Machine Learning Techniques

Shyi-Chyi Cheng (鄭錫齊)
Email:csc@mail.ntou.edu.tw
Tel: 02-24622192-6653

章節目錄

與學習有關的技巧

學習重點

- 神經網路的參數更新法:SGD、Momentum、AdaGrad、Adam等
- 權重預設值設定，在進行正確的學習時，非常重要。
- 使用"Xavier預設值"及"He預設值"，當作權重預設值，效果較佳。
- 使用Batch Normalizaton可以提升學習速度，而且不會過度依賴預設值，效果比較穩定。
- Weight decay與Dropout是控制過度學習的正規化技術。
- 尋找超參數的有效方法是逐漸縮小優良值存在的範圍。

更新參數

- 神經網路的學習目的
  - 透過最佳化(Optimization)方法，尋找能盡量縮小損失函數的最佳參數
- 這個問題困難嗎?
  - 網路很多層→參數空間很大
  - 參數空間的行態複雜，最佳化結構不明確，很難製作最佳化參數尋找地圖
  - 目前為止，沒有公式解
- 目前解法
  - 利用計算參數的梯度(微分)，往梯度方向重複更新參數
  - 這個方法叫準確率梯度下降法(stochastic gradient descent；SGD)

沒地圖、地形不解，要如下下山最快又安全呢?

SGD

- 參數W更新方式:

$$W \leftarrow W - \eta \frac{\partial L}{\partial W}$$

$\frac{\partial L}{\partial W}$:損失函數梯度值

η: 學習率，約在0.01到0.001之間

```
class SGD:
    def __init__(self, lr = 0.01):
        self.lr = lr
    def update(self, params, grads):
        for key in params.key():
            params[key] -= self.lr * grads[key]
```

Python呼叫SGD

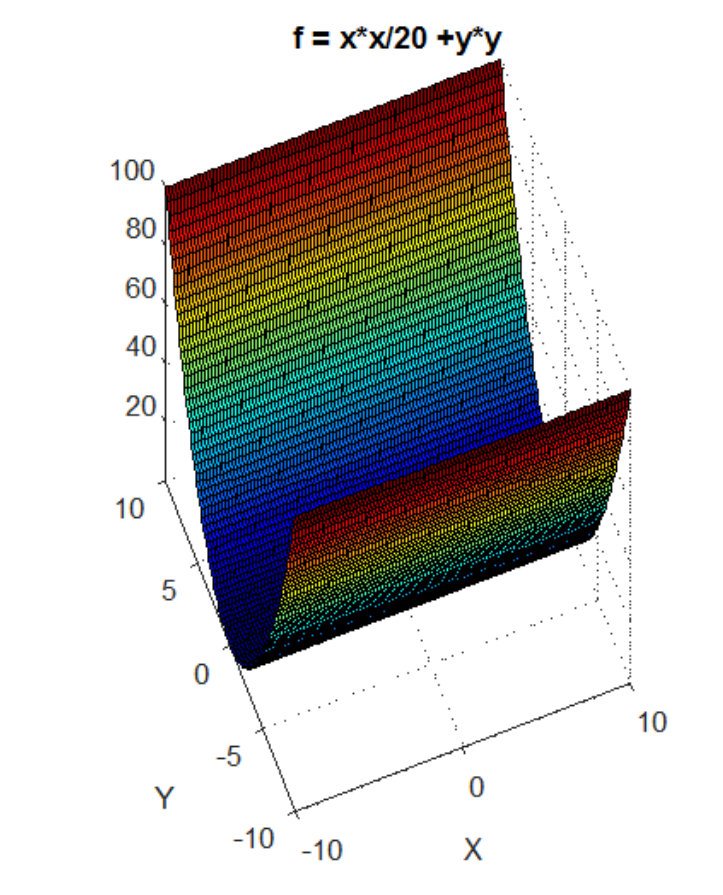
- 虛擬碼(不完整，還不能值直接執行)

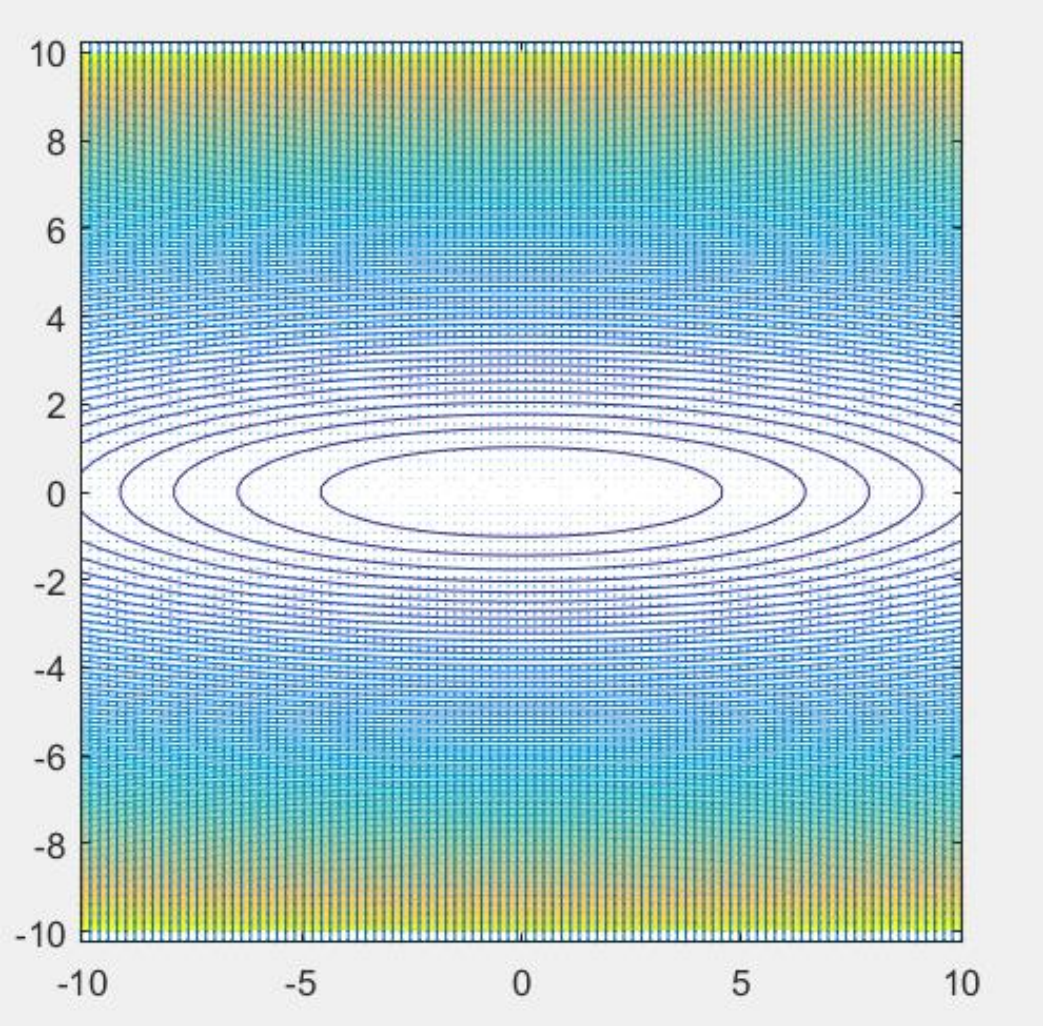
```
network = TwoLayerNet(…)
optimizer = SGD()

for i in range(10000):
    …
    x_batch, t_batch = get_mini_batch(…)  #小批次學習
    grads = network.gradient(x_batch, t_batch)
    params = network.params
    optimizer.update(params, grads)
    …
```

完整程式請參考:
https://github.com/Lasagne/Lasagne/blob/master/lasagne/updates.py

SGD的缺點

$$f(x,y)=\frac{1}{20}x^2+y^2$$

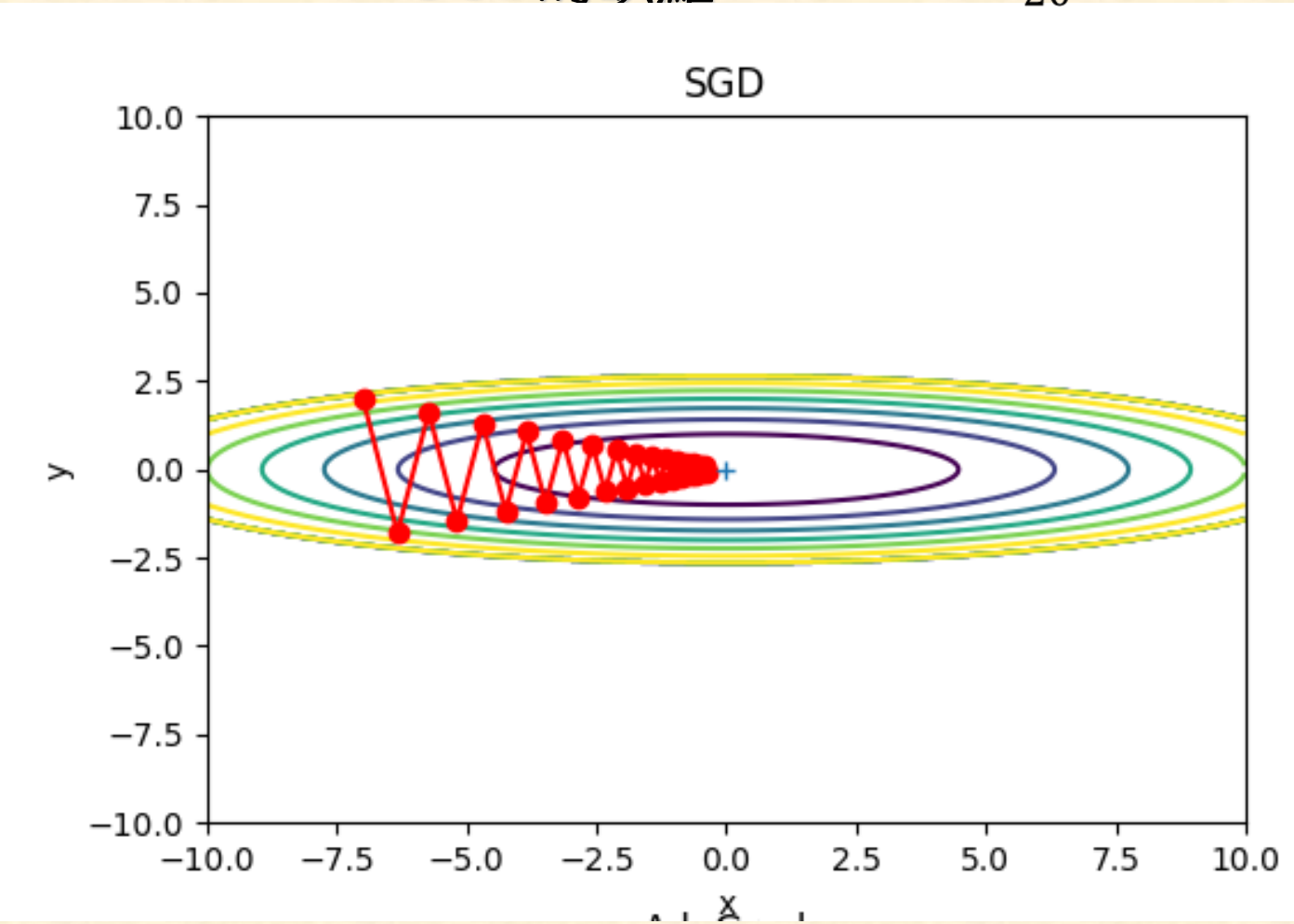




梯度方向不見得指向最佳化值

SGD的缺點

$$f(x,y) = \frac{1}{20}x^2 + y^2$$



函數形狀不具等向性:參數尋找路徑呈鋸齒狀

Momentum (運動量)

