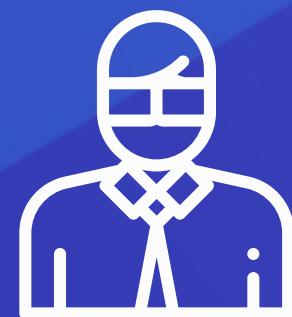


Day 74 Gradient Descent 數學式介紹

Gradient Descent 數學式



陳宇春

出題教練

知識地圖 深度學習簡介

深度學習體驗 - 啟動函數與正規化

深度神經網路

Supervised Learning Deep Neural Network (DNN)

簡介 Introduction

套件介紹 Tools: Keras

組成概念 Concept

訓練技巧 Training Skill

應用案例 Application

卷積神經網路

Convolutional Neural Network (CNN)

簡介 introduction

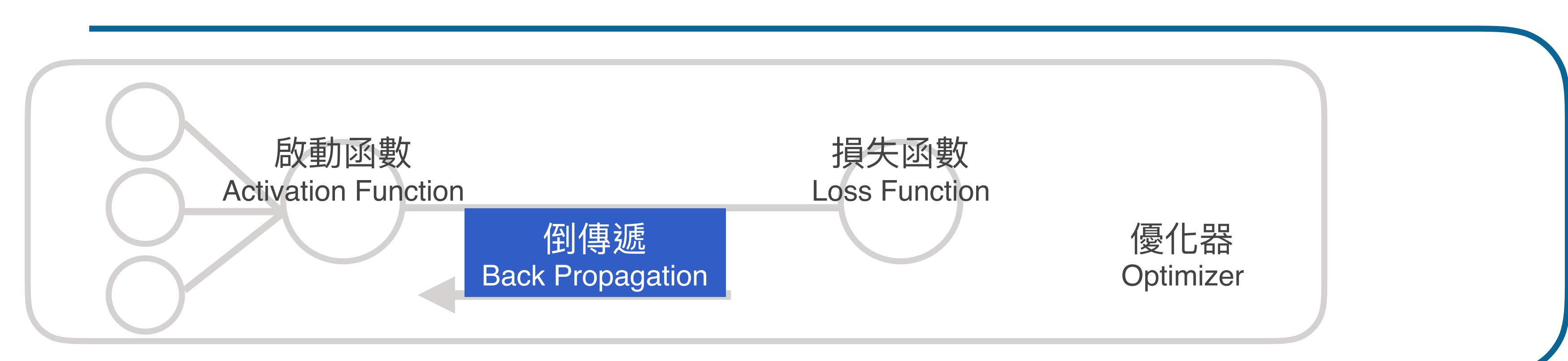
套件練習 Practice with Keras

訓練技巧 Training Skill

電腦視覺 Computer Vision

深度學習組成概念
Concept of DNN

感知器概念簡介



本日知識點目標

- 了解 Gradient Descent 的數學定義與程式樣貌

Gradient梯度

- 在微積分裡面，對多元函數的參數求 ∂ 偏導數，把求得的各個參數的偏導數以向量的形式寫出來，就是梯度。
- 比如函數 $f(x)$, 對 x 求偏導數，求得的梯度向量就是 $(\partial f / \partial x)$ ，簡稱 $\text{grad } f(x)$ 或者 $\nabla f(x)$

*Function(x)=ydata=b+w*xdata*

最常用的優化算法 - 梯度下降

- 目的：沿著目標函數梯度下降的方向搜索極小值（也可以沿著梯度上升的方向搜索極大值）
- 要計算 Gradient Descent，考慮
 - $\text{Loss} = \text{實際 } y_{\text{data}} - \text{預測 } y_{\text{data}}$
 - $= w^* \text{ 實際 } x_{\text{data}} - w^* \text{ 預測 } x_{\text{data}}$ (bias 為 init value，被消除)
 - $\text{Gradient} = \nabla f(\theta)$ ($\text{Gradient} = \partial L / \partial w$)
 - 調整後的權重 = 原權重 $- \eta$ (Learning rate) * Gradient

So,

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


(更新，每走一步更新一次)

梯度下降的算法調優

- Learning rate 選擇，實際上取值取決於數據樣本，如果損失函數在變小，說明取值有效，否則要增大 Learning rate
- 自動更新 Learning rate - 衰減因子 decay
 - 算法參數的初始值選擇。初始值不同，獲得的最小值也有可能不同，因此梯度下降求得的只是局部最小值；當然如果損失函數是凸函數則一定是最優解。
 - 學習率衰減公式¶
 - $lr_i = lr_{start} * 1.0 / (1.0 + decay * i)$
 - 其中 lr_i 為第一迭代 i 時的學習率， lr_{start} 為初始值， $decay$ 為一個介於 $[0.0, 1.0]$ 的小數。從公式上可看出：
 - $decay$ 越小，學習率衰減地越慢，當 $decay = 0$ 時，學習率保持不變
 - $decay$ 越大，學習率衰減地越快，當 $decay = 1$ 時，學習率衰減最快

