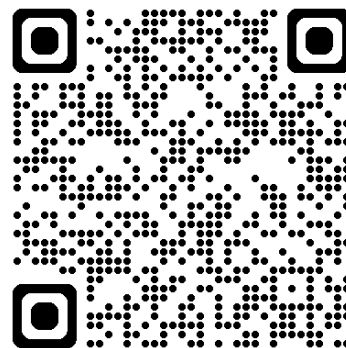




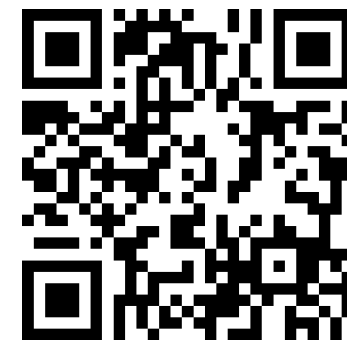
深度學習 Deep Learning

最佳化方法

Instructor: 林英嘉 (Ying-Jia Lin)
2025/09/24



[Course GitHub](#)



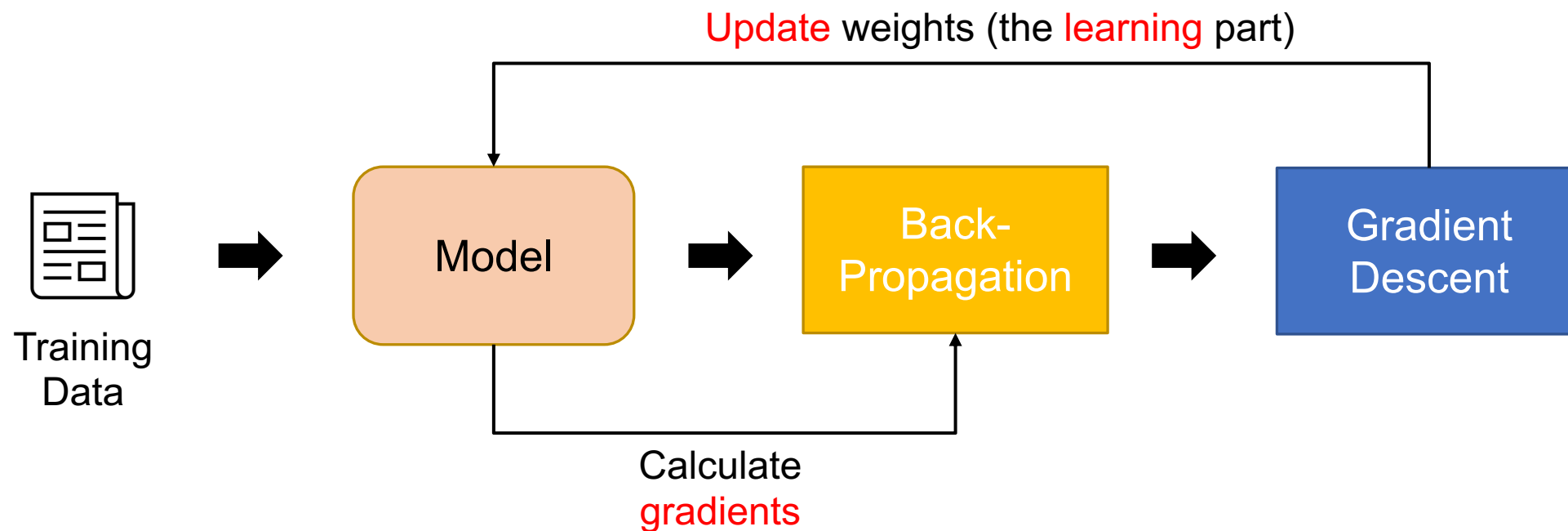
[Slido # DL0924](#)

Outline

- Recap [20 min]
- Gradient Descent (II) - Optimizers [60 min]
- Introduction to PyTorch [35 min]
- Quiz [30 min]



[Recap] 深度學習模型訓練流程



[Recap] Gradient Descent (梯度下降)

Assume x is a **trainable** parameter (**weight**), f is a **differentiable** function:

$$\text{Gradient descent: } x' = x - \eta \nabla_x f(x)$$

η is the **learning rate** (伊塔/欸塔) used for gradient descent.

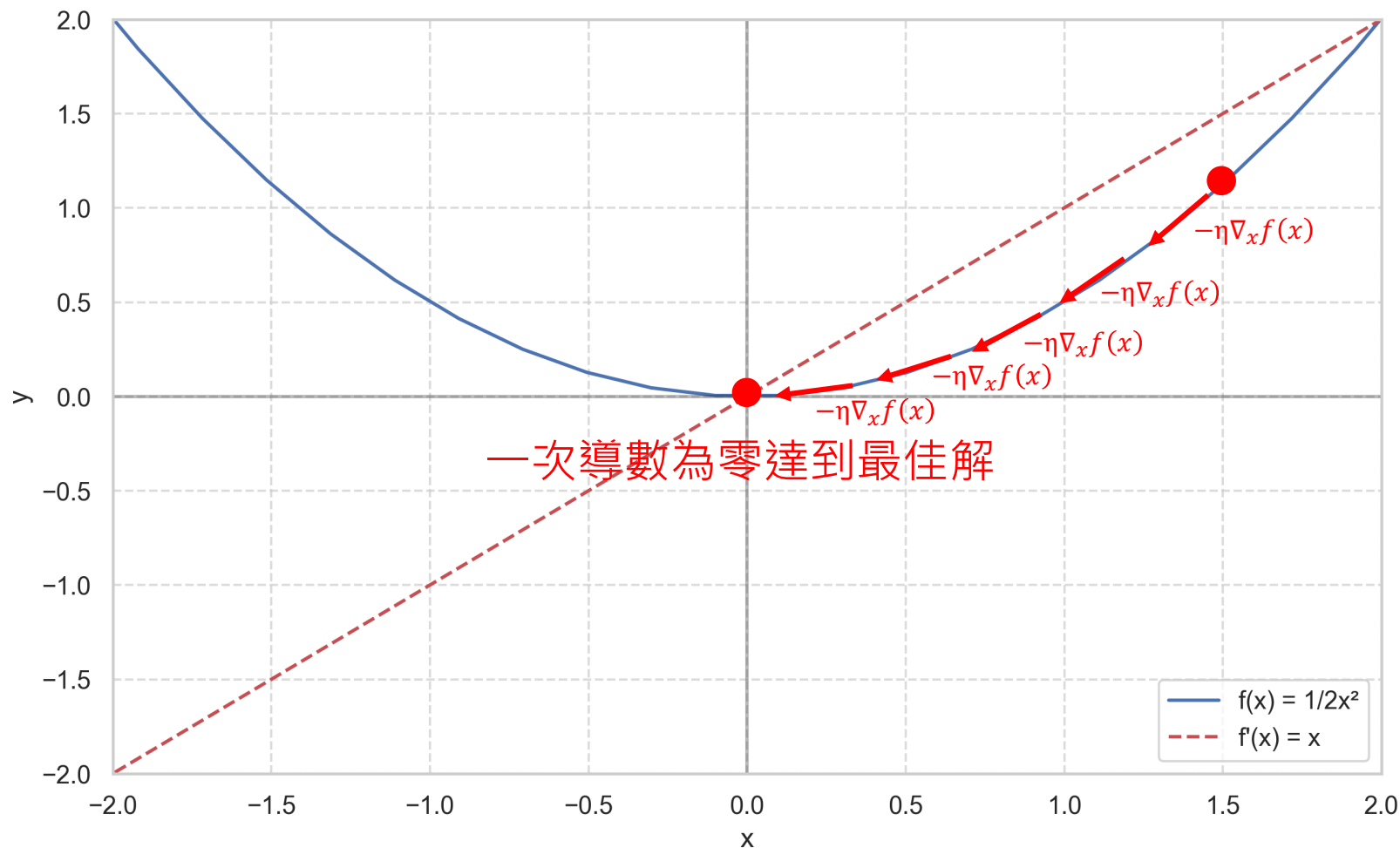
- 調整每次更新參數時的幅度



[Recap] 梯度下降簡易範例

梯度下降公式： $x' = x - \eta \nabla_x f(x)$

- 使用 $f(x) = 1/2x^2$ 作為範例 (x 同樣代表模型參數，為了方便理解，暫時不考慮輸入資料點)



[Recap] Minimize a Regression Model

- 假設我們今天要用 linear regression 來訓練一層的 MLP，模型輸出是 $\hat{y} = wx + b$
- 以均方誤差 (Mean Squared Error) 為例：

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad \leftarrow \text{模型輸出跟正確答案的平均差距}$$

把 $(wx_i + b)$ 代入 \hat{y}_i \longrightarrow
$$= \frac{1}{n} \sum_{i=1}^n (y_i - (wx_i + b))^2$$

- 其中：
 - 訓練目標是讓這個公式在 n 筆訓練資料的平均差距越小越好
 - \mathcal{L} 代表 Loss function ; n 代表有 n 筆訓練資料
 - y_i 為任一筆 ground-truth、 \hat{y}_i 為任一筆 prediction (model output)



[Recap] 利用梯度下降更新參數

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{1}{n} \sum_{i=1}^n 2(y_i - (wx_i + b)) \cdot (-x_i)$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{1}{n} \sum_{i=1}^n 2(y_i - (wx_i + b)) \cdot (-1)$$

更新w :

$$w_t = w_{t-1} - \eta \frac{\partial \mathcal{L}}{\partial w_{t-1}}$$

現在這個
時間點的
權重值

↑
↑
上一個的
時間點的
權重值

更新b :