- 參數W更新方式:

$$v \leftarrow \alpha v - \eta \frac{\partial L}{\partial W}$$
$$W \leftarrow W + v$$

$\frac{\partial L}{\partial W}$:損失函數梯度值

η: 學習率，約在0.01到0.001之間

v: 對應速度

α: 磨擦係數 ($\alpha \sim 0.8$)

Momentum (運動量)

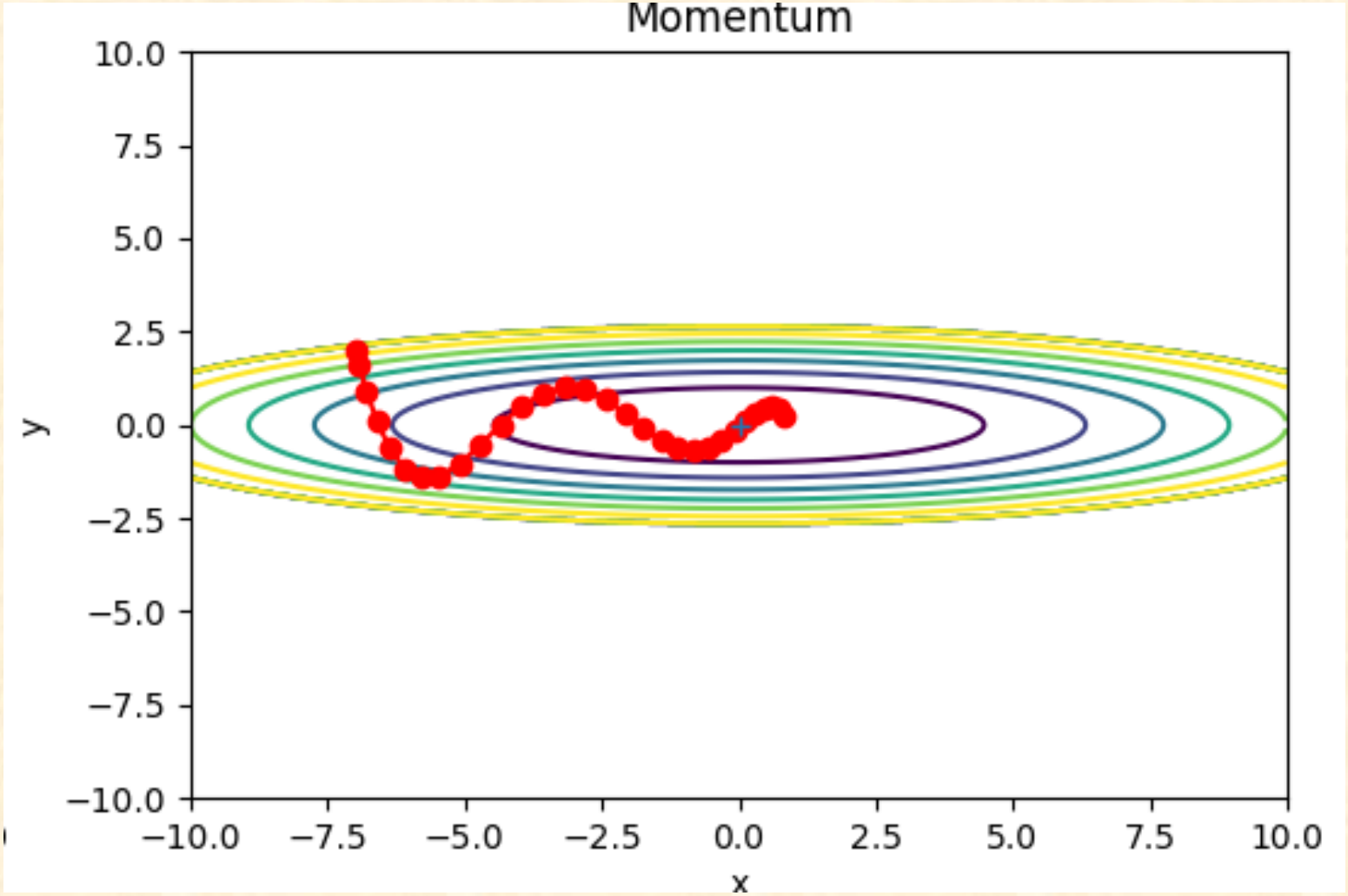
❖ Python執行方式:

```
class Momentum:
    def __init__(self, lr=0.01, momentum=0.9):
        self.lr = lr
        self.momentum = momentum
        self.v = None

    def update(self, params, grads):
        if self.v is None:
            self.v = {}
            for key, val in params.items():
                self.v[key] = np.zeros_like(val)

        for key in params.keys():
            self.v[key] = self.momentum*self.v[key] - self.lr*grads[key]
            params[key] += self.v[key]
```

參數結構相同的資料當作字典變數維持下來

Momentum (運動量)

$$f(x,y) = \frac{1}{20}x^2 + y^2$$



AdaGrad

- 學習率η值對神經網路學習影響很大，太小學習速度太慢；太大則會往外擴散，無法正確學習。
- 可使用學習衰減(learning rate decay)解決這個問題
  - 開始學習時，先用大的η值，然後逐漸降低η值
- 參數W更新方式:

$$\mathrm{h} \leftarrow \mathrm{h} + \frac{\partial L}{\partial \mathrm{W}} \odot \frac{\partial L}{\partial \mathrm{W}}$$

$$\mathrm{W} \leftarrow \mathrm{W} - \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial \mathrm{W}}$$

$\frac{\partial L}{\partial \mathrm{W}}$:損失函數梯度值

η: 學習率，約在0.01到0.001之間

h: 過去之梯度值平方和

AdaGrad

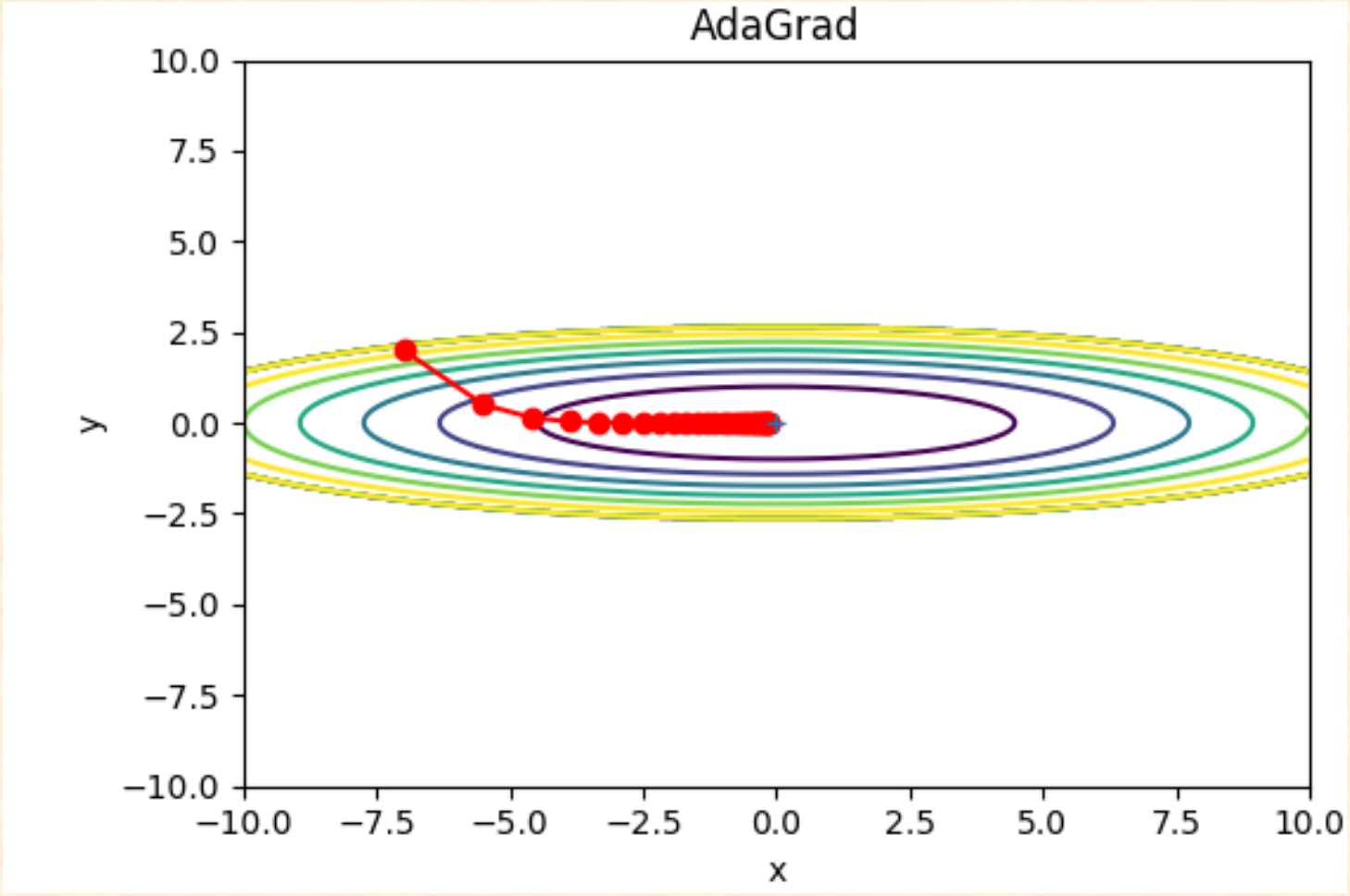
❖ Python執行方式:

```
class AdaGrad:
    def __init__(self, lr=0.01):
        self.lr = lr
        self.h = None

    def update(self, params, grads):
        if self.h is None:
            self.h = {}
            for key, val in params.items():
                self.h[key] = np.zeros_like(val)

        for key in params.keys():
            self.h[key] += grads[key] * grads[key]
            params[key] -= self.lr * grads[key] / (np.sqrt(self.h[key]) + 1e-7)
```

