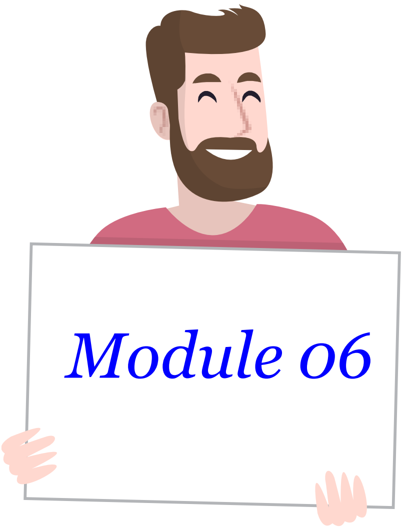
優化神經網路



Estimated time:

45 min.

學習目標

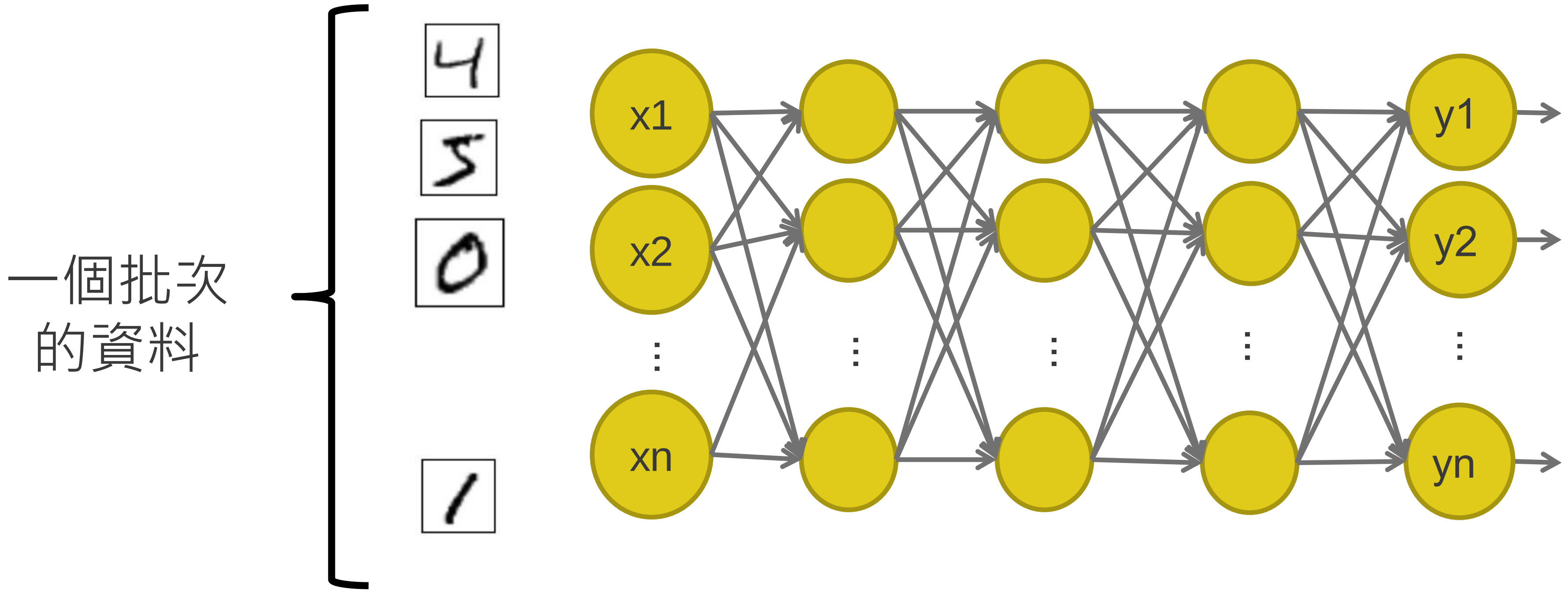
- **6-1: 批次輸入資料**
- **6-2: 優化器的概念**
- **6-3: 可適性學習率優化器**

6-1:批次輸入資料

- 批次輸入資料
- Epoch及Step
- 批次資料計算損失函數
- 優化流程圖

批次輸入資料

- 在實務上，我們常將資料一個批次一個批次輸入神經網路而非一筆一筆輸入網路
  - 批次大小使用者可以自己設定
  - 加快訓練速度



批次輸入資料

- 將資料批次輸入、一次輸入所有、一次一筆資料的比較如下：

輸入資料的方式	特性
一次輸入所有資料	記憶體可能會不夠
一次輸入一筆資料	計算要非常久 容易受單筆極端值資料影響優化品質
一次輸入一個批次的資料 (B < N)	介於上面兩者之間 根據電腦記憶體可以自行調整批次大小