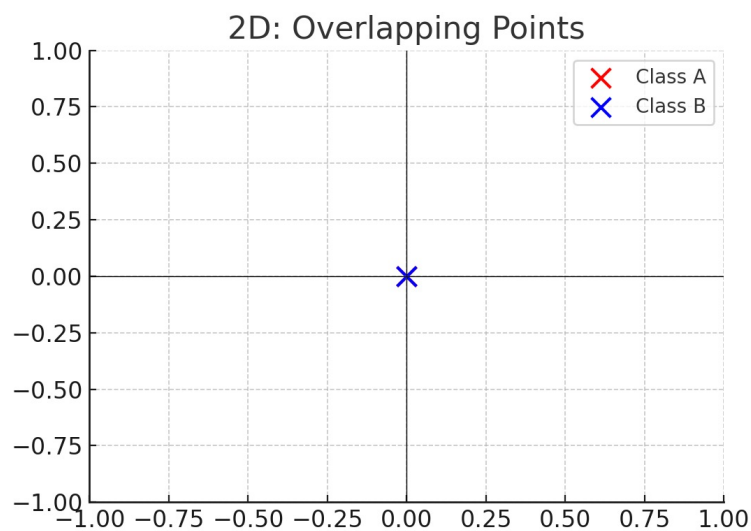
$$b_t = b_{t-1} - \eta \frac{\partial \mathcal{L}}{\partial b_{t-1}}$$

現在這個
時間點的
偏置項

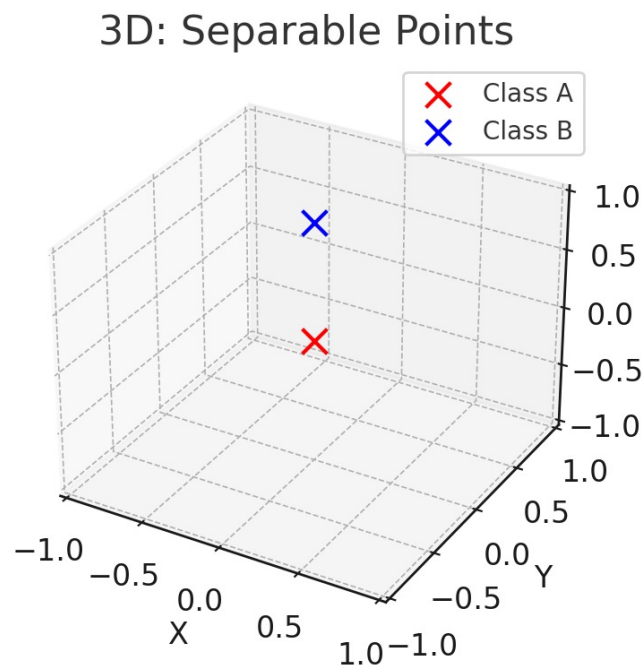
↑
↑
上一個的
時間點的
偏置項



[Recap] 深度學習是在創造非常複雜的空間



weight數量：2



weight數量：3

無限城



weight數量：非常多



Optimizers

基於梯度下降的最佳化方法

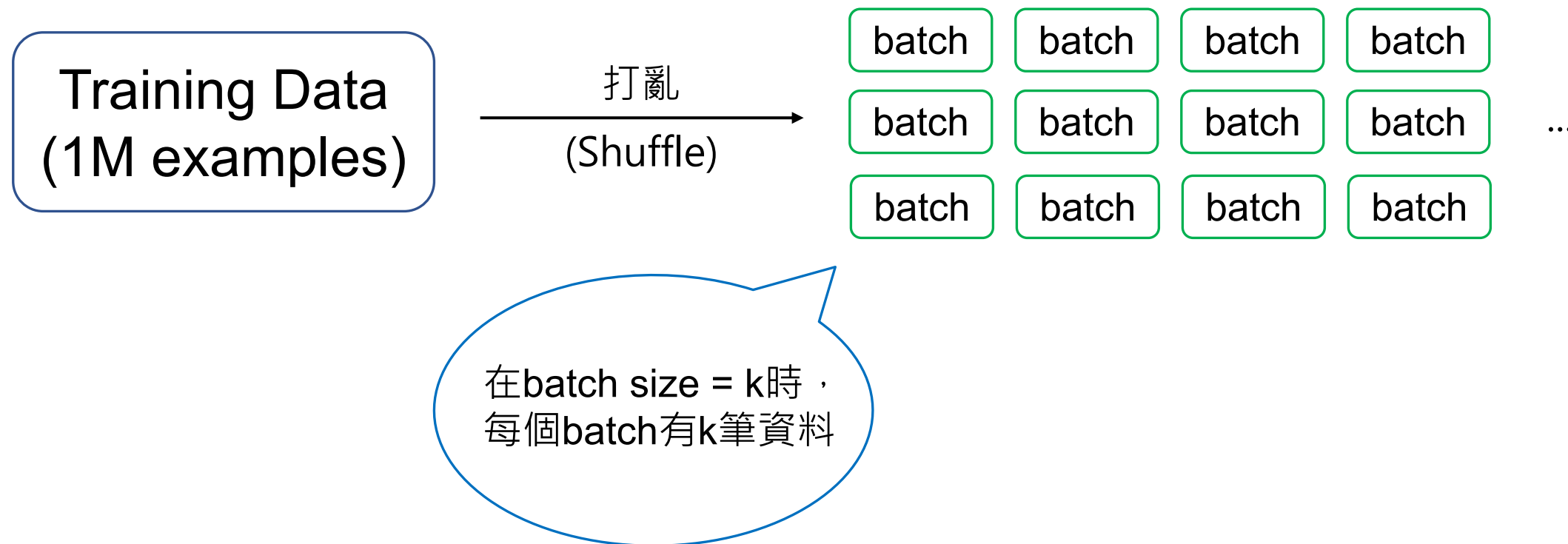
Another training approach: SGD

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

- 在一般的 Gradient descent (GD) 中，我們是將所有的資料算一次梯度之後，才更新一次模型
 - 資料量大的時候，單次模型更新的計算時間長 (但是平行化可以解決)
- **SGD: Stochastic gradient descent**
 - 把資料切成 batches，每次進行 GD 時都是隨機取其中一個 batch 來計算梯度與更新模型



Mini-batch Data



[定義] Epoch 與 Step

- 定義：
 - **Epoch** 是指模型已經看過整個訓練資料集一次的過程
 - **Step** 是指模型在訓練過程中，執行一次前向傳播與反向傳播 (一次更新權重) 的動作
 - 假設：
 - 我們訓練模型時的 batch_size 為 32，訓練資料集有 3,200 筆
 - 則：
 - 代表 1 個 epoch 有 100 個 steps



Optimization with Batches (Pseudo code)

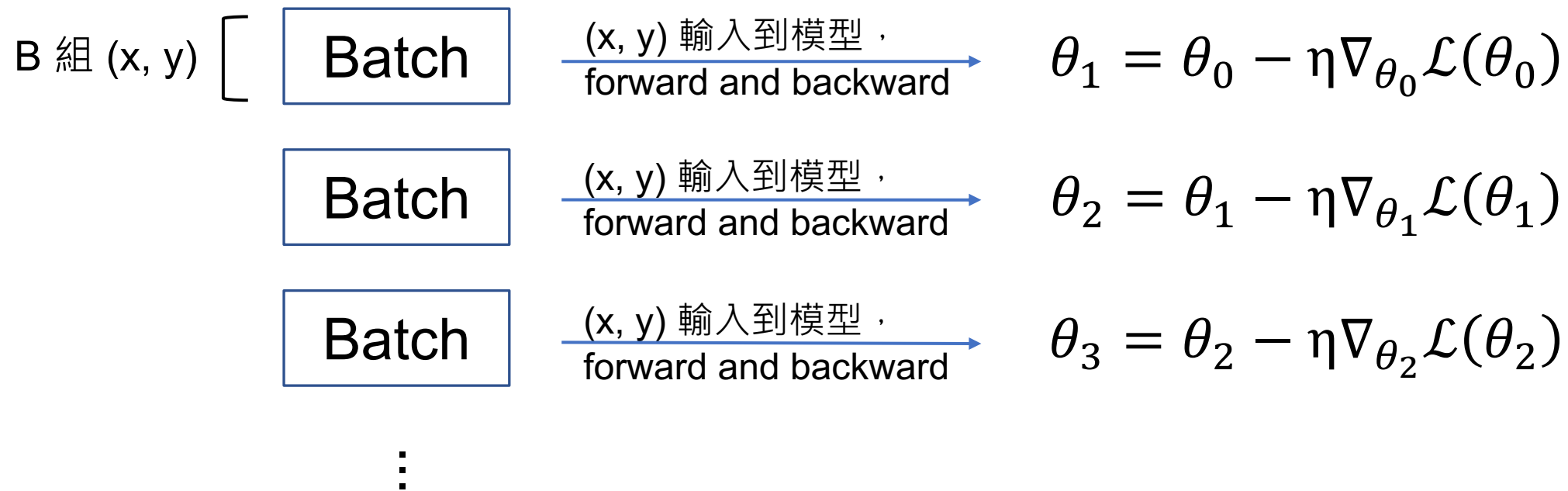
- 假設 training data 有 5,000,000 examples · batch_size = 1,000
 - Number of steps = $5000000 / 1000 = 5000$
 - 代表會進行 5000 次 gradient descent (更新 5000 次參數)

```
for i in range(5000):  
    start_index = i * 1000  
    end_index = start_index + 1000  
    x = train_x[start_index: end_index]  
    y = train_y[start_index: end_index]  
  
    optimizer.zero_grad()  
  
    # Calculate gradients via BP  
    outputs = model(x)  
    loss.backward()  
    # do gradient descent (update parameters)  
    optimizer.step()
```