AdaGrad

$$f(x,y) = \frac{1}{20}x^2 + y^2$$

- Python執行結果:



Adam

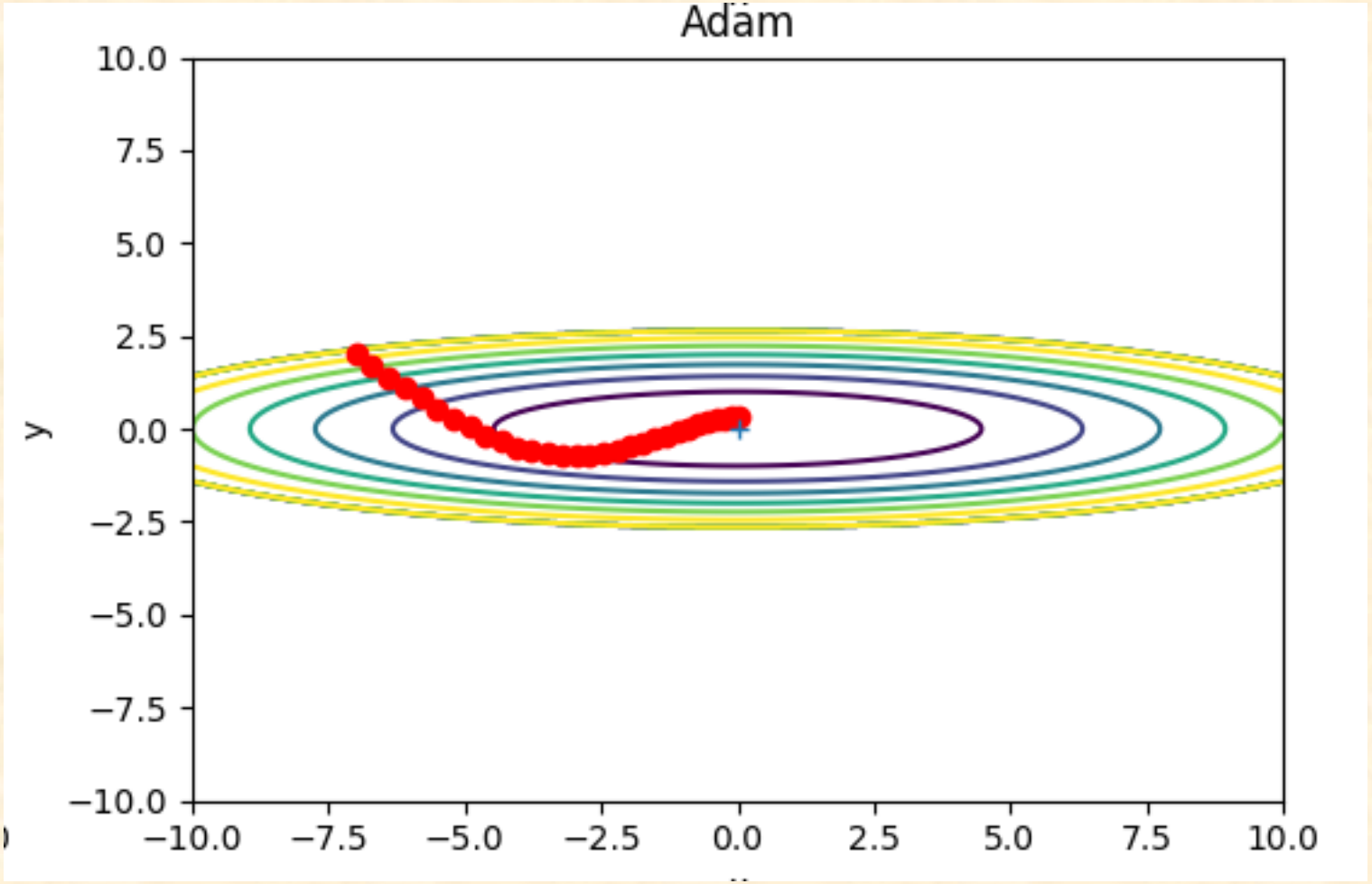
- Adam融合Momentum及AdaGrad兩種分法

```
class Adam:
    def __init__(self, lr=0.001, beta1=0.9, beta2=0.999):
        self.lr = lr
        self.beta1 = beta1
        self.beta2 = beta2
        self.iter = 0
        self.m = None
        self.v = None
    def update(self, params, grads):
        if self.m is None:
            self.m, self.v = {}, {}
            for key, val in params.items():
                self.m[key] = np.zeros_like(val)
                self.v[key] = np.zeros_like(val)
        self.iter += 1
        lr_t  = self.lr * np.sqrt(1.0 - self.beta2**self.iter) / (1.0 - self.beta1**self.iter)
        for key in params.keys():
            self.m[key] += (1 - self.beta1) * (grads[key] - self.m[key])
            self.v[key] += (1 - self.beta2) * (grads[key]**2 - self.v[key])
            params[key] -= lr_t * self.m[key] / (np.sqrt(self.v[key]) + 1e-7)
```

Adam

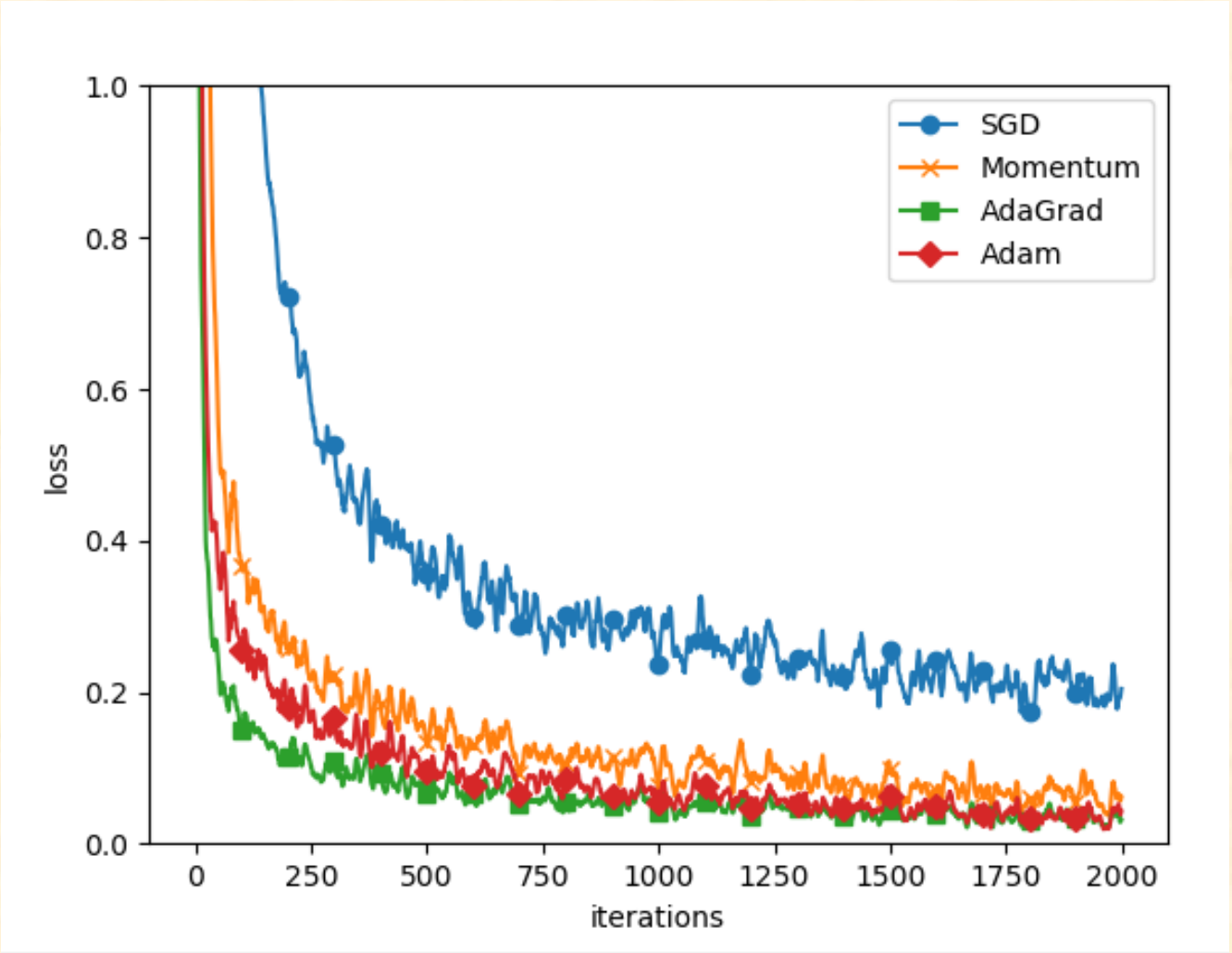
$$f(x,y) = \frac{1}{20}x^2 + y^2$$

❖ Python執行結果:



利用MNIST資料集比較更新手法

- 請參考ch06/optimizer_compare_mnist.py



權重的預設值

- 權重的預設值影響神經網路的學習正確性。
- 太大的權重值會造成過度學習。
  - 權重衰減(Weight decay)的方法，就是為了產生過大的權重值而設計的方法
- 權重值要盡量小，但要避免全變成0→無法正確學習
  - 利用標準差結合隨機數產生器，可控制及產生適當的權重值
  - Ex. 0.01* np.random.raddn(10,100) 可產生標準差為0.01的常態分佈權重值
- 權重值不能預設為均一值，相反地最好每個權重值都不一樣。
- 不同的權重值會產生不同的隱藏層的活性化分佈

隱藏層的活性化分佈

- 請參考ch06/weight_init_activation_histogram.py

```
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

input_data = np.random.randn(1000, 100)  # 1000個資料
node_num = 100  # 各隱藏層的結點(神經元)數量
hidden_layer_size = 5  # 隱藏層有5層
activations = {}  # 在這裡儲存Activation函數的結果

for i in range(hidden_layer_size):
    if i != 0:
        x = activations[i-1]

    # 權重值的預設值實驗
    w = np.random.randn(node_num, node_num) * 1
    a = np.dot(x, w)
    z = sigmoid(a)
    activations[i] = z
```

