2}{\partial z_1^2} \frac{\partial z_1^2}{\partial a_1^1} \frac{\partial a_1^1}{\partial z_1^1} \frac{\partial z_1^1}{\partial w_1^1}$$

$$\frac{\partial L}{\partial w_1^1} = \frac{\partial}{\partial \hat{y}} \overset{L}{\boxed{\frac{1}{2n}(\hat{y}-y)^2}} \frac{\partial}{\partial a_1^2} \overset{\hat{y}}{\boxed{a_1^2}} \frac{\partial}{\partial z_1^2} \overset{a_1^2}{\boxed{\sigma(z_1^2)}} \frac{\partial}{\partial a_1^1} \overset{z_1^2}{\boxed{(w_1^2\, a_1^1)}} \frac{\partial}{\partial z_1^1} \overset{a_1^1}{\boxed{\sigma(z_1^1)}} \frac{\partial}{\partial w_1^1} \overset{z_1^1}{\boxed{(w_1^1 \cdot x_1 + w_2^1 \cdot x_2)}}$$

$$= \frac{1}{n}(\hat{y}-y) \cdot \sigma(z_1^2)\left(1-\sigma(z_1^2)\right) w_1^2 \cdot \sigma(z_1^1)(1-\sigma(z_1^1))x_1$$