Optimization with Batches

- 初始參數： θ_0



Gradient Descent vs. Stochastic Gradient Descent

這頁的 SGD 是指 **Mini-batch**
Stochastic Gradient Descent

- 計算 $\frac{\partial \mathcal{L}}{\partial w}$ 和 $\frac{\partial \mathcal{L}}{\partial b}$

Gradient Descent:

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^n (y_i - (wx_i + b))^2$$

全部的 n ，但如果 n 很大時需要計算久，且需要較大的記憶體

Stochastic
Gradient Descent:

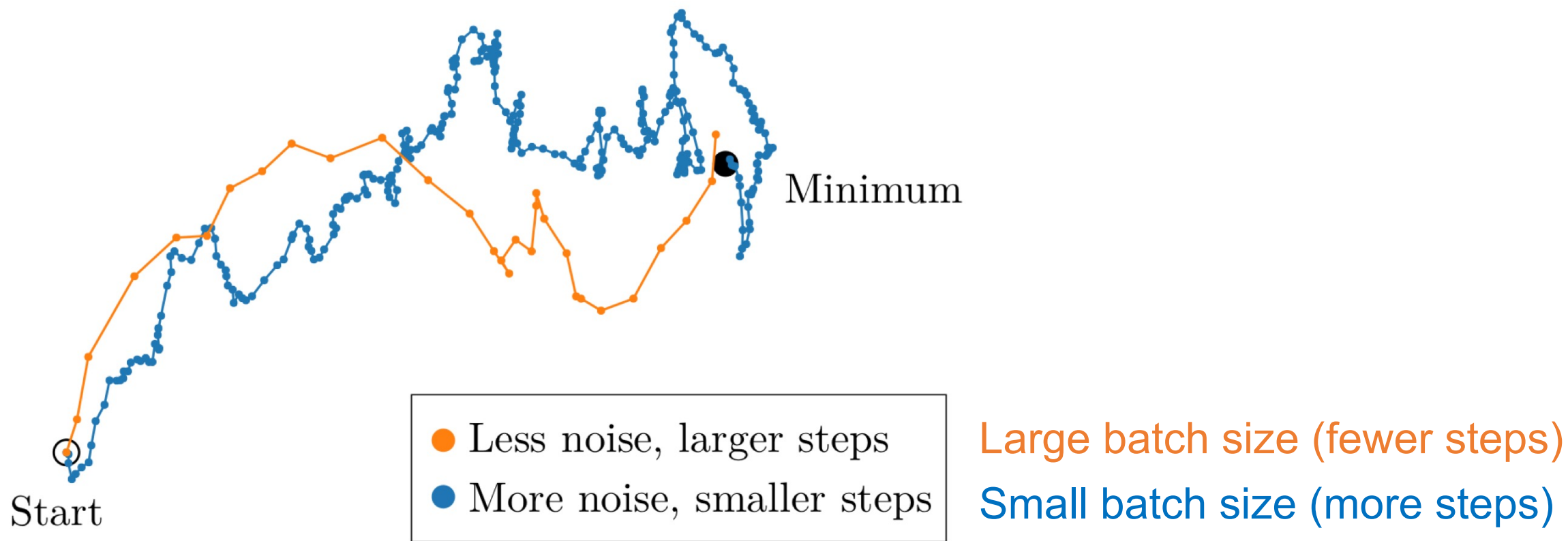
$$\mathcal{L} = \frac{1}{m} \sum_{i=1}^m (y_i - (wx_i + b))^2$$

m 改用採樣的 ($m \ll n$)，可能為 8 / 16 / 32 / 64 / 128

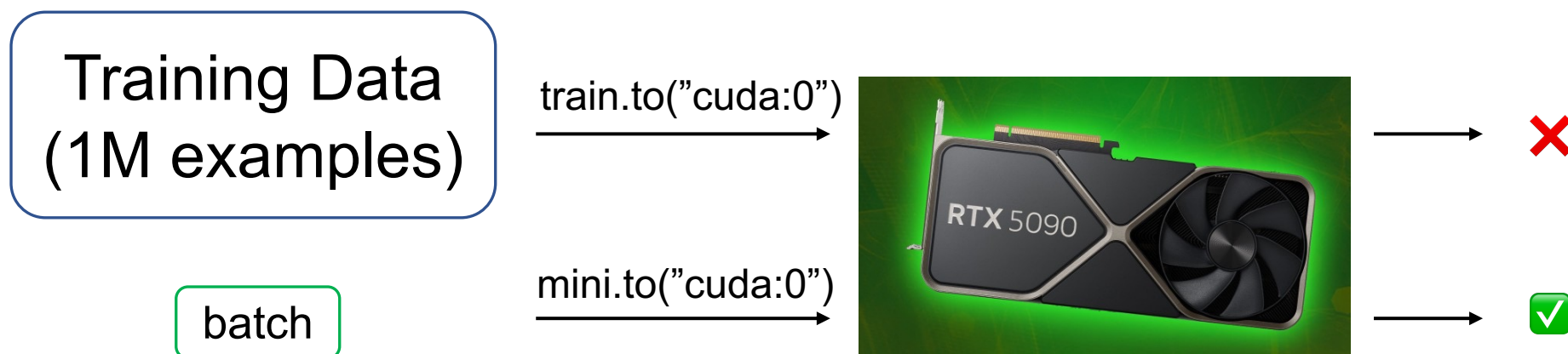
這裡的 m 就代表 **batch_size**



Full-batch 與 mini-batch 比較 #1：訓練穩定性



Full-batch 與 mini-batch 比較 #2：記憶體用量



- Typically, during training or test time, we use a small set of data (mini-batch) at one time for running a deep learning model on GPUs.



Question #1

- 和 full batch size 的 GD 比起來，用 mini-batch 方式訓練的效果一定比較好嗎？
- Ans: 對深度學習模型來說，通常比較好，因為 full batch size 的方法 1 個 epoch 只會更新模型一次，容易跑到 **local minimum** 的位置

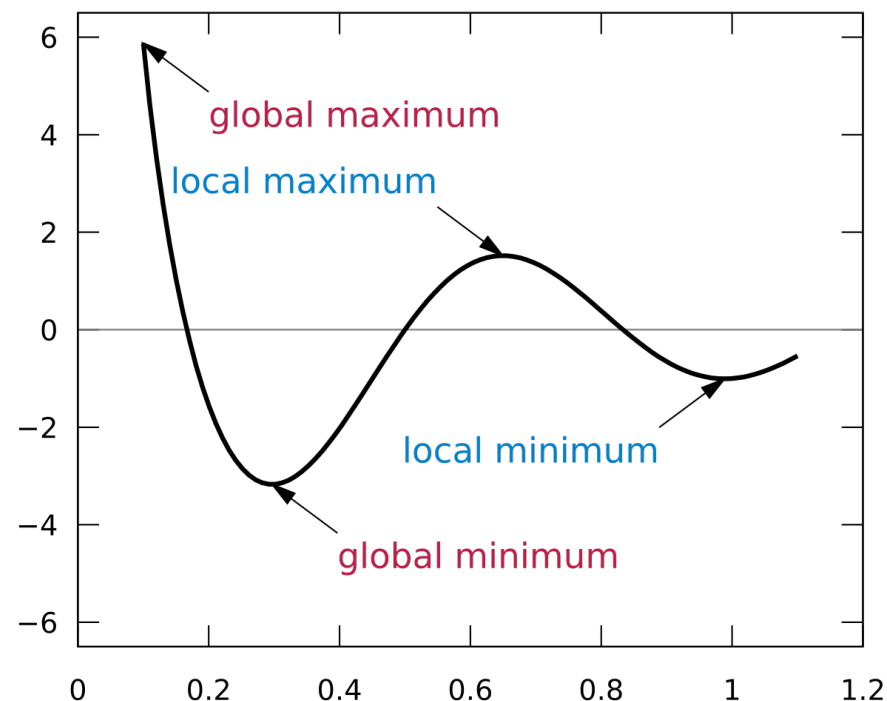


Figure source:
https://en.wikipedia.org/wiki/Maximum_and_minimum



Full-batch and mini-batch GD 比較表格

本堂課講的 SGD

	Batch Gradient Descent	Stochastic Gradient Descent	Mini-batch Stochastic Gradient Descent
單次更新所使用的訓練資料筆數	Entire training set	1	mini_batch_size (Hyperparameter)
訓練穩定性	Stable	Low	Medium
訓練時期更新頻率 (number of steps)	Low 1 time per epoch	High 1 time per sample	Medium 1 time per mini-batch
缺點	Easily falling into local minimum	Training variance is too big	Hard to determine the best batch size

廣義上 SGD 指的是 Mini-batch Stochastic Gradient Descent



Question #2

- batch_size 大比較好還是 batch_size 小比較好？

SB: small batch (256)

LB: large batch (training set size*0.1)

Table 2: Performance of small-batch (SB) and large-batch (LB) variants of ADAM on the 6 networks listed in Table 1

Name	Training Accuracy		Testing Accuracy	
	SB	LB	SB	LB
F_1	99.66% \pm 0.05%	99.92% \pm 0.01%	98.03% \pm 0.07%	97.81% \pm 0.07%
F_2	99.99% \pm 0.03%	98.35% \pm 2.08%	64.02% \pm 0.2%	59.45% \pm 1.05%
C_1	99.89% \pm 0.02%	99.66% \pm 0.2%	80.04% \pm 0.12%	77.26% \pm 0.42%
C_2	99.99% \pm 0.04%	99.99% \pm 0.01%	89.24% \pm 0.12%	87.26% \pm 0.07%
C_3	99.56% \pm 0.44%	99.88% \pm 0.30%	49.58% \pm 0.39%	46.45% \pm 0.43%
C_4	99.10% \pm 1.23%	99.57% \pm 1.84%	63.08% \pm 0.5%	57.81% \pm 0.17%