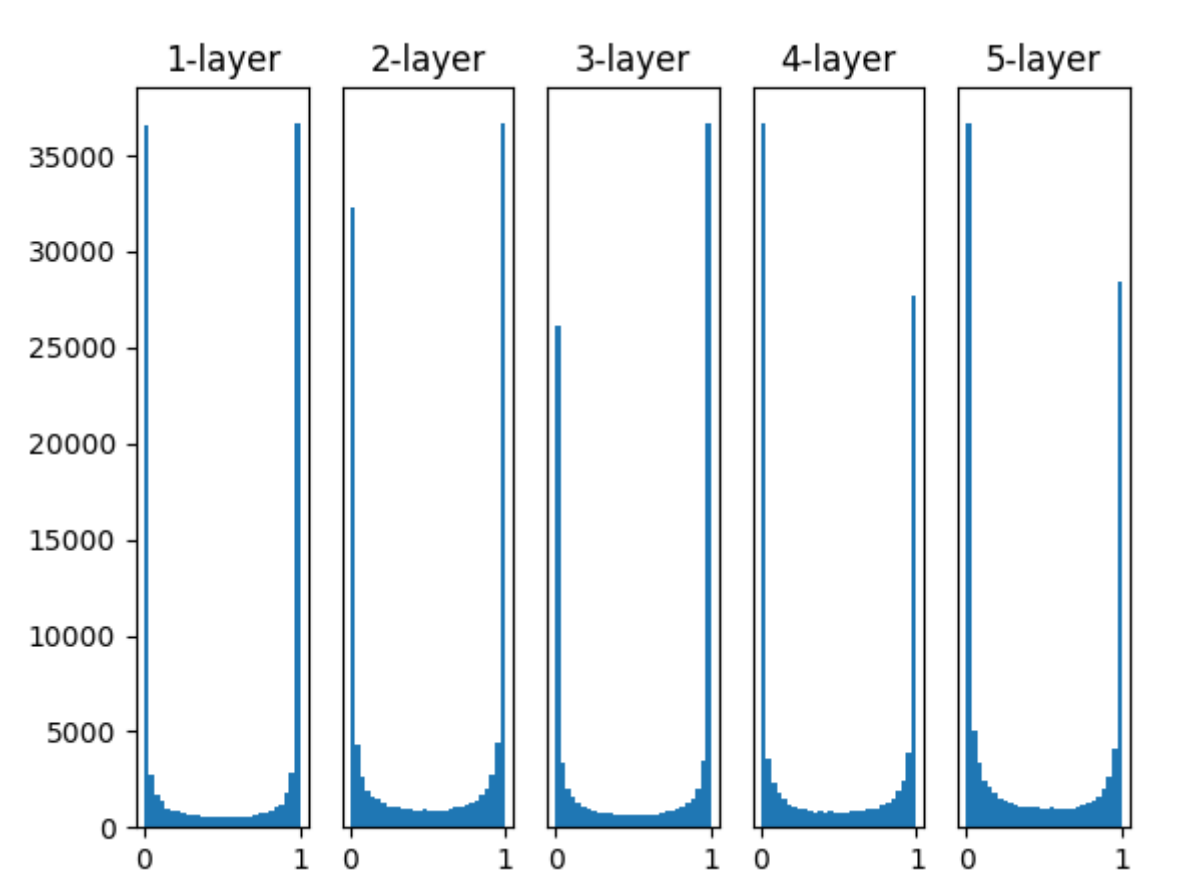
隱藏層的活性化分佈

- 請參考ch06/weight_init_activation_histogram.py

```
# 繪製分佈圖
for i, a in activations.items():
    plt.subplot(1, len(activations), i+1)
    plt.title(str(i+1) + "-layer")
    if i != 0: plt.yticks([], [])
    # plt.xlim(0.1, 1)
    # plt.ylim(0, 7000)
    plt.hist(a.flatten(), 30, range=(0,1))
plt.show()
```

隱藏層的活性化分佈

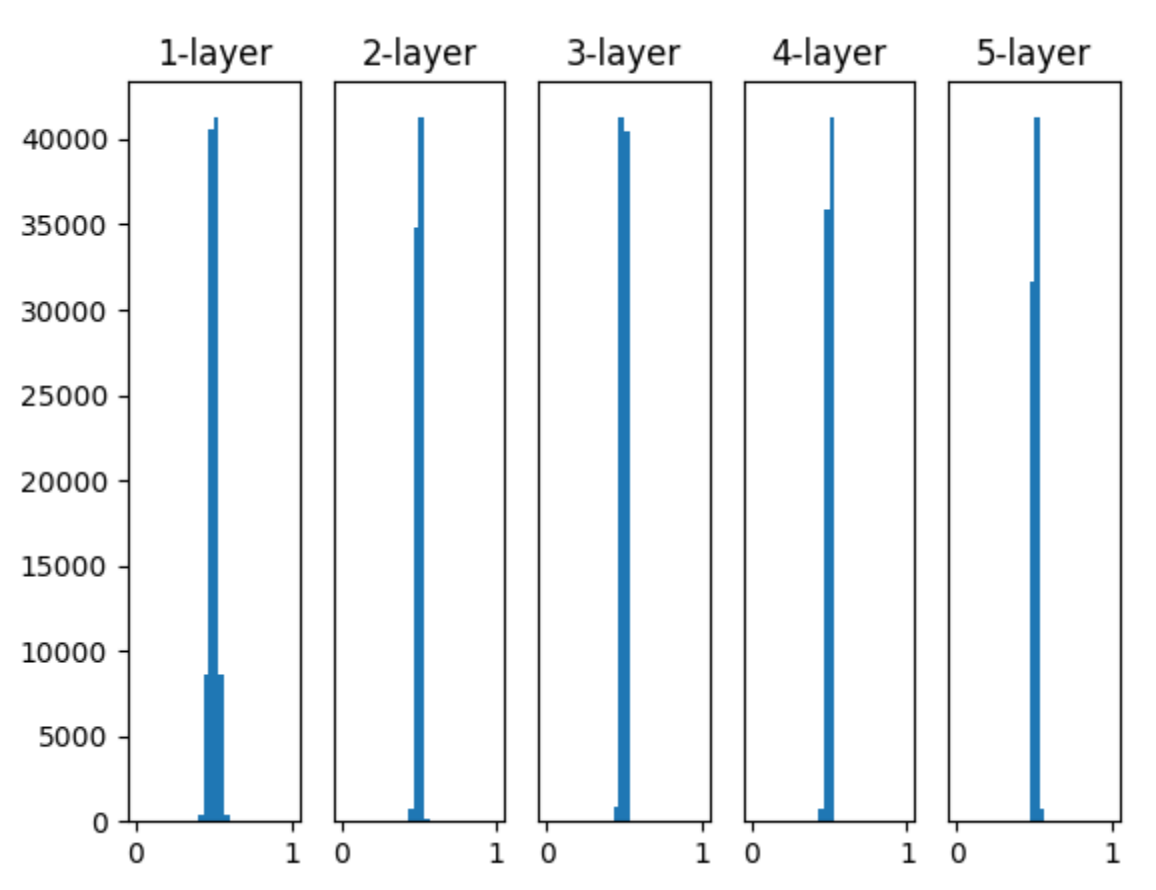
- 請參考ch06/weight_init_activation_histogram.py



隱藏層的活化值分佈篇向0跟1兩極端→權重微分值也因而趨近0→反向傳播的梯度值會逐漸變小、消失→產生"梯度消失"問題。

隱藏層的活性化分佈

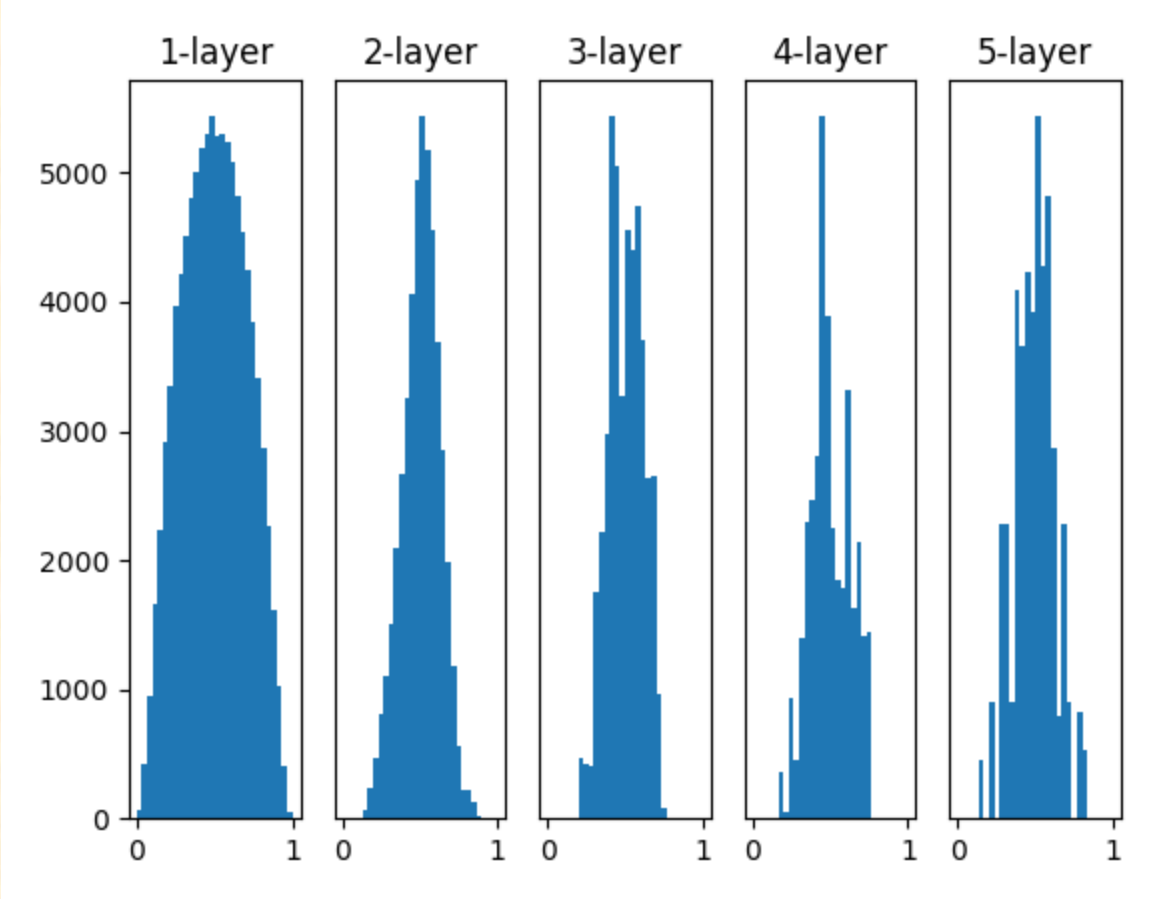
- 請參考ch06/weight_init_activation_histogram.py
- 修改權重預設值為: w = 0.01* np.random.raddn(10,100)