$$\widehat{w_1^1} = w_1^1 - \eta \cdot (\ldots\ldots)$$

單純使用連鎖律做計算，的確可以得到梯度的數值，
然而在越深層的網路中這樣的計算負荷就會急遽增加

$$\widehat{w_1^2} = w_1^2 - \eta \cdot \frac{\partial L}{\partial w_1^2}$$

$$\frac{\partial L}{\partial w_1^2} = \boxed{\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_1^2} \frac{\partial a_1^2}{\partial z_1^2}} \frac{\partial z_1^2}{\partial w_1^2}$$

$$\widehat{w_1^1} = w_1^1 - \eta \cdot \frac{\partial L}{\partial w_1^1}$$

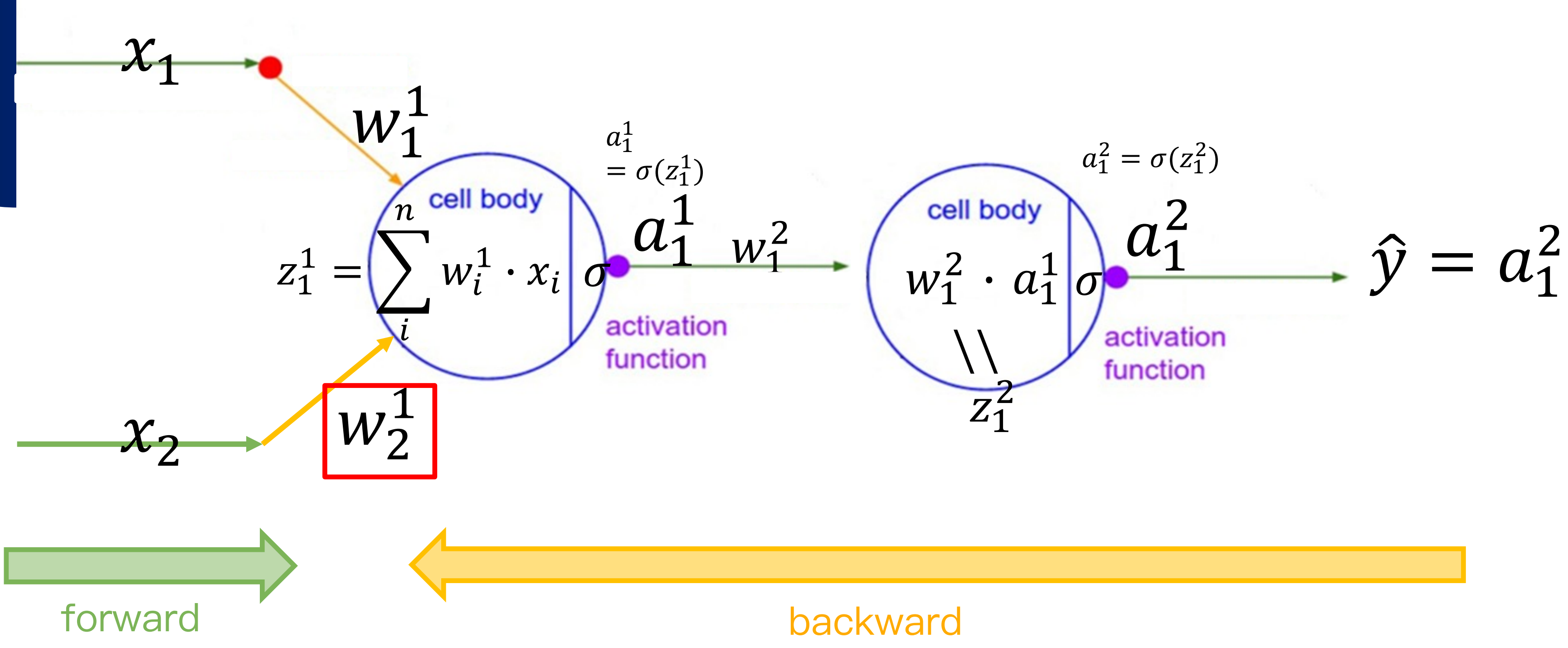
$$\frac{\partial L}{\partial w_1^1} = \boxed{\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_1^2} \frac{\partial a_1^2}{\partial z_1^2}} \frac{\partial z_1^2}{\partial a_1^1} \frac{\partial a_1^1}{\partial z_1^1} \frac{\partial z_1^1}{\partial w_1^1}$$