梯度下降的算法調優

- 使用 momentum 是梯度下降法中一種常用的加速技術。

$x \leftarrow x - a * dx$ (x 沿負梯度方向下降)

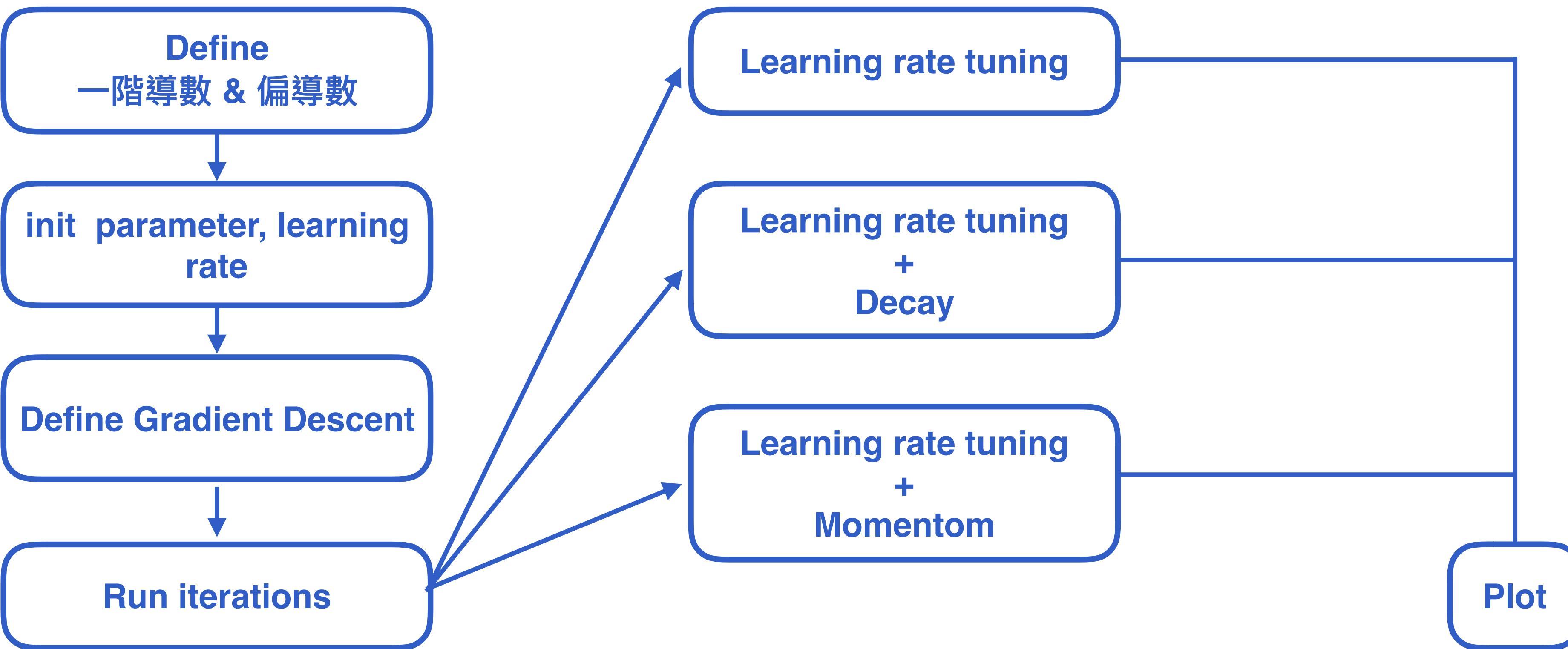
$v = \beta * v - a * dx$

$x \leftarrow x + v$

- 其中 β 即 momentum 係數，通俗的理解上面式子就是，如果上一次的 momentum (即 β) 與這一次的負梯度方向是相同的，那這次下降的幅度就會加大，所以這樣做能夠達到加速收斂的過程