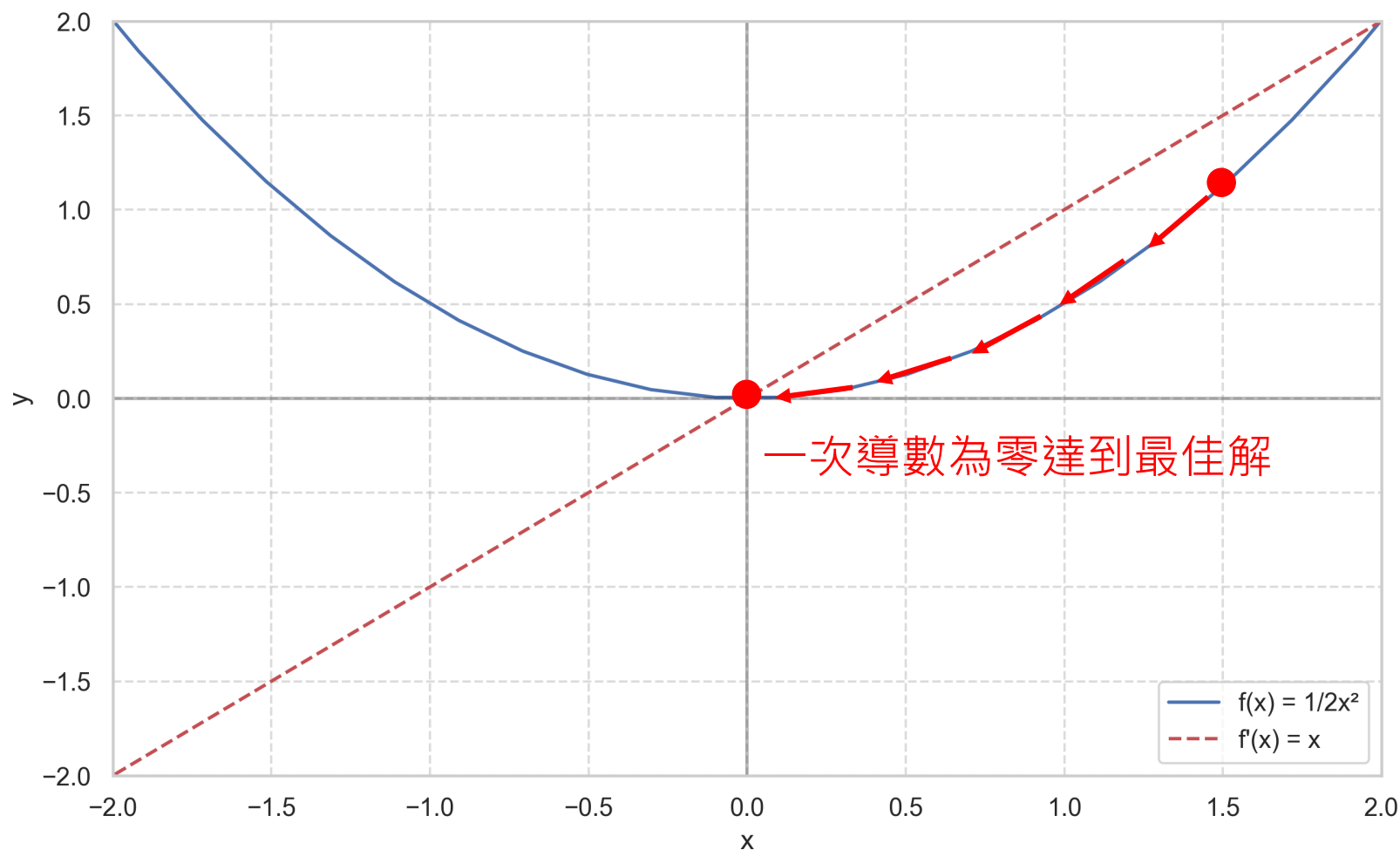
Small-batch vs. Large-batch Training

	Small batches	Large batches
單次更新所需要的時間	較短	較長 (平行化後差異縮小)
單次更新記憶體用量	較少	較多
訓練穩定性	較不穩定	較穩定
訓練時期更新頻率 (number of steps)	較多	較少
1 Epoch 訓練時間	較快	較慢 (平行化後較快)
訓練後的模型效能	可能較好	可能較差

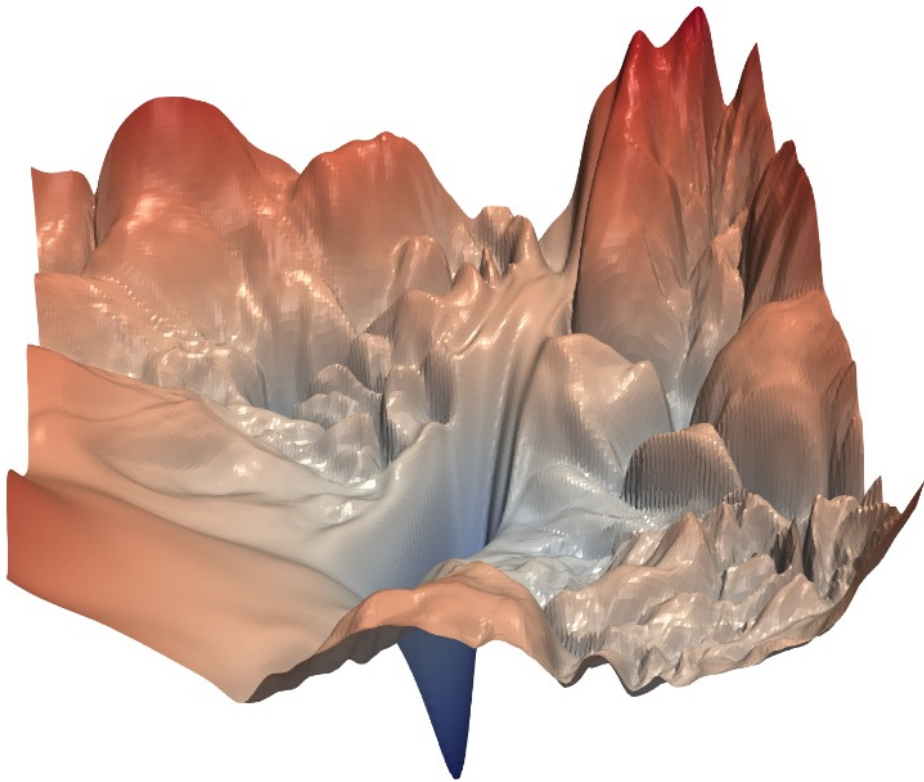


Loss function 到底長怎樣？

- 使用 $f(x) = 1/2x^2$ 作為簡單範例



Even more complicated ...



← The loss surfaces of ResNet-56 with/without skip connections.

<https://arxiv.org/abs/1712.09913>
Li, Hao, et al. "Visualizing the loss landscape of neural nets."
Advances in neural information processing systems 31 (2018).



SGD Problem #1 (紅色代表解決策略)