Epoch及Step

- 在深度學習裡
  - 讓電腦把所有訓練資料輸入過一次叫做一個”Epoch”
  - 讓電腦把一個批次的資料輸入過一個叫做一個”Step”
  - 通常我們都會用Epoch來衡量網路訓練多久

批次資料計算損失函數

- 將資料批次輸入、一次輸入所有、一次一筆資料三種情況，機算損失函數如下
  - 計算時如果一次輸入資料不只一筆，損失函數記得取平均值

輸入資料的方式	MSE	Cross-Entropy
一次輸入所有資料	$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$	$\frac{1}{N}\sum_{j=1}^{N}(-\sum_{i=0}^{class\,\#}\hat{y}_i \ln(y_i))$
一次輸入一筆資料	$\sum_{i=1}^{class\,\#}(y_i - \hat{y}_i)^2$	$-\sum_{i=0}^{class\,\#}\hat{y}_i \ln(y_i)$
一次輸入一個批次的資料 (B < N)	$\frac{1}{B}\sum_{i=1}^{B}(y_i - \hat{y}_i)^2$	$\frac{1}{B}\sum_{j=1}^{B}(-\sum_{i=0}^{class\,\#}\hat{y}_i \ln(y_i))$

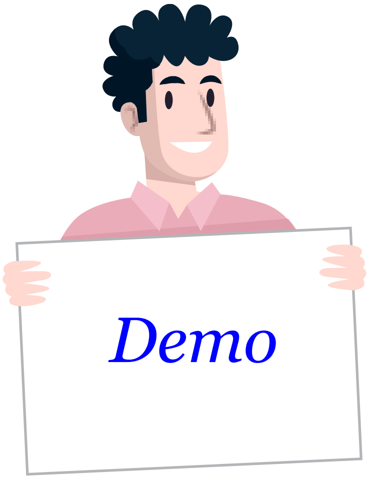
優化流程圖

- 神經網路優化的流程
  - 輸入第一批次資料 → 計算神經網路及損失函數 → 計算神經網路及損失函數→ 輸入第二批次資料 ......
  - 藉由這樣反覆的流程，我們最終可以找到一組不差的參數

輸入第一個批次的資料 → 計算神經網路及損失函數 → 輸入第二個批次的資料 → 計算神經網路及損失函數 ......