前述流程 / python程式 對照

前述流程



前述流程 / python 程式 對照

python 程式 (請參閱今日範例)

```
# 目標函數:y=(x+3)^2
def func(x):
    return np.square(x+3)

# 目標函數一階導數:dy/dx=2*(x+3)
def dfunc(x):
    return 2 * (x+3)

def GD(w_init, df, epochs, lr):
    """
    梯度下降法。給定起始點與目標函數的一階導函數，求在epochs次反覆運算中x的更新值
    :param w_init: w的init value
    :param df: 目標函數的一階導函數
    :param epochs: 反覆運算週期
    :param lr: 學習率
    :return: x在每次反覆運算後的位置
    """

    xs = np.zeros(epochs+1) # 把 "epochs+1" 轉成dtype=np.float32
    x = w_init
    xs[0] = x
    for i in range(epochs):
        dx = df(x)
        # v表示x要跨出的幅度
        v = - dx * lr
        x += v
        xs[i+1] = x
    return xs
```

前述流程 / python 程式 對照

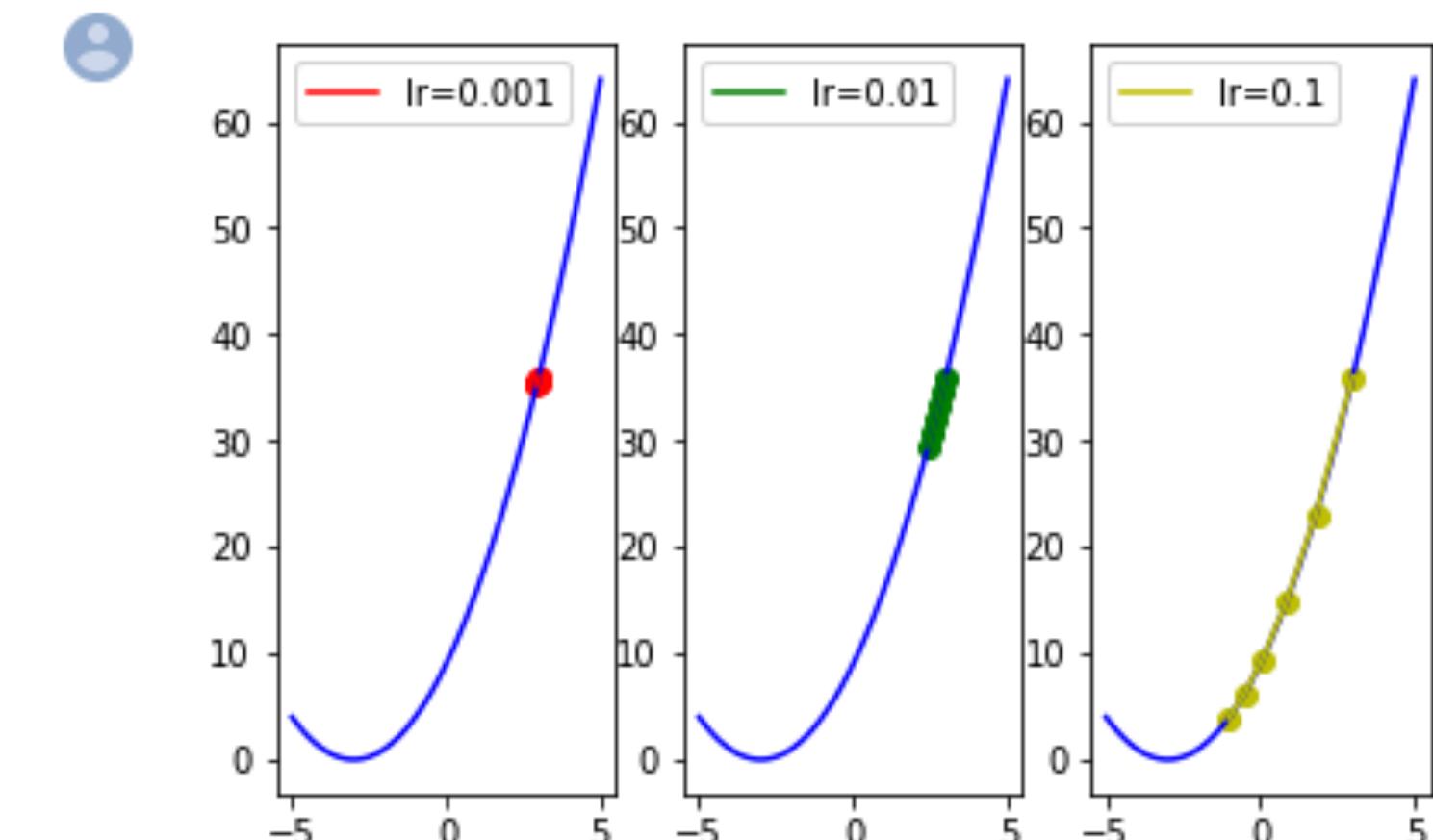
python 程式 (請參閱今日範例)