- SGD 在訓練過程中是使用隨機的 mini-batch 進行最佳化
- 如果隨機取到的 mini-batch 突然產生很大的梯度，可能使訓練不穩定

Gradient descent: $x_{t+1} = x_t - \eta \nabla_x f(x_t)$

訓練不穩定代表 x_{t+1} 在不該
改變很多的情況下改變太多

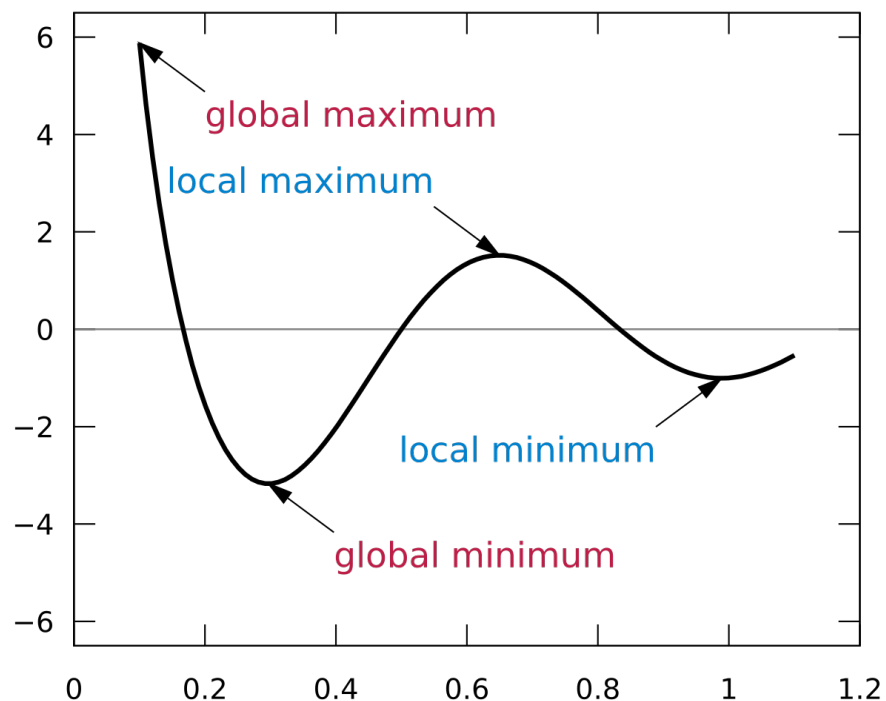
觀察梯度歷史紀錄：過去的時間點如果梯度小的話，現在這個時間點理論上梯度不該太大

調整學習率，該大的時候大一點，該小的時候小一點

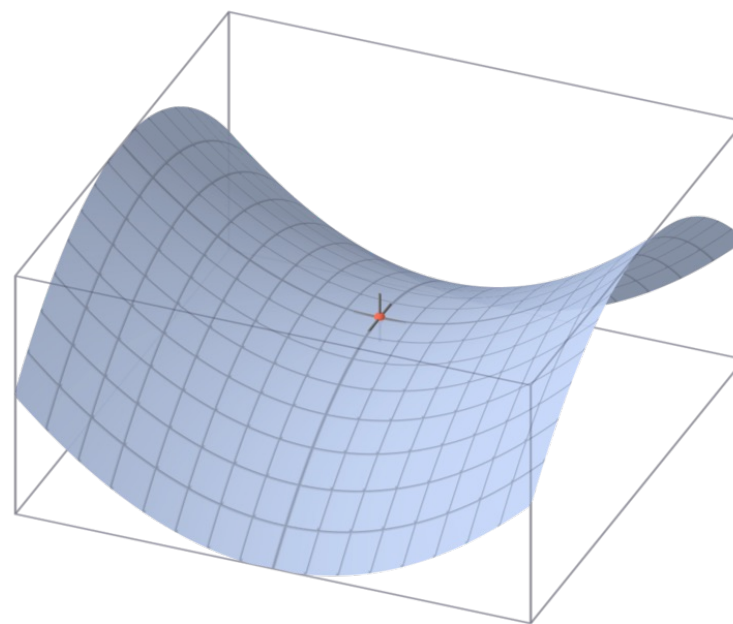


SGD Problem #2 : 容易卡住在梯度小的點

1. Local maximum / minimum



2. Saddle Point



Saddle Point
(鞍點)

Figure source:

https://en.wikipedia.org/wiki/Maximum_and_minimum

https://en.wikipedia.org/wiki/Saddle_point



如何有效訓練深度學習模型？

Optimizer:

- 對梯度動手腳
 - Momentum
 - ~~Nesterov Momentum~~(不常用)
- 自動調整學習率
 - Adagrad
 - RMSprop
 - Adam



Momentum

https://pytorch.org/docs/main/_modules/torch/optim/sgd.html#SGD

- Momentum 將過去的梯度記錄下來
- 每次更新參數時，會考慮到上一個時間點的梯度
 - 同方向 -> 參數改變幅度大一點；不同方向 -> 參數改變幅度小一點

Gradient descent: $x_{t+1} = x_t - \eta \nabla_x f(x_t)$

Momentum:

$$v_t = \gamma v_{t-1} + \eta \nabla_x f(x_t)$$

上一個時間點的梯度

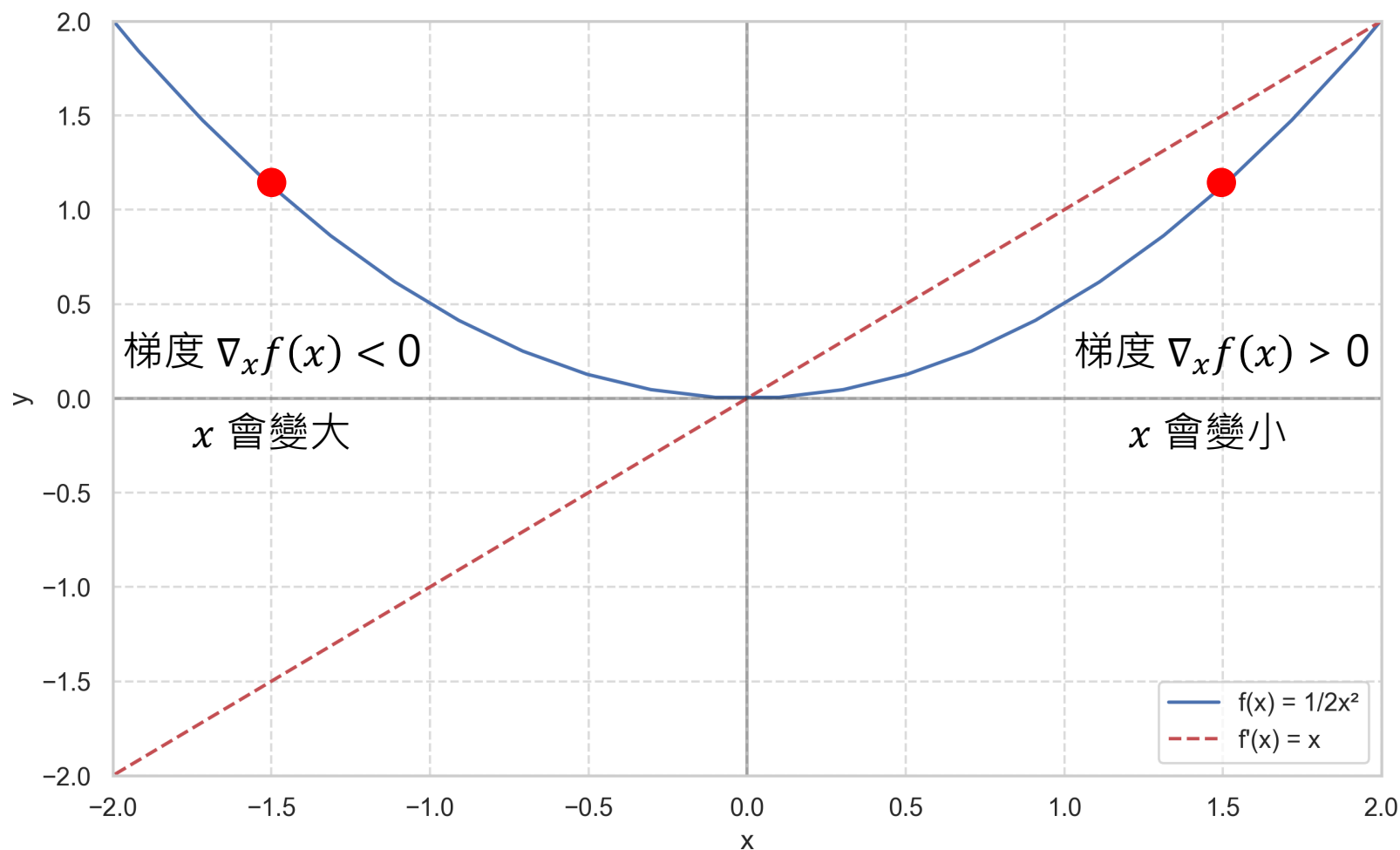
Momentum term
(超參數，常設為 0.9)

$$x_{t+1} = x_t - v_t$$



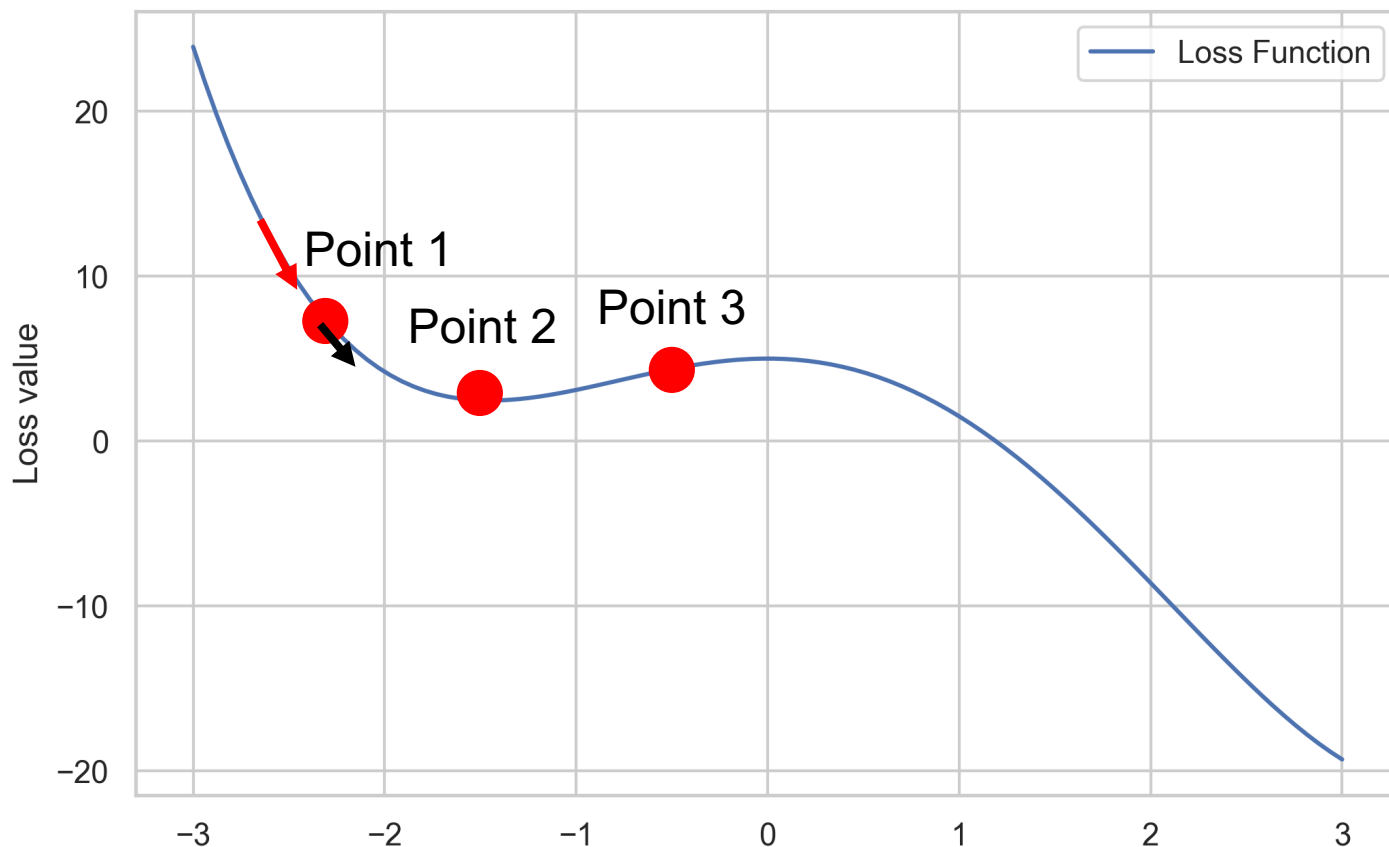
[Recap] 以兩個點來觀察 Gradient Descent 的特性

$$x' = x - \eta \nabla_x f(x)$$



SGD + Momentum (example)