6-1 Demo

- 開啟Demo_6-1.ipynb
- 批次輸入實作

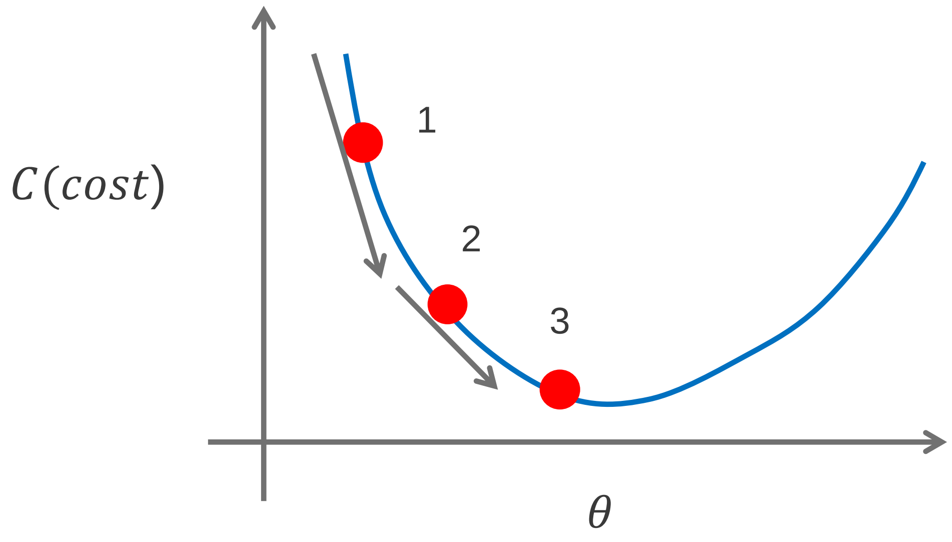


6-2:優化器的概念

- 梯度下降法
- 動量梯度下降法

梯度下降法

- 一種以數值化演算法，其藉由不斷迭代的方式可以找到一個函數的區域極小值
  - 需要給定一個函數並初始化起始點



隨機選取 θ_1 當起始點

計算 $\frac{dC(\theta_1)}{d\theta}$

$$\theta_2 \leftarrow \theta_1 - \eta \frac{dC(\theta_1)}{d\theta}$$

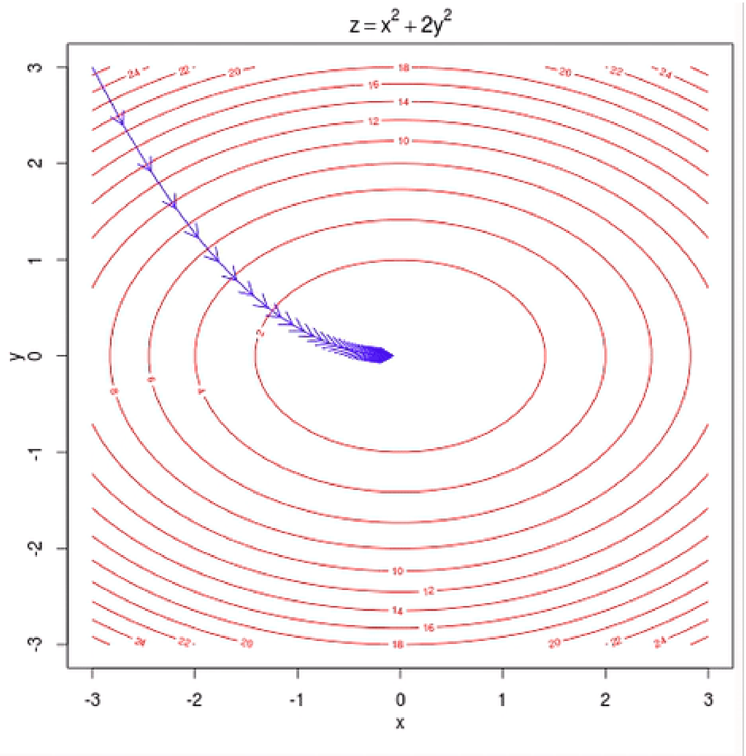
計算 $\frac{dC(\theta_2)}{d\theta}$

$$\theta_3 \leftarrow \theta_2 - \eta \frac{dC(\theta_2)}{d\theta}$$

學習率

梯度下降法

- 梯度下降法也適用在多個變數的函數上面，下圖為兩個變數的函數做梯度下降法
  - 實際上函數不管有多少變數，梯度下降法都可以使用



$$\theta = \begin{bmatrix} x \\ y \end{bmatrix} \quad \nabla C(\theta) = \begin{bmatrix} dz/dx \\ dz/dy \end{bmatrix}$$

隨機將 θ_1 當起始點

計算 $\nabla C(\theta_1)$

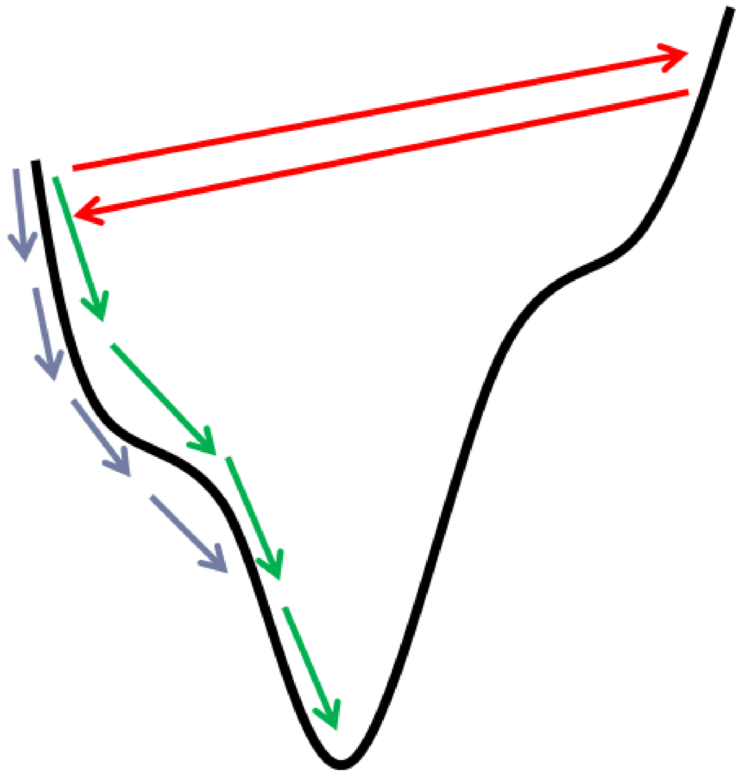
$$\theta_2 \leftarrow \theta_1 - \eta \nabla C(\theta_1)$$