已被計算過的部分

$$\frac{\partial L}{\partial w_1^1} = \boxed{\frac{\partial}{\partial \hat{y}} \frac{1}{2n}(\hat{y} - y)^2 \frac{\partial}{\partial a_1^2} a_1^2 \frac{\partial}{\partial z_1^2} \sigma(z_1^2)} \frac{\partial}{\partial a_1^1}(w_1^2\, a_1^1) \frac{\partial}{\partial z_1^1} \sigma(z_1^1) \frac{\partial}{\partial w_1^1}(w_1^1 \cdot x_1 + w_2^1 \cdot x_2)$$

$$= \boxed{\frac{1}{n}(\hat{y} - y) \cdot \sigma(z_1^2)\left(1 - \sigma(z_1^2)\right)} w_1^2 \cdot \sigma(z_1^1)(1 - \sigma(z_1^1))x_1$$

在更新$w_1^2$時就已經計算到部分的梯度，因此不需重複計算，可直接使用已計算完的數值

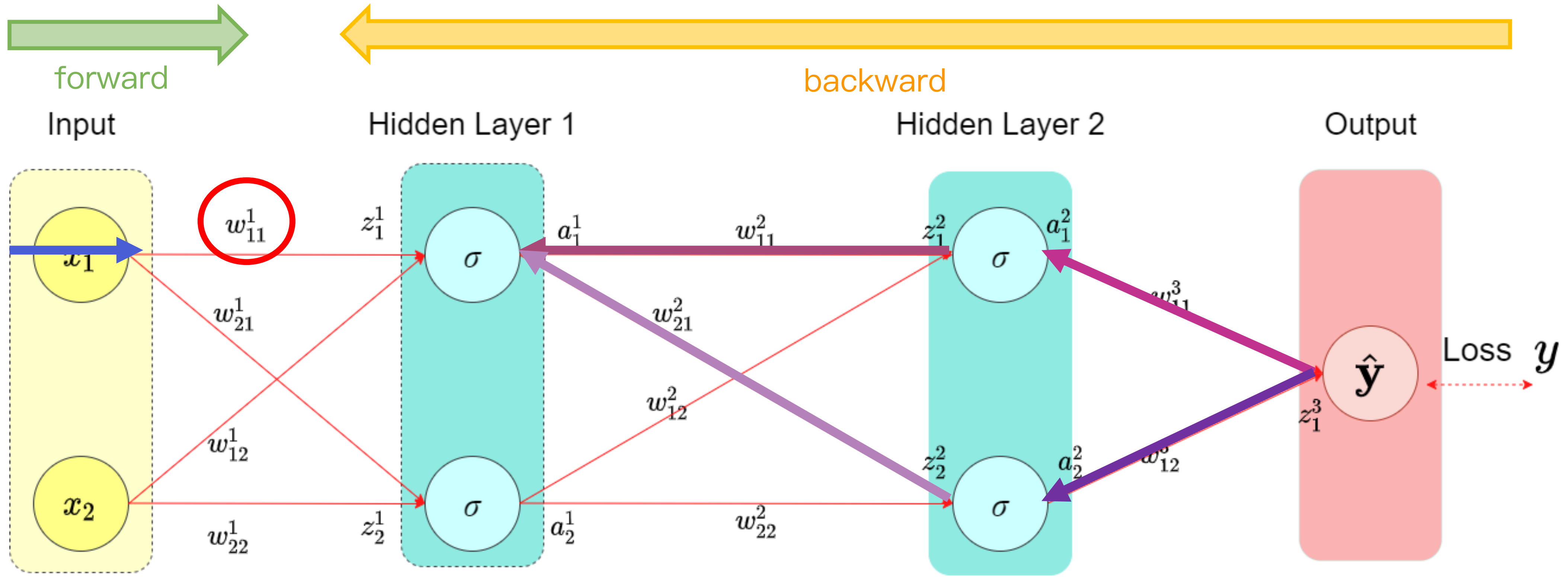



另外，同一個神經元的不同權重只有在forward部分不同，因此同樣可以不需重複計算

$$\widehat{w_1^1} = w_1^1 - \eta \cdot \frac{\partial L}{\partial w_1^1} \qquad \frac{\partial L}{\partial w_1^1} = \boxed{\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_1^2} \frac{\partial a_1^2}{\partial z_1^2} \frac{\partial z_1^2}{\partial a_1^1} \frac{\partial a_1^1}{\partial z_1^1}} \frac{\partial z_1^1}{\partial w_1^1}$$

$$\widehat{w_2^1} = w_2^1 - \eta \cdot \frac{\partial L}{\partial w_2^1} \qquad \frac{\partial L}{\partial w_2^1} = \boxed{\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_1^2} \frac{\partial a_1^2}{\partial z_1^2} \frac{\partial z_1^2}{\partial a_1^1} \frac{\partial a_1^1}{\partial z_1^1}} \frac{\partial z_1^1}{\partial w_2^1}$$