隱藏層的活化值分佈集中0.5附近→沒產生"梯度消失"問題
→但產生活化值分佈偏差問題。
當大部分的神經元輸出值都一樣時，表示有很多神經元是多餘的。

隱藏層的活性化分佈

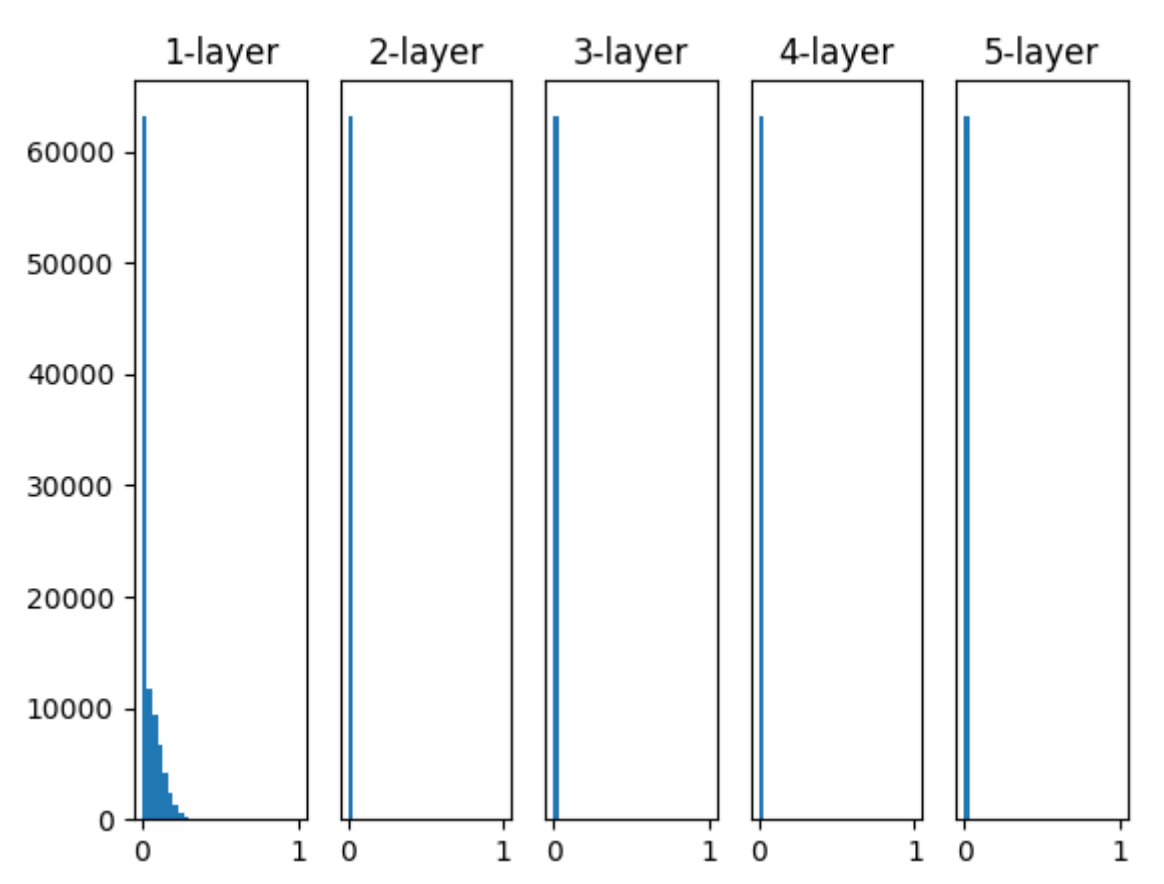
- 請參考ch06/weight_init_activation_histogram.py
- 修改權重預設值為Xavier預設值:假設上層具n個節點，則以$\frac{1}{\sqrt{n}}$ 標準差的常態分佈來初始化



隱藏層的活化值分佈較廣，學習效果較佳

ReLU的權重值預設

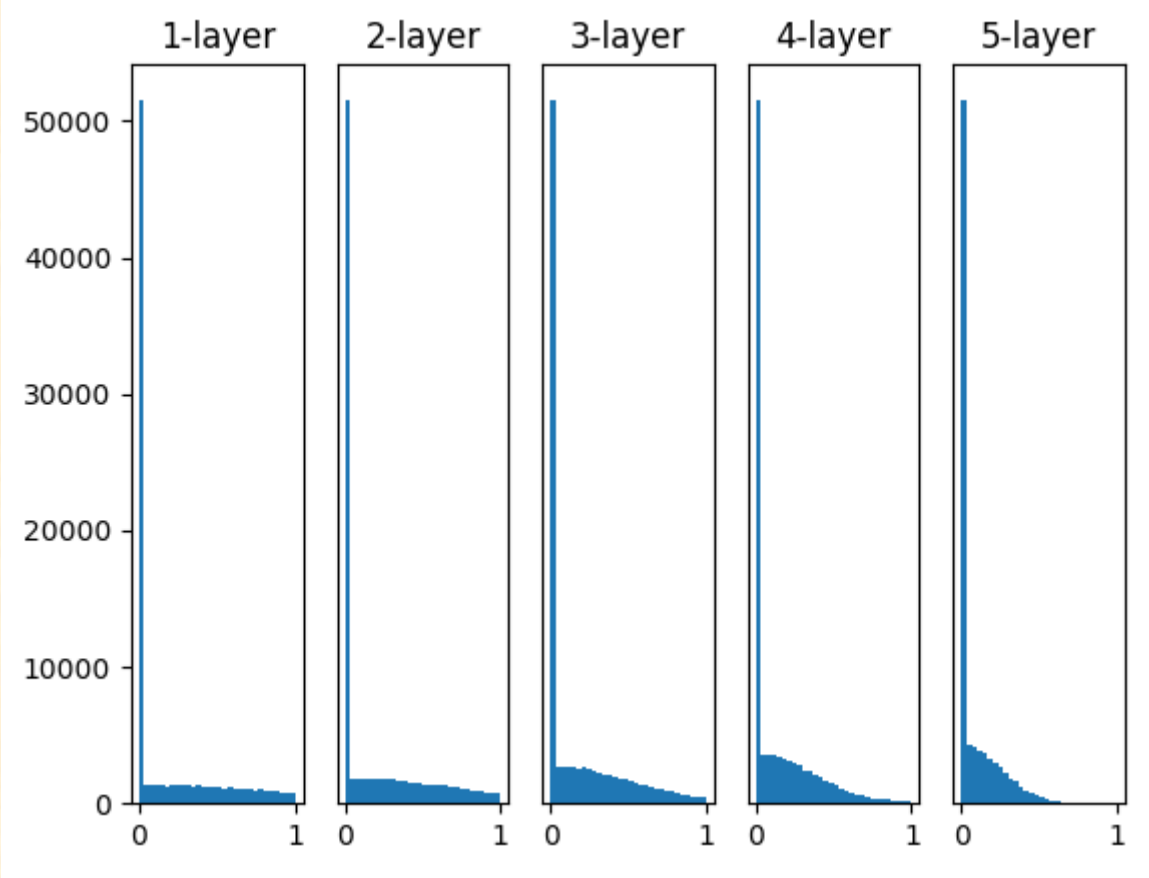
- 請參考ch06/weight_init_activation_histogram.py
- 權重預設值為: w = 0.01* np.random.raddn(10,100)



隱藏層的活化值分佈篇向0跟1兩極端→權重微分值也因而趨近0
→反向傳播的梯度值會逐漸變小、消失→產生"梯度消失"問題

ReLU的權重值預設

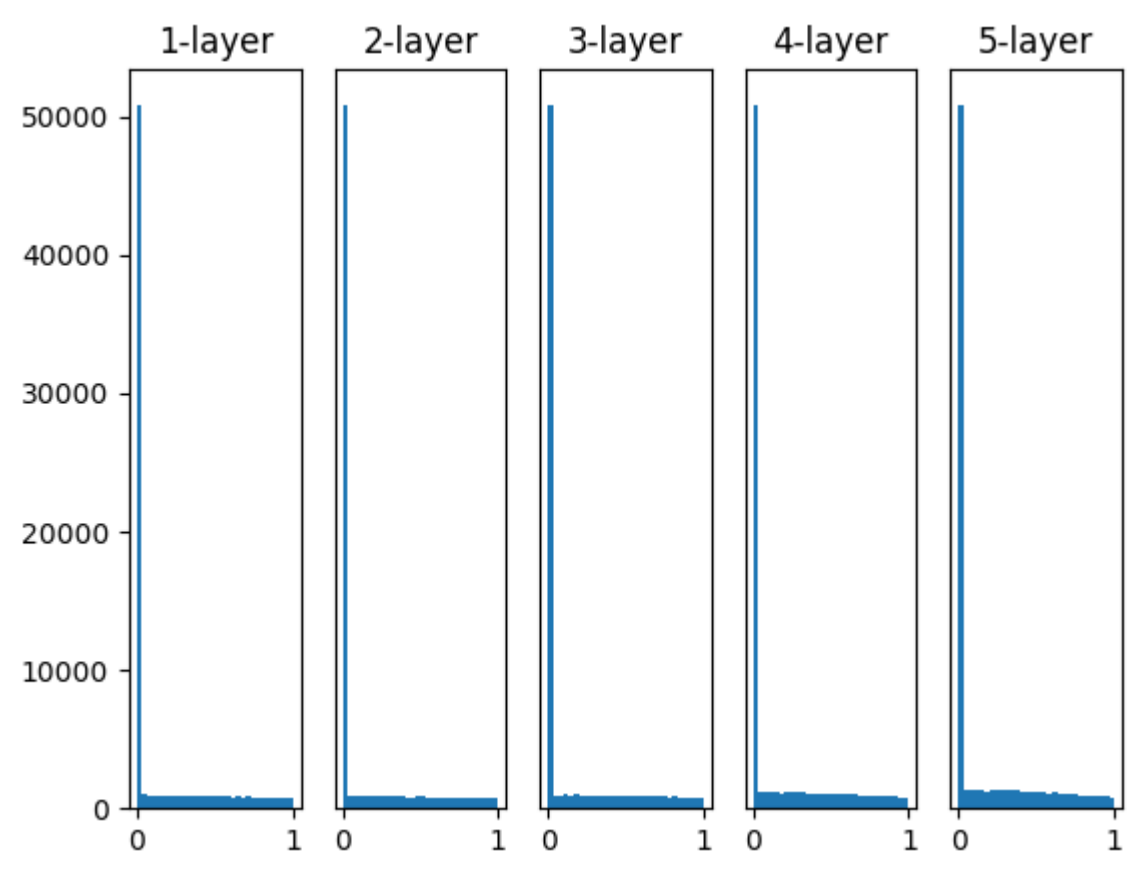
- 請參考ch06/weight_init_activation_histogram.py
- 權重預設值為: “Xavier預設值”



