高三數甲學習歷程

壹、 動機：

高一的時候，上了有關 AI 的課程在 AI 學習之中有一個非常重要的演算法叫做梯度下降演算法，是跟微分有關的，而在高三上的數學課我們學到了微積分，所以我想嘗試完成使用梯度下降演算法。

除此之外在函數的世界中我們常常需要找到所謂的極值，一般而言我們會用一次微分等於零，來尋找斜率為零的地方，便能推測這裡可能是此函數的極值所在。

但在現實的世界中，我們所見的函數往往不是在數學課本上看到那麼得單純而且所求的值會是很好看的數字，

例如:

我們可以假設一個基本的多變量線性回歸式

f(x)=$w_1X_1 + w_2xX_2 + \cdots + X_ix_i + b\cdots(1)$

接著我們計算均方差。

MSE=$\frac{1}{n}\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2\cdots(2)$

接著我們將 (1) 代入 (2) 式，以求均方差最小值為目標，試圖找到該式的最佳解。

這時新的方均差如下:

MSE=$\frac{1}{n}\sum_{i=1}^{n}\left(w_1X_1 + w_2X_2 + \cdots + w_iX_i + b - \overline{X}\right)^2$

這複雜的算式其實在人工智慧的領域非常重要，但當我們面對層層網路疊加下所產出生的損失函數會非常的麻煩。

所以區域最佳化算法的存在意義就在這裡，全域解計算量大，費時費力，只好以迭代區域解的方式來近似全域解，因此我想找方法可以來完成這件事。

貳、 目的

1. 了解梯度下降演算法
2. 探討跌代次數對梯度下降演算法之影響
3. 探討學習率梯度下降演算法之影響
4. 探討平面梯度下降演算法
5. 用梯度下降演算法預測回歸直線方程式

參、 探討過程

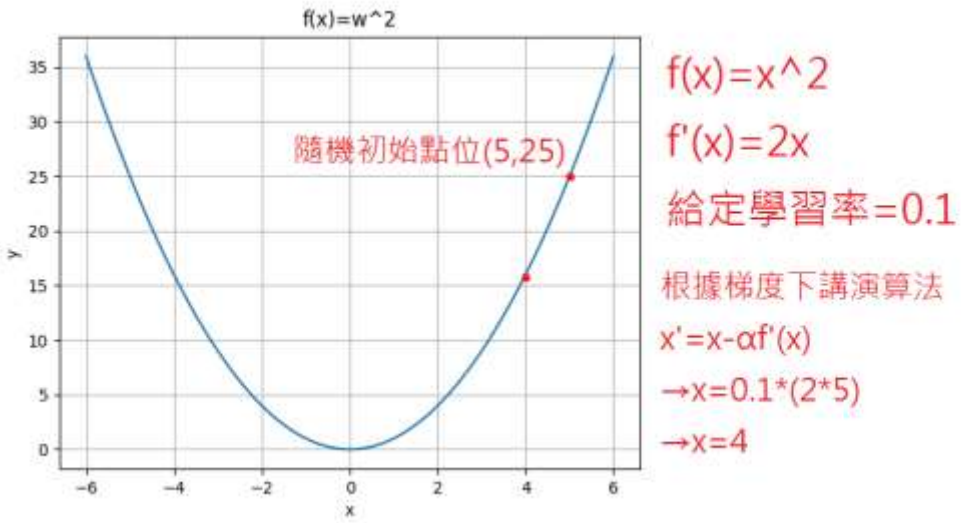
梯度下降演算法(gradient descent)

又名為：批梯度下降法(batch gradient descent, BGD)：
梯度下降法的定義是：
$w^{j+1} = w^j - \alpha \nabla E(w)$
其中 ∇就是梯度，一次微分的意思。
若是單純的 y=ax+b 的情況，一次微分就是斜率也可以說他為梯度。

基本型態：

首先我們給定一個二次函數，$f(x)=x^2$，二次函數是比較常見的數學模型，且長得較簡單，可以來當我初探的函數。
(以下的圖都為 python Matplotlib 所繪)
假定，第一次隨機初始化點位，x=5，學習率為 0.1，且迭代 11 次。