# 反向傳播 Back propagation



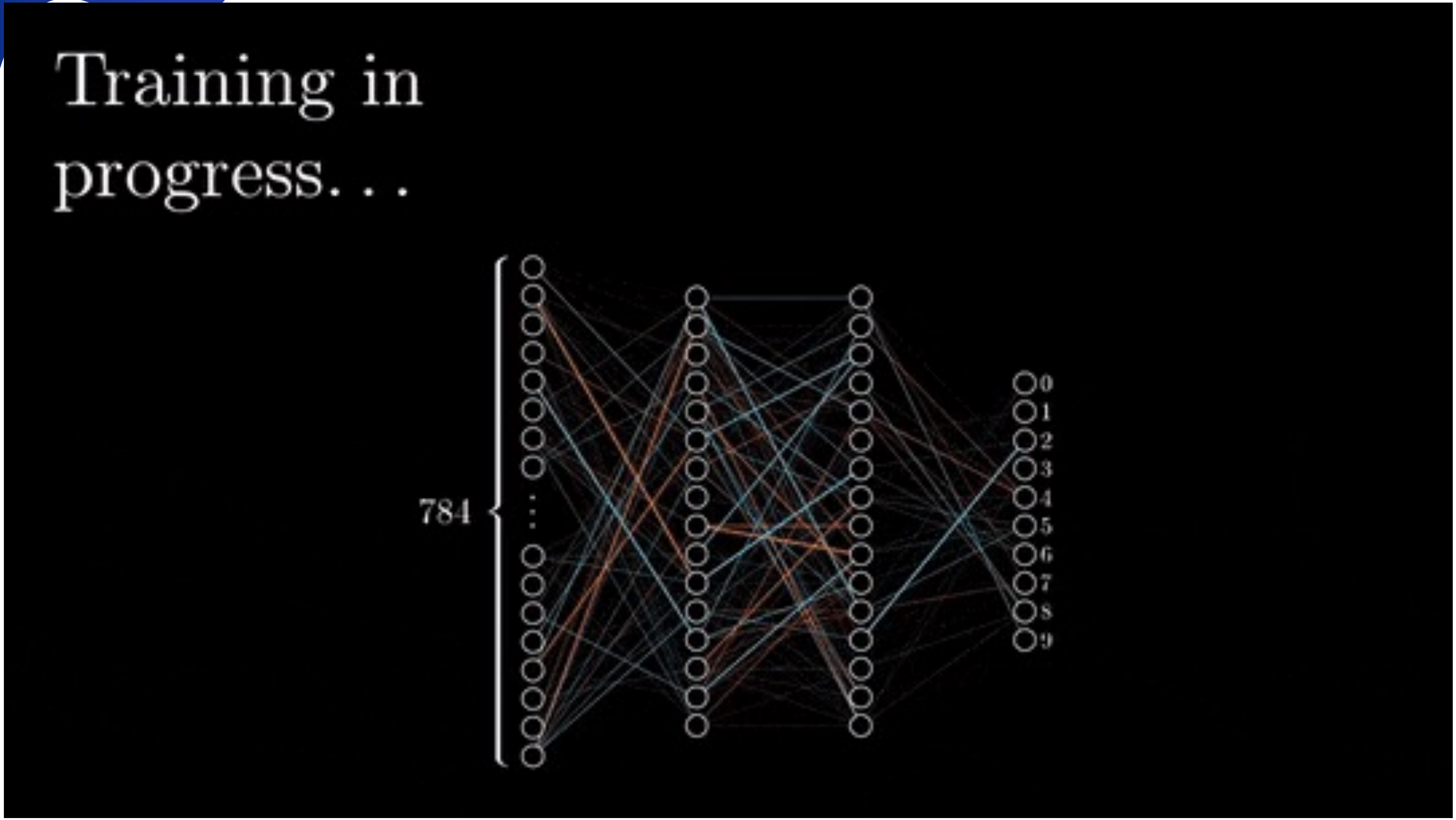

$$\widehat{w^1_{11}} = w^1_{11} - \eta \cdot \frac{\partial L}{\partial w^1_{11}} \qquad \frac{\partial L}{\partial w^1_{11}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^3_1} \left( \frac{\partial z^3_1}{\partial a^2_1} \frac{\partial a^2_1}{\partial z^2_1} \frac{\partial z^2_1}{\partial a^1_1} \quad \right) \frac{\partial a^1_1}{\partial w^1_{11}}$$