隱藏層的活化值分佈篇向0跟1兩極端→權重微分值也因而趨近0→反向傳播的梯度值會逐漸變小、消失→產生”梯度消失”問題

ReLU的權重值預設

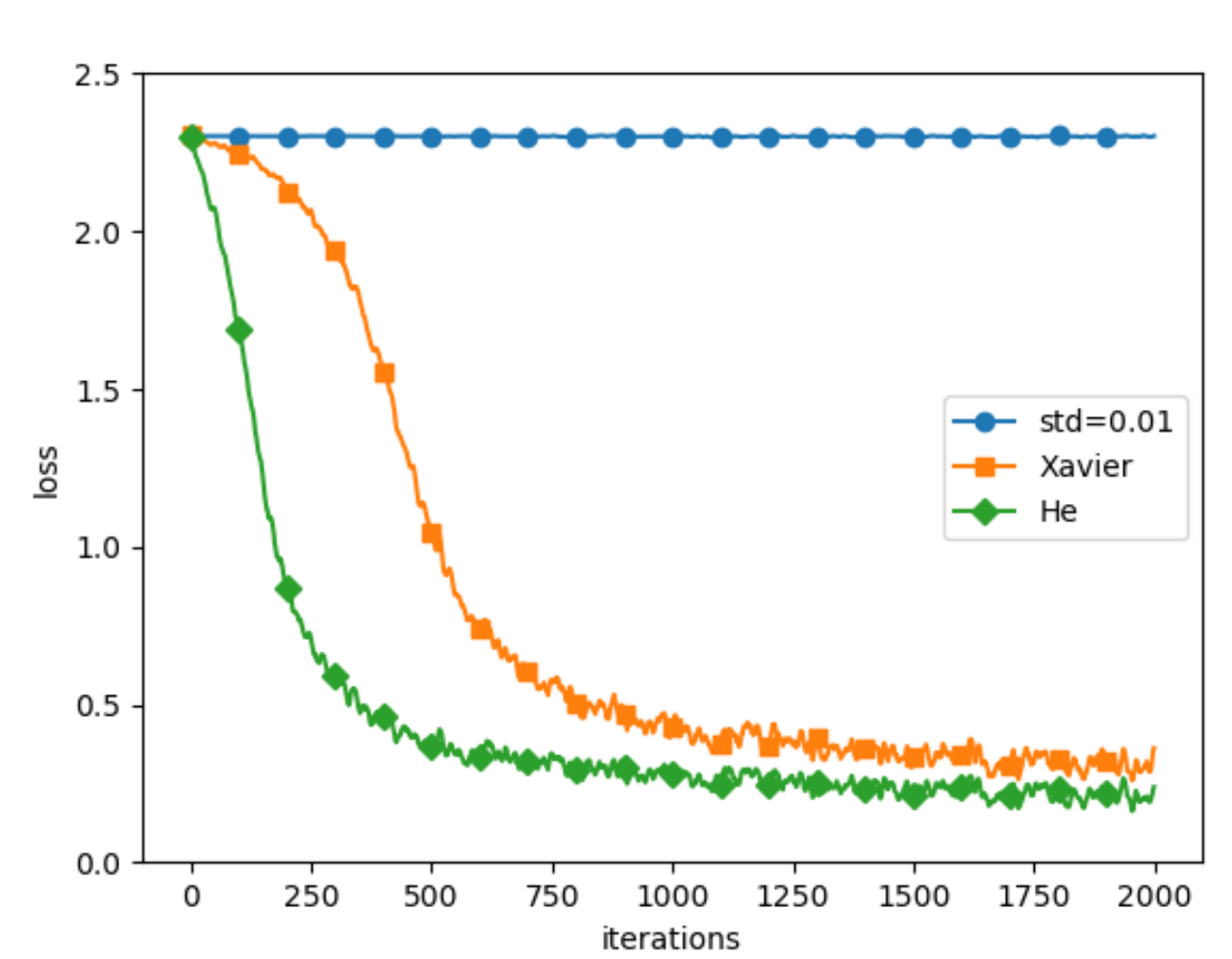
- 請參考ch06/weight_init_activation_histogram.py
- 權重預設值為 "He預設值":假設上層具n個節點，則以$\frac{\sqrt{2}}{\sqrt{n}}$ 標準差的常態分佈來初始化



隱藏層的活化值分佈較廣，學習效果較佳

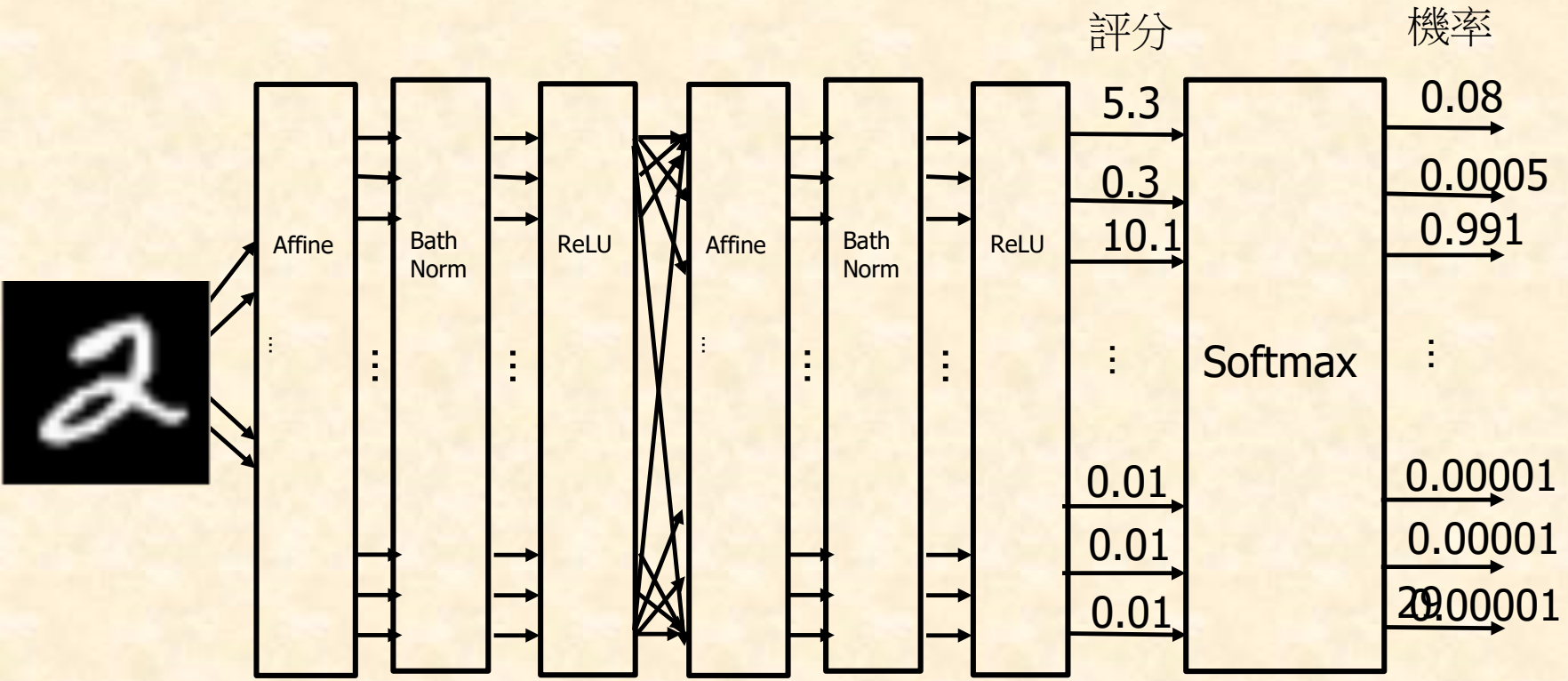
利用MNIST資料集比較權重預設手法

- 請參考ch06/weight_init_compare.py



Batch Normalization

- Batch normalization(Batch Norm)是一種技巧，用以強制性調整活化值分佈廣度之正規化處理
- 優點
  - 快速學習:以小批次學習
  - 不會過度依賴預設值
  - 控制過度學習(減少Dropout等必要性)