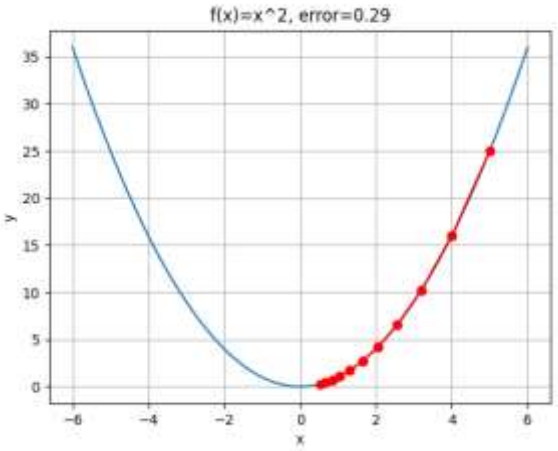
第一次迭代：x 從 5 移動至 4



在初始化點位 (5, 25) 上，經微分計算後得到梯度，為 +10，這會影響以下兩件事情：

方向：將梯度的方向取負號，就是我們想要移動的方向。
大小：由於學習率固定，因此梯度值愈大，每次移動的距離愈遠！
第二次迭代：x 從 4 移動至 3.2
第三次迭代：x 從 3.2 移動至 2.56
這個反覆迭代的過程會一直到終止條件出現為止，例如：

1.迭代次數達到某個值。
2.迭代後的 loss 值間的差異小於某個數。
3.程式執行總時間限制。
最後以下為成果圖和程式碼:



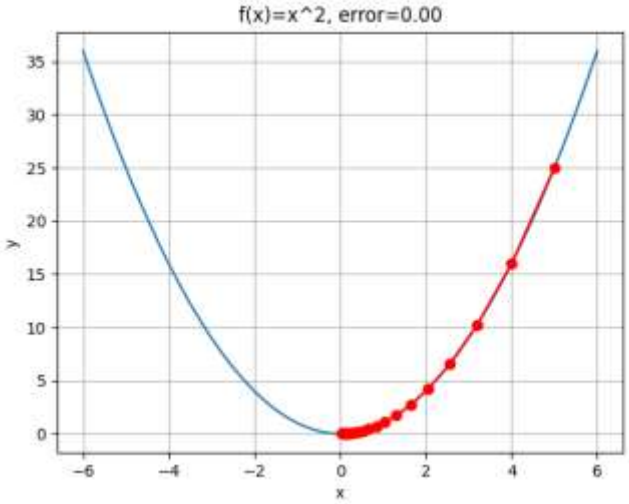
程式碼：

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

# 學習率
alpha = 0.2

# 初始位置
x_init = 5
fig=plt.figure()
x = np.arange(-6, 6.1, 0.1)
y = x**2
plt.plot(x, y)

plt.xlabel('x')
plt.ylabel('y')
plt.title('f(x)=w^2')
plt.grid(True)

x_old = 0; x_new = 0
x_old = x_init
i=0
while (2*x_old)>0.1:
    if i>=1:
        x_old = x_new
    if i ==10:
        break
    i=i+1
    # (2*w_old) 為二次函數的的微分式
    x_new = x_old - alpha * (2*x_old)

    plt.plot((x_old, x_new), (x_old**2, x_new**2), 'ro-')
    plt.title('f(x)=x^2, α=' + str(alpha))
print(x_old)
plt.show()