# 為什麼需要反向傳播？

利用反向傳播（Back Propagation），
我們可以用有效率的方式找到損失函數對於權重的梯度，
進而利用梯度下降法來最佳化每一個權重。

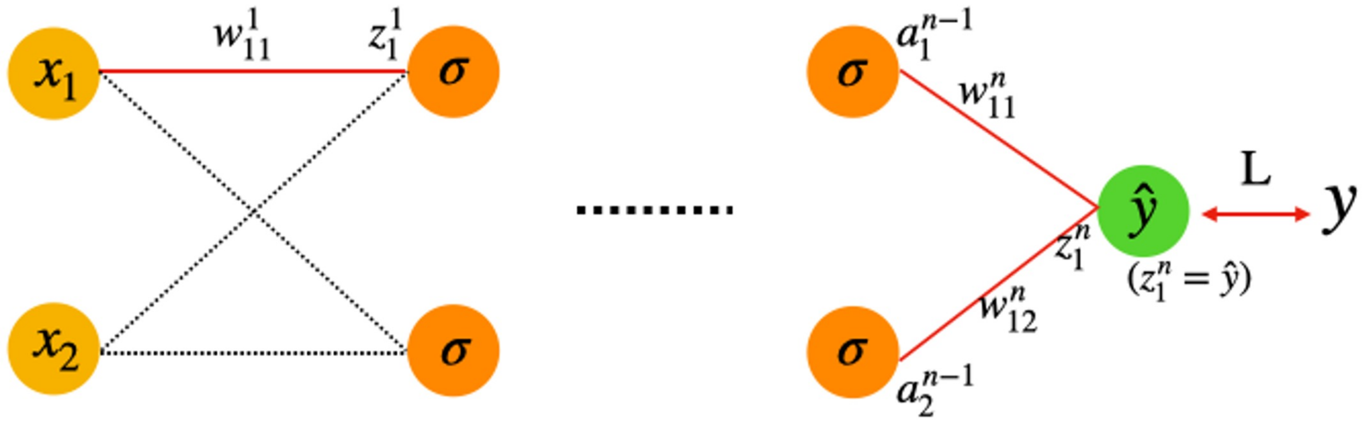


資料來源: Math.py




資料來源：https://storopoli.io/ciencia-de-dados/

# 梯度消失




$$\widehat{w_{11}^1} = w_{11}^1 - \eta \cdot \cancelto{0}{\frac{\partial L}{\partial w_{11}^1}}$$

- **3 layers**

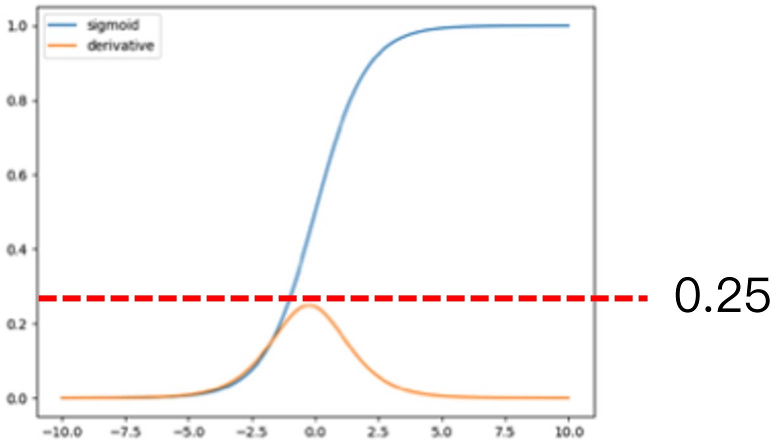
$$\frac{\partial L}{\partial w_{11}^1} = \frac{1}{n}(\hat{y} - y)[\sum_{i}^{2} w_{1i}^3 \sigma'(z_i^2) w_{i1}^2] \sigma'(z_1^1) x_1$$

- **n layers**

$$\frac{\partial L}{\partial w_{11}^1} = \frac{1}{n}(\hat{y} - y) w\underline{\sigma'(z)} w\underline{\sigma'(z)} w\underline{\sigma'(z)} \cdots x_1$$