Batch Normalization

- 每一小 批次 B=$\{x_1, x_2, \ldots, x_m\}$輸入資料，計算

  平均值$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$

  分散度$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$

- 再進行讓輸入資料的平均為0、分散為1的正規化處理

$$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

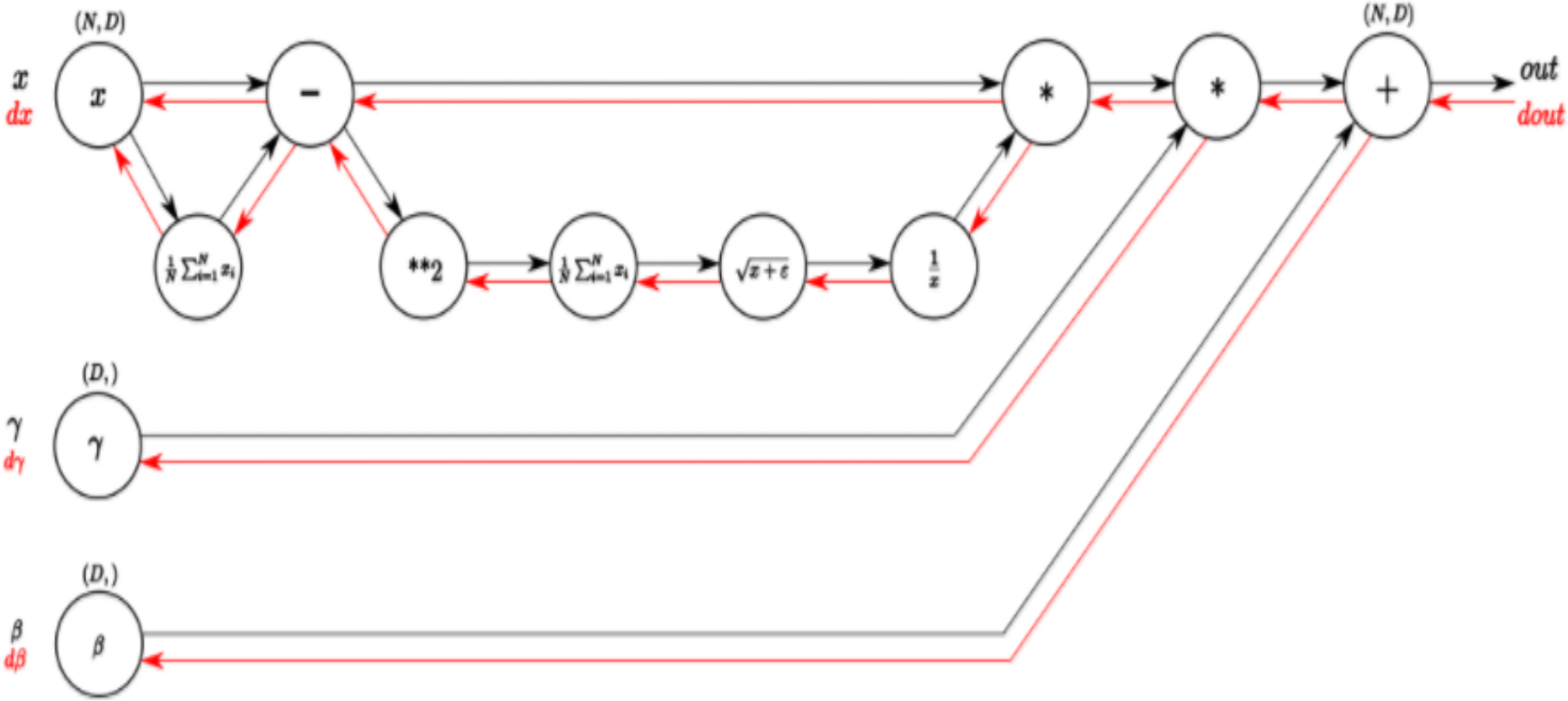
  - 在活化函數之前正規化，可以減少資料分佈的偏差

- 進一步的正規化動作

$$y_i \leftarrow \gamma \hat{x}_i + \beta$$

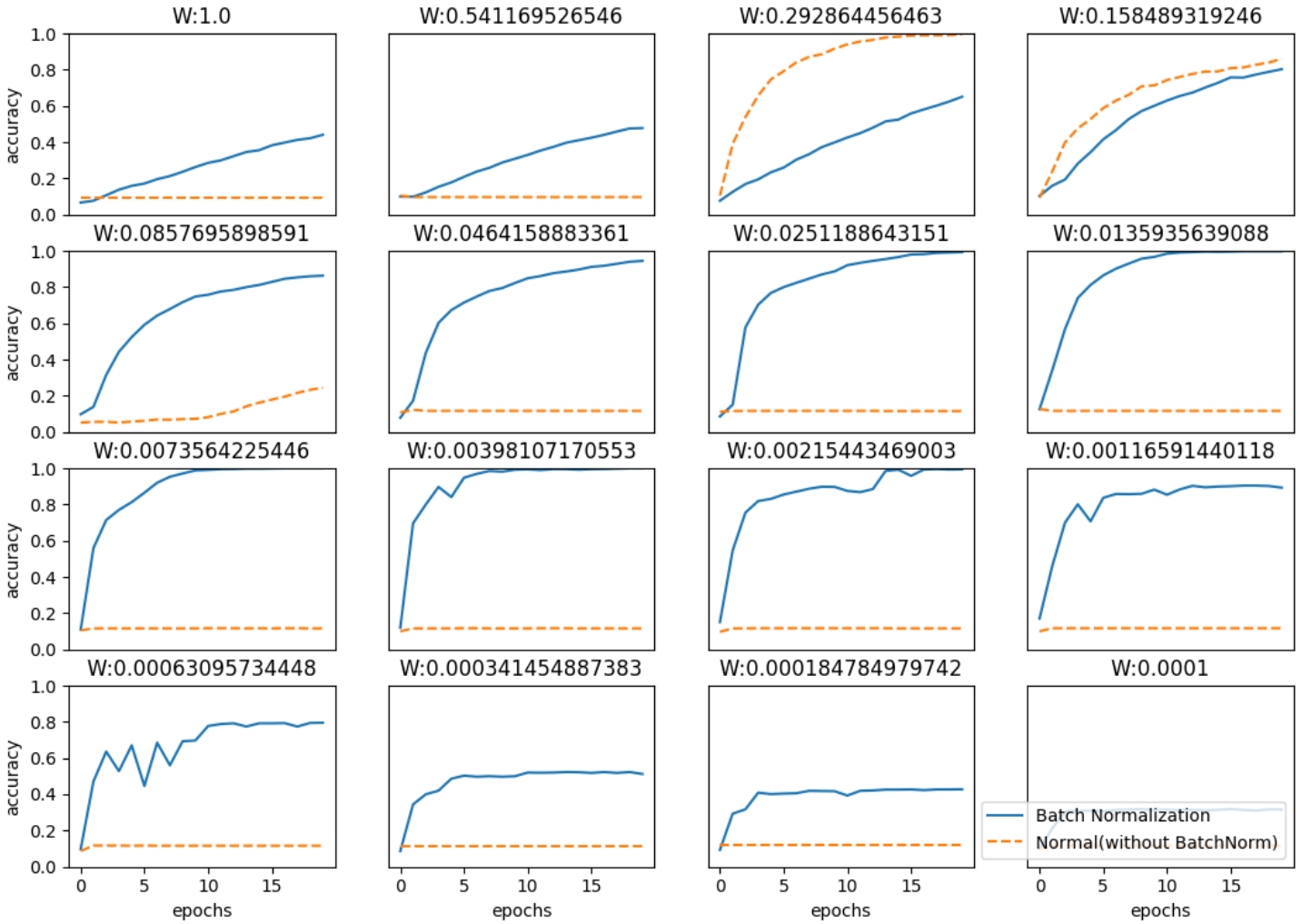
- 初值設定: $\gamma = 1_i \beta = 0$

Computational Graph of Batch Normalization Layer



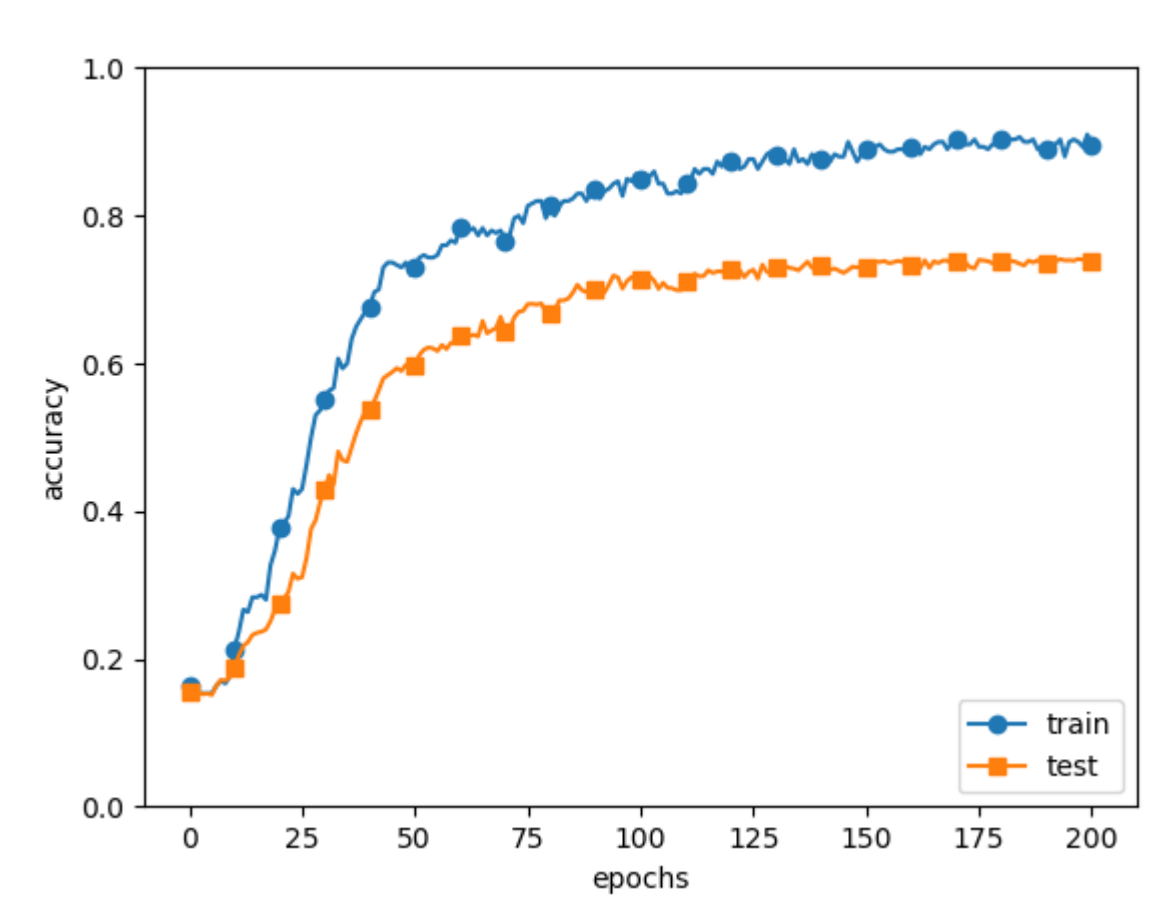
Batch Normalization的評價

- 請參考ch06/batch_norm_test.py



過度學習（overfitting）

- 主因
  - 擁有大量參數，表現力高的模型
  - 訓練資料太少
- 以MNIST資料集當例子，訓練資料從60,000縮成300，網路變7層的結果