學習率較小時，收斂到正確結果的速度較慢。學習率較大時，容易在搜索過程中發生震盪。

```
[ ] line_x = np.linspace(-5, 5, 100)
line_y = func(line_x)
plt.figure('Gradient Descent: Learning Rate')

w_init = 3
epochs = 5
x = w_init
lr = [0.001, 0.01, 0.1]

color = ['r', 'g', 'y']
size = np.ones(epochs+1) * 10
size[-1] = 70
for i in range(len(lr)):
    x = GD(w_init, dfunc, epochs, lr=lr[i])
    plt.subplot(1, 3, i+1)
    plt.plot(line_x, line_y, c='b')
    plt.plot(x, func(x), c=color[i], label='lr={}'.format(lr[i]))
    plt.scatter(x, func(x), c=color[i])
    plt.legend()
plt.show()
```



前述流程 / python 程式 對照

python 程式 (請參閱今日範例)

學習率衰減公式

$$lr_i = lr_{start} * 1.0 / (1.0 + decay * i)$$

其中 lr_i 為第一迭代 i 時的學習率， lr_{start} 為原始學習率， $decay$ 為一個介於 $[0.0, 1.0]$ 的小數。從公式上可看出：

$decay$ 越小，學習率衰減地越慢，當 $decay = 0$ 時，學習率保持不變。 $decay$ 越大，學習率衰減地越快，當 $decay = 1$ 時，學習率衰減最快

```
def GD_decay(w_init, df, epochs, lr, decay):
    xs = np.zeros(epochs+1)
    x = w_init
    xs[0] = x
    v = 0
    for i in range(epochs):
        dx = df(x)
        # 學習率衰減
        lr_i = lr * 1.0 / (1.0 + decay * i)
        # v 表示 x 要改变的幅度
        v = - dx * lr_i
        x += v
        xs[i+1] = x
    return xs
```

前述流程 / python 程式 對照

python 程式 (請參閱今日範例)

Momentum (動量)

如何用“動量”來解決:

(1)學習率較小時，收斂到極值的速度較慢。

(2)學習率較大時，容易在搜索過程中發生震盪。

當使用動量時，則把每次w的更新量v考慮為本次的梯度下降量 (-dx*lr)，與上次w的更新量v乘上一個介於[0, 1]的因子momentum的和

$w \leftarrow x - \alpha * dw$ (x沿負梯度方向下降)

$v = \beta * v - \alpha * d w$

$w \leftarrow w + v$

(β 即momentum係數，通俗的理解上面式子就是，如果上一次的momentum (即 β) 與這一次的負梯度方向是相同的，那這次下降的幅度就會加大，所以這樣做能夠達到加速收斂的過程

如果上一次的momentum (即 β) 與這一次的負梯度方向是相反的，那這次下降的幅度就會縮減，所以這樣做能夠達到減速收斂的過程

```

1 def GD_momentum(w_init, df, epochs, lr, momentum):
2     xs = np.zeros(epochs+1)
3     x = w_init
4     xs[0] = x
5     v = 0
6     for i in range(epochs):
7         dx = df(x)
8         # v表示x要改變的幅度
9         v = - dx * lr + momentum * v
10        x += v
11        xs[i+1] = x
12    return xs

```

重要知識點複習：梯度下降法 (Gradient descent)

- Gradient descent 是一個一階最佳化算法，通常也稱為最速下降法。
- 要使用梯度下降法找到一個函數的局部極小值，必須向函數上當前點對應梯度（或者是近似梯度）的反方向的規定步長距離點進行疊代搜索。
 - avoid local minima
 - Item-1：在訓練神經網絡的時候，通常在訓練剛開始的時候使用較大的 learning rate，隨著訓練的進行，我們會慢慢的減小 learning rate
 - 學習率較小時，收斂到極值的速度較慢。
 - 學習率較大時，容易在搜索過程中發生震盪
 - Item-2：隨著 iteration 改變 Learning衰減越大，學習率衰減地越快。衰減確實能夠對震盪起到減緩的作用

重要知識點複習：

- avoid local minima
 - Item-3 : momentum
 - 如果上一次的 momentum 與這一次的負梯度方向是相同的，那這次下降的幅度就會加大，所以這樣做能夠達到加速收斂的過程
 - 如果上一次的 momentum 與這一次的負梯度方向是相反的，那這次下降的幅度就會縮減，所以這樣做能夠達到減速收斂的過程



延伸 閱讀

- 機器學習 - 梯度下降法 (簡體)
- gradient descent using python and numpy (英文)
- 梯度下降算法的參數更新公式 (簡體)



解題時間

It's Your Turn

請跳出PDF至官網Sample Code & 作業
開始解題