計算 $\nabla C(\theta_2)$

$$\theta_3 \leftarrow \theta_2 - \eta \nabla C(\theta_2)$$

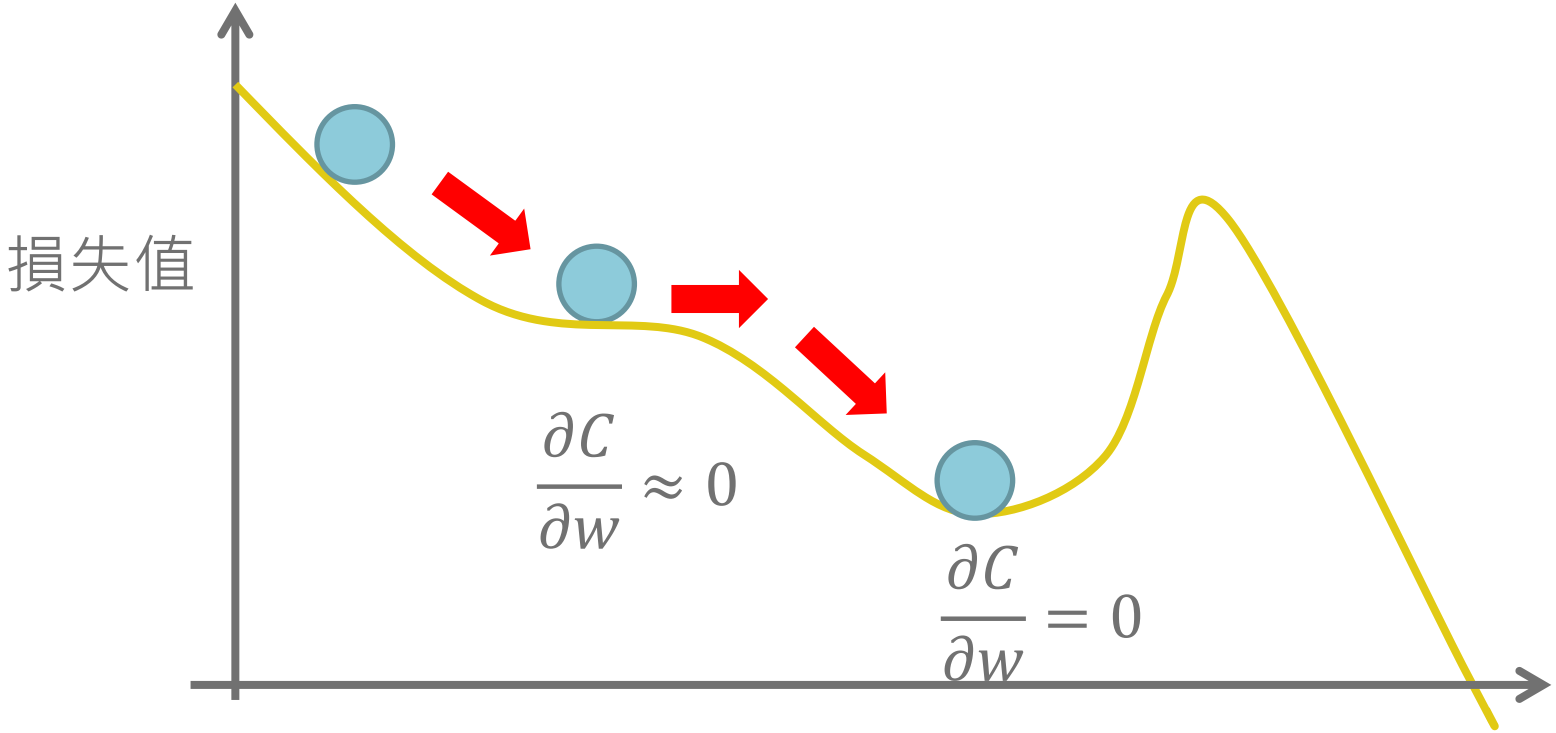
梯度下降法

- 梯度下講法中的學習率
  - 設定太大，有可能找不到實際區域最小值
  - 設定太小，優化時間會拉長



梯度下降法

- **梯度下降法的缺點**
  - 在太平緩的地方會卡住，停止去搜尋更佳的解
  - 為了解決這個問題，有人提出"動量梯度下降法"