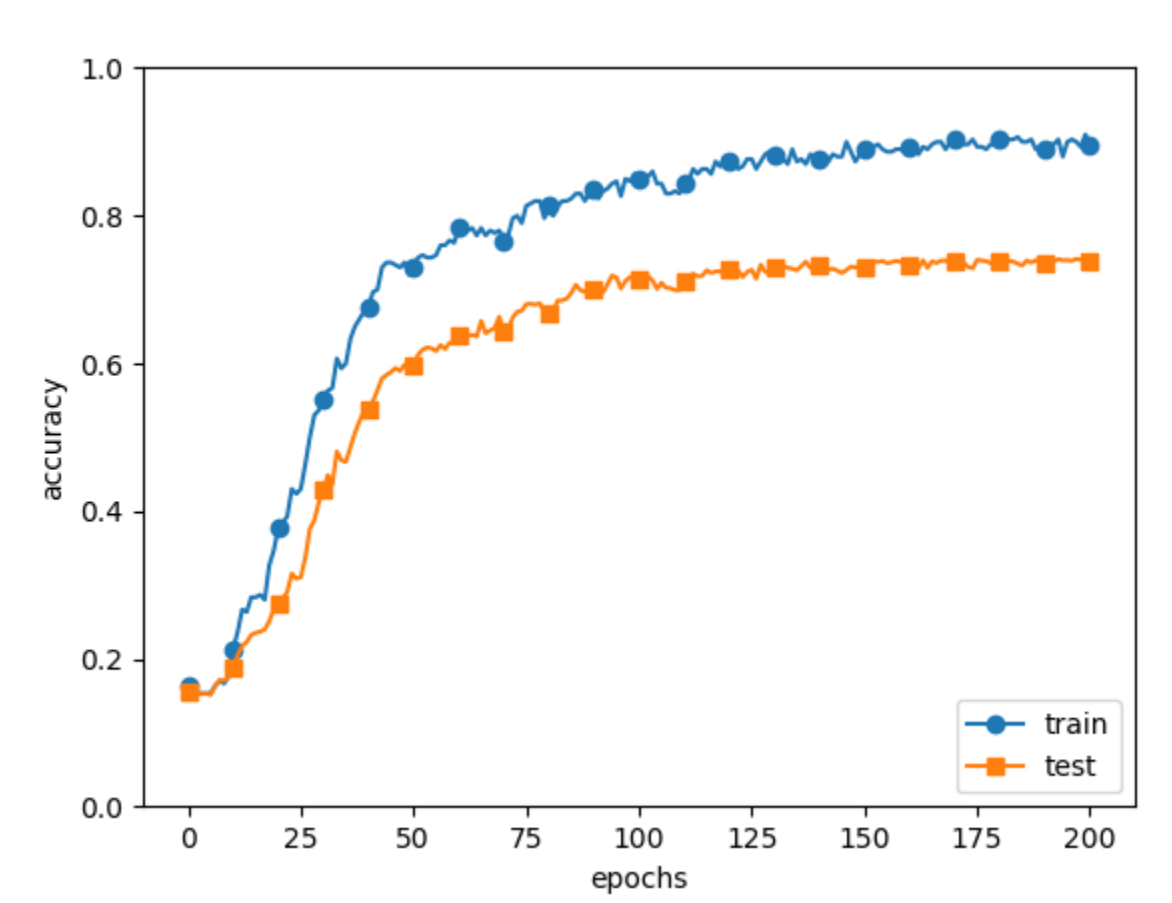
Weight decay

- 常常被用來控制神經網路過度學習的方法
- 基本想法
  - 針對擁有較大權重的部分，課以罰金
- 作法
  - 假設權重是W，新的損失函數為

$$L^{new)} = L^{(old)} + \frac{1}{2}\lambda \mathrm{W}^2$$

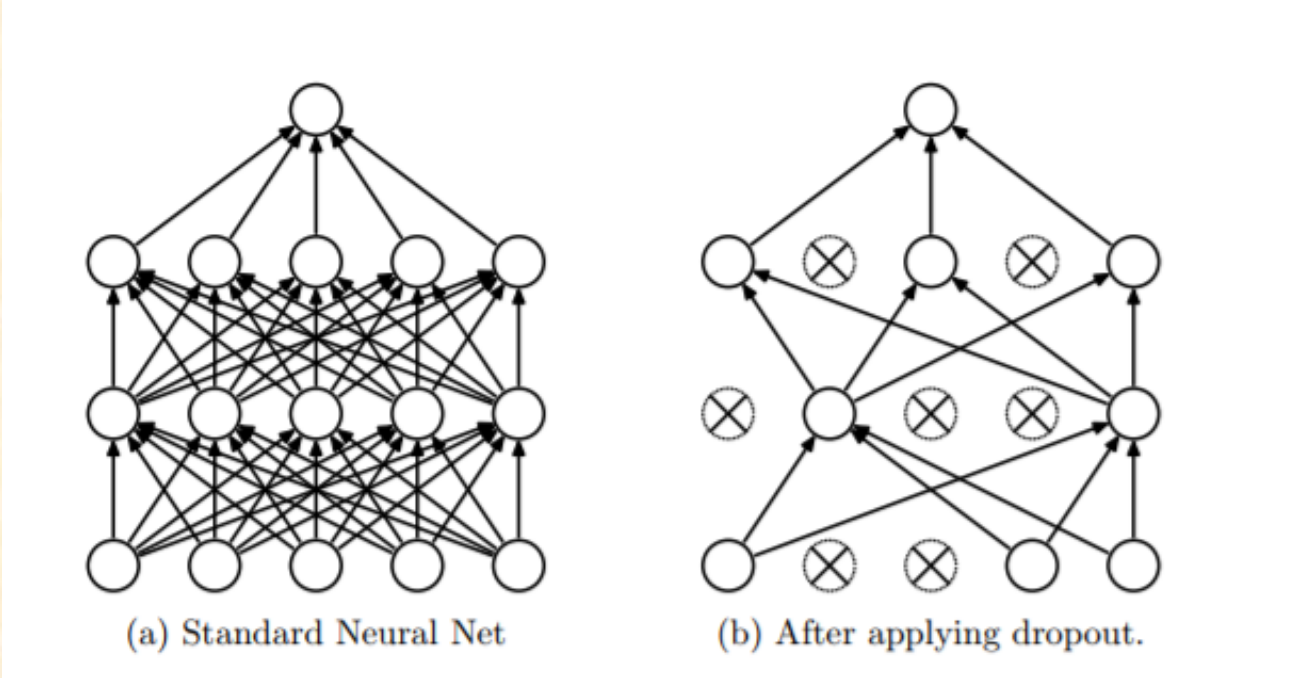
Weight decay實驗結果

❖ $\lambda = 0.1$



Dropout

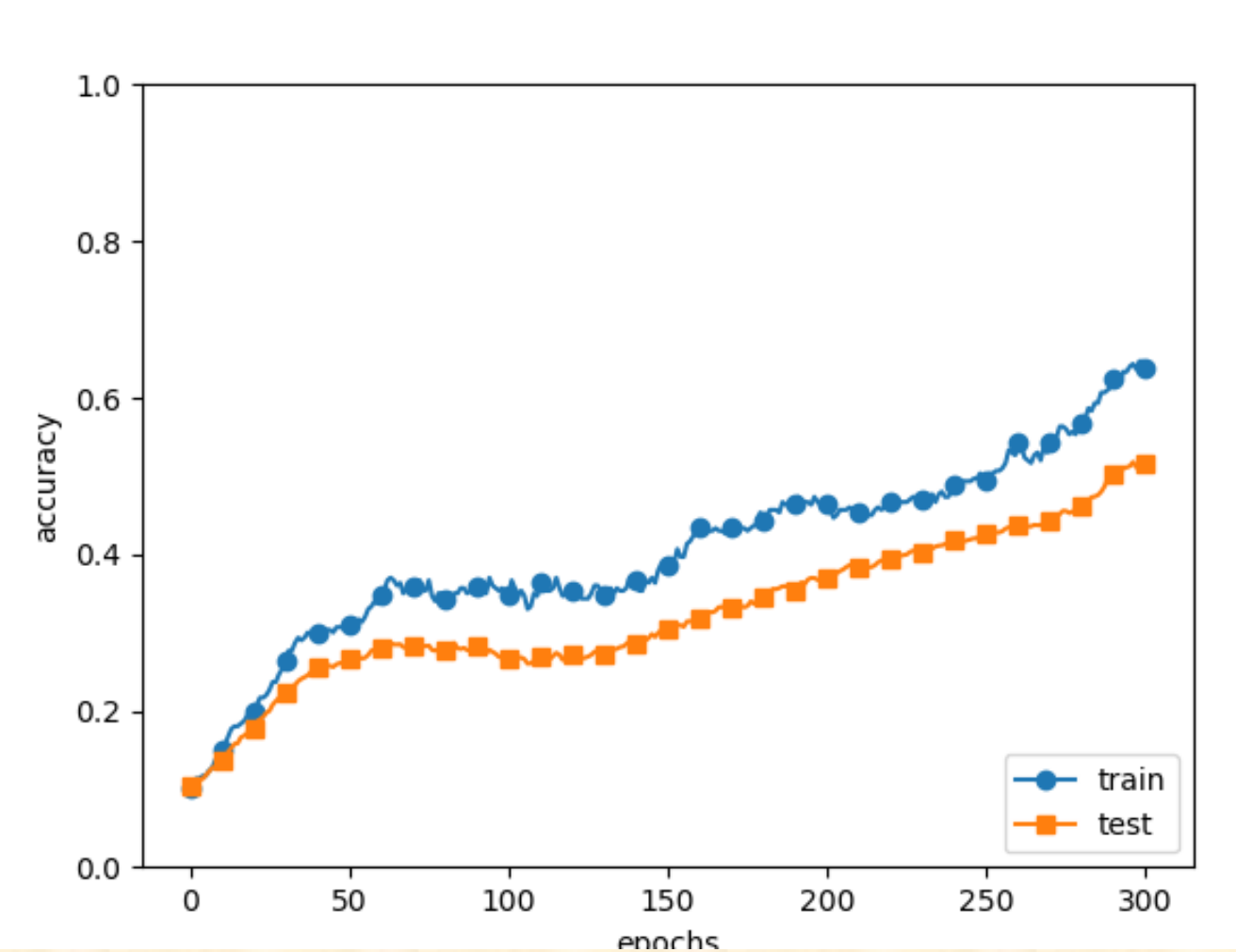
- 常常被用來控制神經網路過度學習的另一種方法
- Dropout會一邊隨機消除神經元、一邊學習的方法
- 訓練時，隨機選出隱藏層的神經元、刪除神經元，使之失去信號傳導功能
- 測試時，會傳遞所有的神經元信號，但各時經元的輸出，要成上訓練時刪除的神經元比例再傳遞。



(a) Standard Neural Net

(b) After applying dropout.

Dropout實驗


1.0
0.8
0.6
0.4
0.2
0.0
accuracy
0
50
100
150
200
250
300
epochs
train
test

驗證超參數

- 資料集應分成3部分
  - 訓練資料集
    - 用以進行參數學習
  - 驗證資料集
    - 用以評估超參數的效能
  - 測試資料集
    - 用以檢查一般化能力
- 超參數最佳化範例
  - 請參考 ch06/hyperparameter_optimization.py

Any Questions?