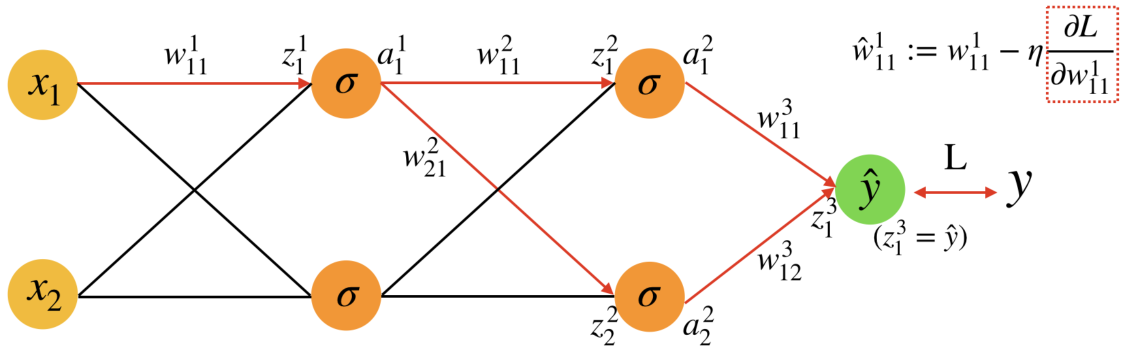
**Too many activation function**

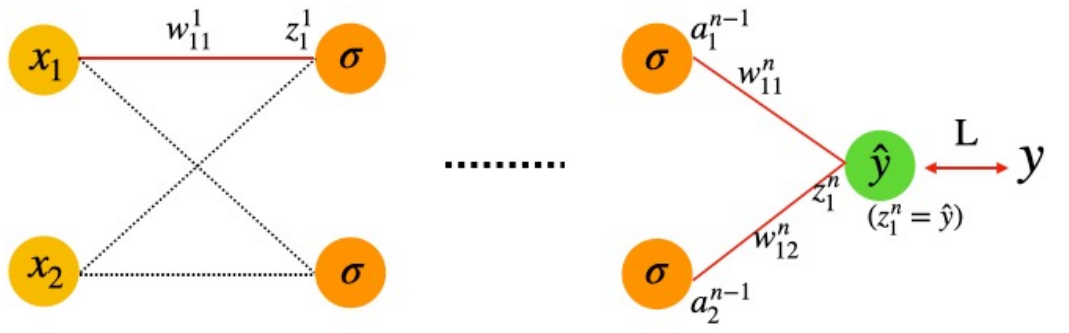
藍線：sigmoid
橘線：sigmoid函數的微分



# 梯度消失

影響$w_{11}^1$Gradient的因素:




3 Layers:

$$\frac{\partial L}{\partial w_{11}^1} = \frac{1}{n}(\hat{y} - y)[\textstyle\sum_{i=1}^2 w_{1i}^3 \underline{\sigma'(z_i^2)} w_{i1}^2]\underline{\sigma'(z_1^1)} x_1$$

$$0.25 \times 0.25 = 0.0625$$

n Layers:

$$\frac{\partial L}{\partial w_{11}^1} = \frac{1}{n}(\hat{y} - y) w \sigma'(z) w \sigma'(z) w \sigma'(z) \ldots x_1$$

Gradient vanishing
梯度消失

# 總結：如何避免梯度消失和梯度爆炸

- 如何避免梯度消失？
  - 使用別種激勵函數，例如ReLU、Mish、Swish
  - 降低層數
  - 使用其他輔助的神經網路層，例如 BatchNormalization
  - 嘗試使用機器學習模型，例如XGBoost，有時機器學習模型表現比 DNN 還要好

- 如何避免梯度爆炸？
  - 使用 Gradient clip 限制梯度的值域大小
  - 使用 Weight clip 限制每次更新後的權重大小

# 總結：如何設計合適的網路架構

- 網路該多深？
  - ➢ 一般來說，特徵數量以及資料量越多，層數會跟著加深，而訓練時間、記憶體用量也隨之增加。

- 每一層的神經元數量該怎麼設定？
  - ➢ 以全連結層 （DNN）為例：
    大原則是隨著層數堆疊，每一層的神經元數量會先變多再變少 ，
    例如：64 → 128 → 256 → 64 → 2。
    輸出層的神經元數量由任務類型決定，例如分類任務中的類別數量。

不同的資料集，其適合的網路架構都有所不同，在建模時除了參考類似任務的架構外，可以多嘗試以找到最合適者。