動量梯度下降法

- **比較原始梯度下降法以及動量法之差異**

隨機將 θ_1 當起始點

計算 $\nabla\theta_1$

$\theta_2 \leftarrow \theta_1 - \eta\nabla\theta_1$

計算 $\nabla\theta_2$

$\theta_3 \leftarrow \theta_2 - \eta\nabla\theta_2$

$\vdots$

原始方法

隨機將 θ_1 當起始點並初始化動量 $v_1 = 0$

計算 $\nabla\theta_1, v_2 = \lambda v_1 - \eta\nabla\theta_1$

$\theta_2 \leftarrow \theta_1 + v_2$

計算 $\nabla\theta_2, v_3 = \lambda v_2 - \eta\nabla\theta_2$

$\theta_3 \leftarrow \theta_2 + v_3$

$\vdots$

結合動量法

6-3:可適性學習率優化器

- 固定學習率優化器之缺點
- Adagrad/RMSprop/Adam

固定學習率優化器之缺點

- 一般梯度下降法的缺點
  - 學習率是固定的
- 可適性學習率優化器能讓學習率在不同情況下做變動
  - 當目前位置離終點很遠的時候，學習率大
  - 當目前位置離終點很近的時候，學習率小

Adagrad/RMSprop/Adam

- **Adagrad是常見之可適性學習率優化器，其算法如下：**

$$w^1 \leftarrow w^0 - \frac{\eta^0}{\sigma^0} g^0 \qquad \sigma^0 = \sqrt{(g^0)^2}$$

$$w^2 \leftarrow w^1 - \frac{\eta^1}{\sigma^1} g^1 \qquad \sigma^1 = \sqrt{\frac{1}{2}[(g^0)^2 + (g^1)^2]} \qquad \eta^t = \frac{\eta}{\sqrt{t+1}} \qquad g^t = \frac{\partial L(\theta^t)}{\partial w}$$

$$w^3 \leftarrow w^2 - \frac{\eta^2}{\sigma^2} g^2 \qquad \sigma^2 = \sqrt{\frac{1}{3}[(g^0)^2 + (g^1)^2 + (g^2)^2]}$$

$$\vdots$$

$$w^{t+1} \leftarrow w^t - \frac{\eta^t}{\sigma^t} g^t \qquad \sigma^t = \sqrt{\frac{1}{t+1}\sum_{i=0}^{t}(g^i)^2}$$

Adagrad/RMSprop/Adam

- **RMSprop是常見之可適性學習率優化器，其算法如下：**

$$w^1 \leftarrow w^0 - \frac{\eta}{\sigma^0} g^0 \qquad \sigma^0 = g^0$$

$$w^2 \leftarrow w^1 - \frac{\eta}{\sigma^1} g^1 \qquad \sigma^1 = \sqrt{\alpha(\sigma^0)^2 + (1-\alpha)(g^1)^2}$$

$$w^3 \leftarrow w^2 - \frac{\eta}{\sigma^2} g^2 \qquad \sigma^2 = \sqrt{\alpha(\sigma^1)^2 + (1-\alpha)(g^2)^2}$$

$$\vdots$$

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sigma^t} g^t \qquad \sigma^t = \sqrt{\alpha(\sigma^{t-1})^2 + (1-\alpha)(g^t)^2}$$

Adagrad/RMSprop/Adam

- **Adam是常見之可適性學習率優化器，其算法如下：**

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
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)
        $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
        $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
        $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
    end while
    return θ_t (Resulting parameters)

https://arxiv.org/pdf/1412.6980.pdf

6-3 Demo

- 開啟Demo_6-3.ipynb
- 不同優化器使用

線上Corelab

- 題目1：資料批次輸入網路
  - 請以100筆為單位，將其一次輸入DNN網路，並輸出Cross-Entropy值
- 題目2：EPOCH的使用與優化器的搭配
  - 請執行10個EPOCH，每個EPOCH輸入100筆資料到網路中，並輸出Cross-Entropy值
- 題目3：學習率與梯度下降法，觀察損失函數的變化
  - 建立5層的DNN網路、損失函數請使用cross entropy、優化器請選用Gradient Descent、Batch size請設定200、Epoch請設定50、Learning rate請設定0.02

本章重點精華回顧

- 批次輸入資料
- Epoch及Step差異
- 優化器原理
- 常見之可適性學習率優化器

Lab:優化神經網路

- **Lab01: 批次輸入**
- **Lab02: 改變學習率**
- **Lab03: 不同優化器使用**

Estimated time:
20 minutes


THANK YOU