(Point 1 -> Point 2 -> Point 3)



SGD + Momentum:

$$v_t = \gamma v_{t-1} + \eta \nabla_x f(x_t)$$

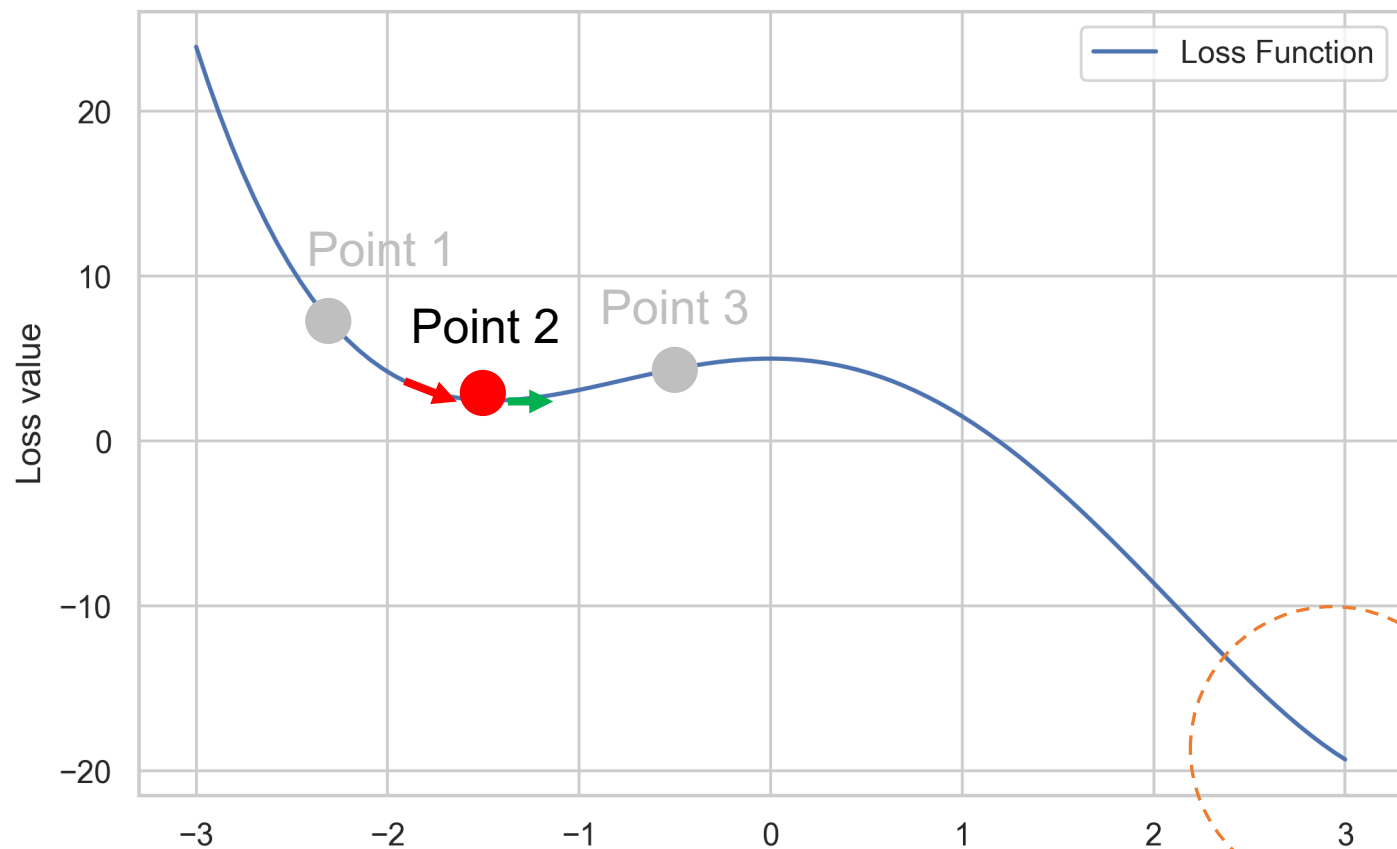
$$x_{t+1} = x_t - v_t$$

	P1
Gradient	-3.5
Past (v_{t-1})	-4
x 改變量 (GD)	→
x 改變量 (Momentum)	→
x 總改變量	→



SGD + Momentum (example)

(Point 1 -> Point 2 -> Point 3)



SGD + Momentum:

$$v_t = \gamma v_{t-1} + \eta \nabla_x f(x_t)$$

$$x_{t+1} = x_t - v_t$$

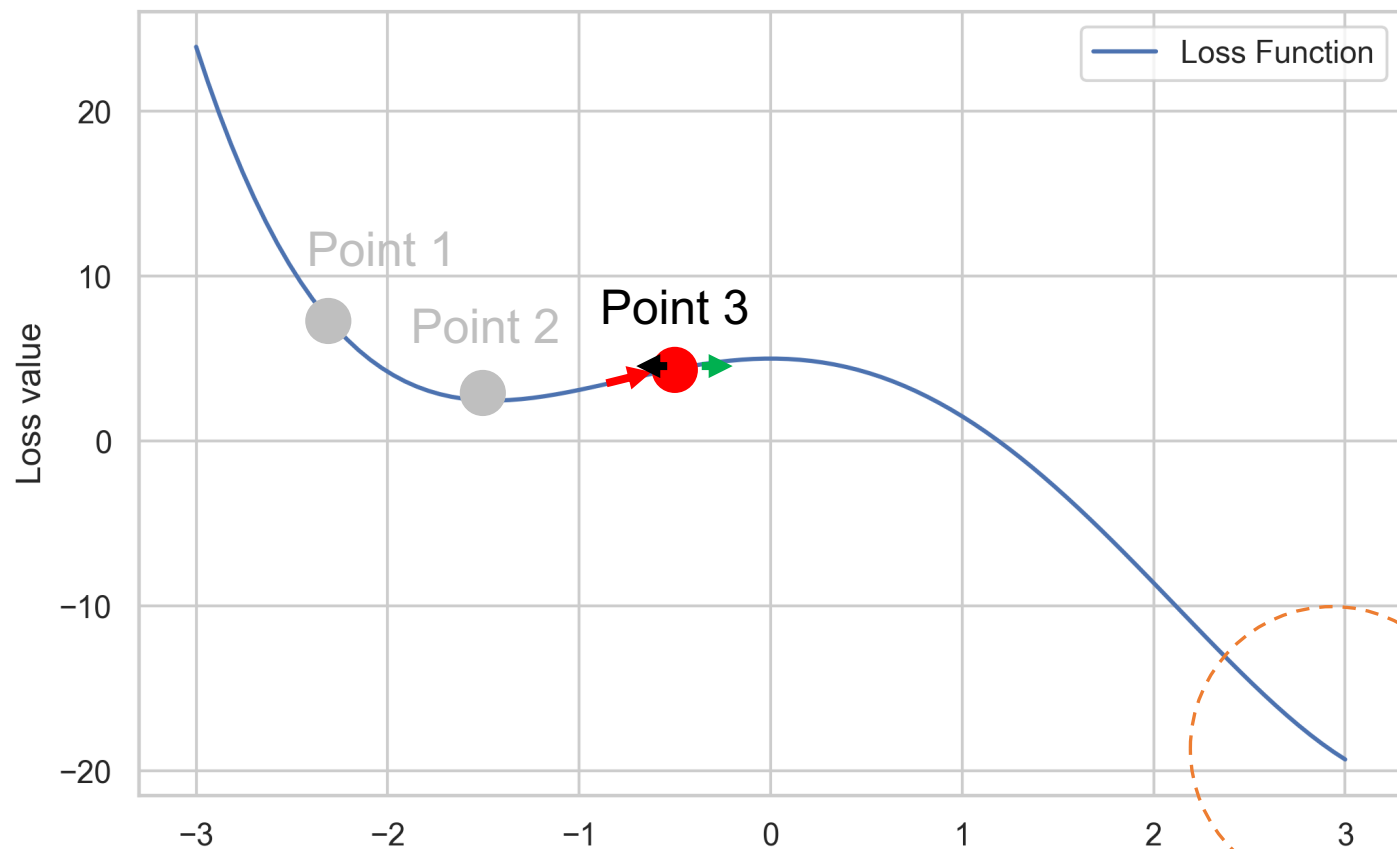
	P1	P2
Gradient	-3.5	0
Past (v_{t-1})	-4	-2
x 改變量 (GD)	→	
x 改變量 (Momentum)	→	→
x 總改變量	→	→



最低點

SGD + Momentum (example)

(Point 1 -> Point 2 -> Point 3)



SGD + Momentum:

$$v_t = \gamma v_{t-1} + \eta \nabla_x f(x_t)$$

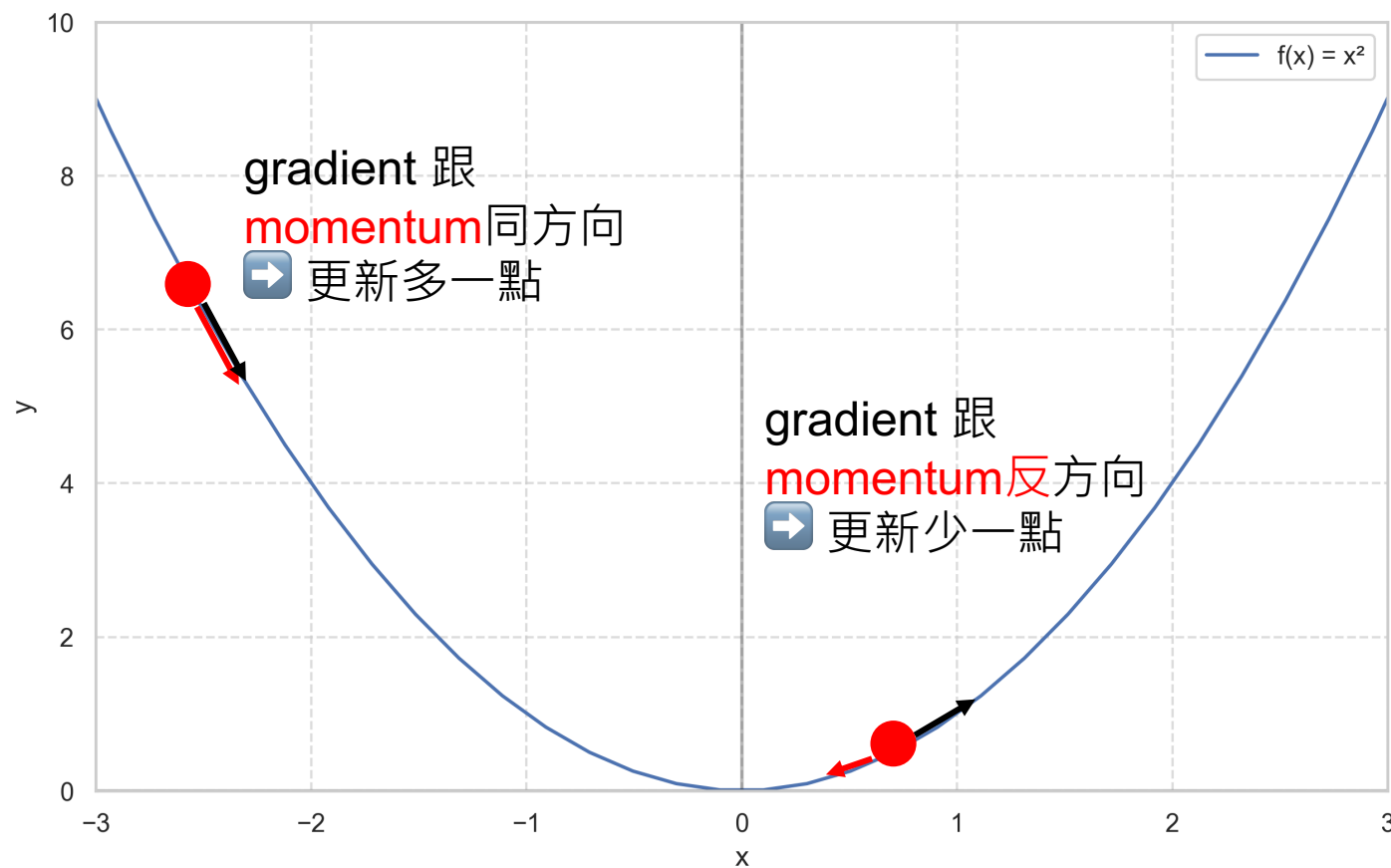
$$x_{t+1} = x_t - v_t$$

	P1	P2	P3
Gradient	-3.5	0	1
Past (v_{t-1})	-4	-2	-2
x 改變量 (GD)	→		←
x 改變量 (Momentum)	→	→	→
x 總改變量	→	→	→

最低點



Momentum



Summary of Momentum

- SGD 可能因為隨機採樣而梯度大幅震盪，而 Momentum 可以緩衝這種震盪
- Momentum 增加歷史紀錄的功能可以：
 - 加速收斂 -> 解決 SGD Problem #1
 - 逃離鞍點 (Saddle point) -> 解決 SGD Problem #2
 - 但 Momentum 也可能超過 convergence

只是理論，不一定能夠做到！(Deep Learning 是很複雜的 function)



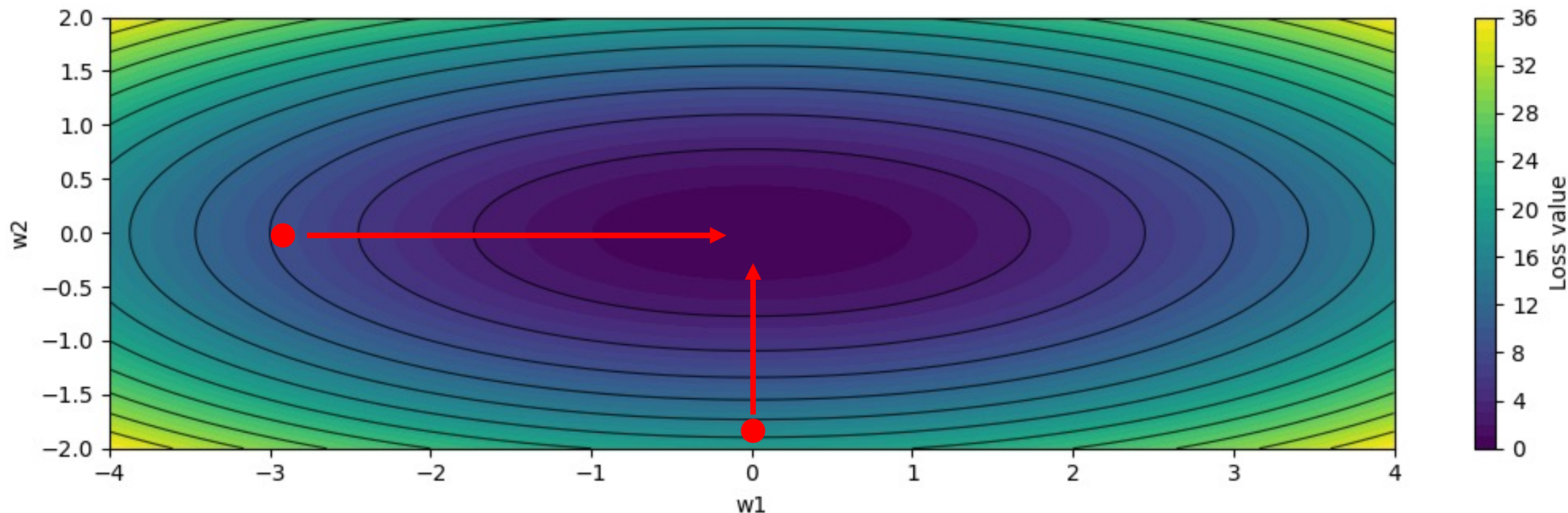
如何有效訓練深度學習模型？

Optimizer:

- 對梯度動手腳
 - Momentum
 - Nesterov Momentum
- 自動調整學習率
 - Adagrad
 - RMSprop
 - Adam



Learning rate 應該依據不同參數而不同



w_1 方向的梯度通常較大 (從 loss function 的橢圓形可觀察到)

w_2 方向的梯度通常較小



Adagrad (Adaptative Gradient)

Journal of Machine Learning Research 12 (2011) 2121-2159

Submitted 3/10; Revised 3/11; Published 7/11

Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*

John Duchi

*Computer Science Division
University of California, Berkeley
Berkeley, CA 94720 USA*

JDUCHI@CS.BERKELEY.EDU

Elad Hazan

*Technion - Israel Institute of Technology
Technion City
Haifa, 32000, Israel*

EHAZAN@IE.TECHNION.AC.IL

Yoram Singer

*Google
1600 Amphitheatre Parkway
Mountain View, CA 94043 USA*

SINGER@GOOGLE.COM

Editor: Tong Zhang

Duchi, John, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." Journal of machine learning research 12.7 (2011).



Adagrad (Adaptative Gradient)

- Adagrad 可以根據歷史梯度總和自動調整 **learning rate**
 - 需要一個 r 來記錄歷史梯度 (平方和)

Gradient descent:
$$x_{t+1} = x_t - \boxed{\eta} \nabla_x f(x_t)$$

Adagrad:
$$x_{t+1} = x_t - \boxed{\frac{\eta}{\delta + \sqrt{r_t}}} \nabla_x f(x_t)$$

$$r_t = r_{t-1} + g \odot g \quad \text{其中 } g = \nabla_x f(x_t)$$

- δ 是一個很小的數字，讓分母不要為0



Element-wise Scaling

- 在多維度情況 (有多個變數 x , Multivariate) 下，梯度是所有偏導數的向量：

$$\nabla_x f = \left[\frac{\partial f}{\partial x_1}, \quad \dots, \quad \frac{\partial f}{\partial x_n} \right]$$

↑ ↑ ↑

每個項的 g 都不同，進而能得出不同 r ，
最終每個項的 **learning rate** 都不同


$$x_{n,t+1} = x_{n,t} - \boxed{\frac{\eta}{\delta + \sqrt{r_{n,t}}}} \nabla_x f(x_{n,t})$$



Element-wise Multiplication

$$r_t = r_{t-1} + \boxed{g \odot g} \quad \text{其中 } g = \nabla_x f(x_t)$$

- 在多維度情況 (有多個變數 x , Multivariate) 下，梯度是所有偏導數的向量：

$$\nabla_x f = \left[\frac{\partial f}{\partial x_1}, \quad \dots, \quad \frac{\partial f}{\partial x_n} \right]$$


$$g \odot g = \left[\left(\frac{\partial f}{\partial x_1} \right)^2, \quad \dots, \quad \left(\frac{\partial f}{\partial x_n} \right)^2 \right]$$



對 Adagrad 的觀察與思考

Adagrad:

$$r_t = r_{t-1} + g \odot g$$

其中 $g = \nabla_x f(x_t)$

$$x_{t+1} = x_t - \frac{\eta}{\delta + \sqrt{r_t}} \nabla_x f(x_t)$$

- 特性：梯度大的學習率自動調小一點；梯度小的學習率自動調大一點
- 缺點：**Learning rate** 或許會下降太快 (因為 r_t 持續累積平方和)
 - 不適用於許多深度學習模型 (實驗角度)



RMSProp

- RMSProp (Root Mean Square Propagation) 是 Adagrad 的改版，為了避免 r 很快就變很大，使得學習率很快就變得很低 (此時神經網路會很難更新)
 - Hinton, Geoffrey, Nitish Srivastava, and Kevin Swersky. "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent." Cited on 14.8 (2012): 2.



RMSProp

Hinton, Geoffrey, Nitish Srivastava, and Kevin Swersky.
"Neural networks for machine learning lecture 6a overview
of mini-batch gradient descent." Cited on 14.8 (2012): 2.

Adagrad:

$$r_t = r_{t-1} + g \odot g$$

其中 $g = \nabla_x f(x_t)$

$$x_{t+1} = x_t - \frac{\eta}{\delta + \sqrt{r_t}} \nabla_x f(x_t)$$

Running average (moving average) · 依照比例抑制 $g \odot g$ 所造成 r 的增長

RMSProp:

$$r_t = \overset{\downarrow}{p} r_{t-1} + (\overset{\downarrow}{1} - \overset{\downarrow}{p}) g \odot g$$

p 為 decay rate (常設為 0.9)

$$x_{t+1} = x_t - \frac{\eta}{\sqrt{\delta + r_t}} \nabla_x f(x_t)$$

δ 是一個很小的數字，讓分母不要為0



Adam

- Adam: adaptive moment estimation
- 結合了 Momentum、Adagrad 和 RMSProp 的概念



Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." ICLR 2015. <https://arxiv.org/abs/1412.6980>

Adam

Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." ICLR 2015. <https://arxiv.org/abs/1412.6980>

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t .

Require: α : Stepsize **Learning rate**

Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates

Require: $f(\theta)$: Stochastic objective function with parameters θ

Require: θ_0 : Initial parameter vector

$m_0 \leftarrow 0$ (Initialize 1st moment vector)

$v_0 \leftarrow 0$ (Initialize 2nd moment vector)

$t \leftarrow 0$ (Initialize timestep)

while θ_t not converged **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters) **Adagrad**

end while

return θ_t (Resulting parameters)

Momentum

Adagrad, RMSProp

[Adam]

m 記錄梯度

v 記錄梯度平方

移動平均取自 **RMSProp**



訓練初期的問題

[Adam]
 m 記錄梯度
 v 記錄梯度平方

假設 $\beta_1 = 0.9$

Momentum歷史記錄

$$m_t = \underset{0}{\beta_1} m_{t-1} + (1 - \underset{0.1}{\beta_1}) \underset{0.5}{\nabla_x f(x_t)}$$

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)

(以 β_1^t 為例)

$$\beta_1^t = \beta_1^1 \times \beta_1^2 \times \cdots \times \beta_1^t$$

$\beta_1^t = 0.9$	無校正 (即 m_t)	校正後
\hat{m}_t	$0.1 * 0.5 = 0.05$	$0.1 * 0.5 / 0.1 = 0.5$



代表訓練初期參數更新幅度小 😞



訓練初期的問題 (t 次項)

[Adam]

m 記錄梯度

v 記錄梯度平方

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)

$$\beta_1^t = \beta_1^1 \times \beta_1^2 \times \cdots \times \beta_1^t$$

t	β_1^t	校正項 $1 - \beta_1^t$
1	0.9	0.1
2	$0.9^2 = 0.81$	0.19
3	$0.9^3 = 0.729$	0.271
10	$0.9^{10} = 0.3487$	0.6513
100	$0.9^{100} = 0.000026$	0.999974

$1 - \beta_1^t$ 越來越接近1

代表隨著訓練時間增加，
越來越不需要校正



比較參數更新

[Adam]

m 記錄梯度

v 記錄梯度平方

Adam:

$$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon) \text{ (Update parameters)}$$

其中 α 是學習率

Momentum:

$$v_t = \gamma v_{t-1} + \eta \nabla_x f(x_t)$$

其中 η 是學習率

$$x_{t+1} = x_t - v_t$$

Adagrad:

$$r_t = r_{t-1} + g \odot g$$

其中 $g = \nabla_x f(x_t)$

$$x_{t+1} = x_t - \frac{\eta}{\delta + \sqrt{r_t}} \nabla_x f(x_t)$$

再加上移動平均紀錄的概念，Adam 是 Momentum, RMSProp, Adagrad 的集大成





<https://x.com/DBahdanau/status/1916666861808456176>

AdamW will be in Week 7

The paper of AdamW:

Published as a conference paper at ICLR 2019

DECOUPLED WEIGHT DECAY REGULARIZATION

Ilya Loshchilov & Frank Hutter

University of Freiburg

Freiburg, Germany,

{ilya, fh}@cs.uni-freiburg.de

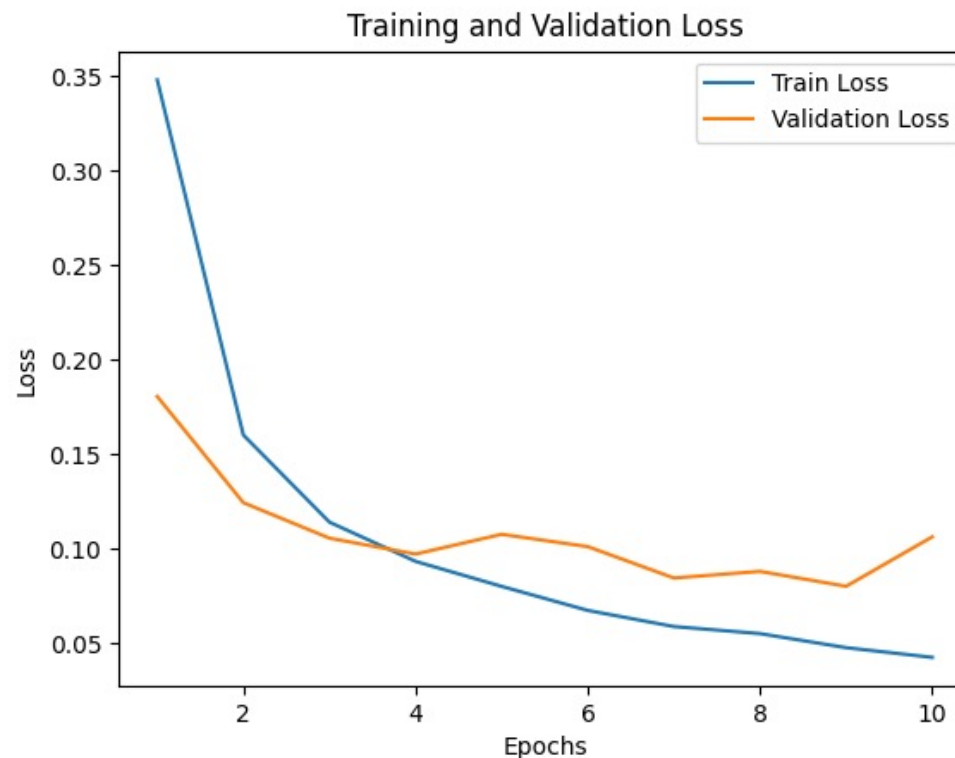
Week7: 過擬合、正規化、模型訓練技巧、期末專案介紹



訓練完成之後

Convergence

- Convergence (收斂) 是指在模型訓練過程中，loss 的變化逐漸趨於穩定，可能已經到達最佳解，表示模型的學習進度減緩
- 收斂表示模型可能已達到某個局部或全局最小值，但不一定代表它找到了最佳解 (因為有可能只是局部最小值)，也不保證模型此時具備良好的泛化能力 (val_loss 也同樣很低)



Convergence and gradients

- 在經過很多個 steps (t 很大) 的更新之後，gradients 等於零的情況：

$$\nabla_{\theta_t} \mathcal{L}(\theta_t) = 0$$

- \mathcal{L} : loss function
- θ_t : 在 t 個時間點 (step) 的的參數組合 (包含各層的w跟b)
- $\nabla_{\theta_t} \mathcal{L}(\theta_t)$: 梯度

此時代表模型可能已經達到目標函數的最佳解



Additional Links

- Optimizers
 - <https://www.ruder.io/optimizing-gradient-descent/#fn5>
- Saddle point
 - https://www.youtube.com/watch?v=8aAU4r_pUUU
- Deep Learning Book Chapter 8
 - <https://www.deeplearningbook.org/contents/optimization.html>



Thank you!

Instructor: 林英嘉

 yjlin@cgu.edu.tw

TA: 劉美辰

 m1461014@cgu.edu.tw