fig.savefig('α=0.2.png')
```

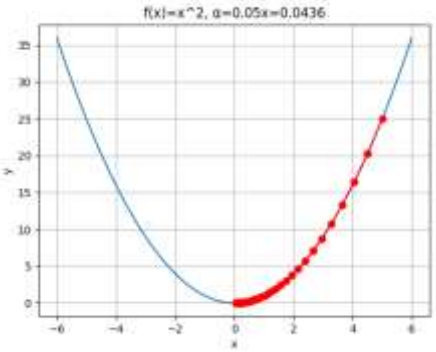
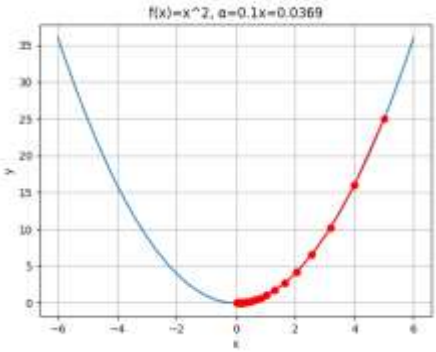
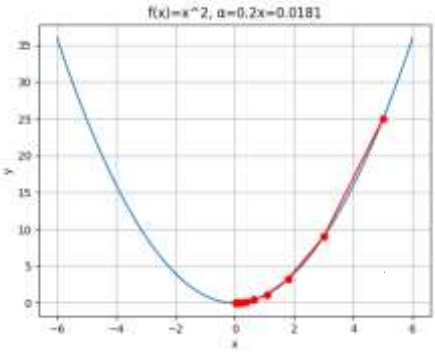
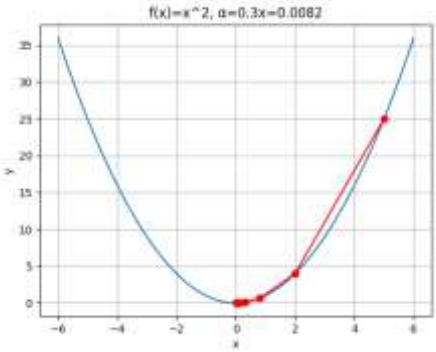
增加跌代停止條件

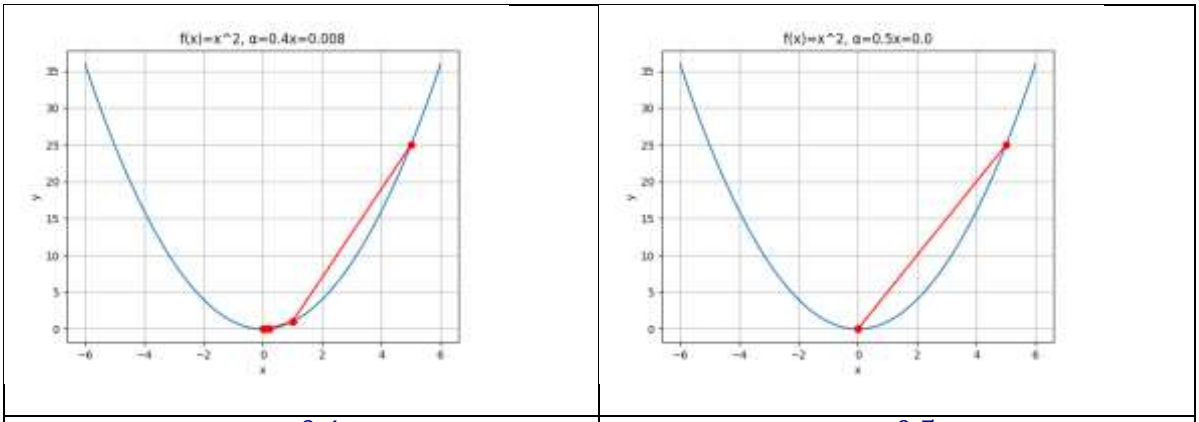
從上圖可知我覺得距離極植還是差的有點多，於是我增加迭代次數，試圖找到停止條件，於是我把條件從原本的 11 次改為梯度小於 0.1

從右圖可知我們增加跌代條件可以看到更靠近極值且 error 更靠近 0，代表更加有效



探討不同學習率對尋找極植之影響：

	
$\alpha = 0.05$	$\alpha = 0.1$
	
$\alpha = 0.2$	$\alpha = 0.3$

	
$\alpha = 0.4$	$\alpha = 0.5$

我發現當學習率越低不一定代表越好，因為我的迭代停止條件設定為梯度小於一的時候，學習率越低代表每次步幅越小，使得最終得到的值會越接近梯度等於 0.1 時的條件。

結論:

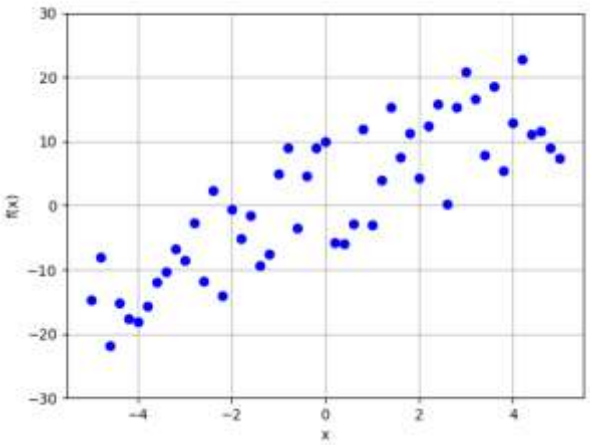
在限定梯度的情況下學習率不是越小越好，學習率的選擇不是越小越好，而是與跌代終止條件有關。

梯度下降演算法的實際運用

我們就先隨機產生一組數據，再來做線性回歸

我們先假定方程式為 y=3x+2

一開始先給定一個固定的參數，再加上隨機變動值作為雜訊其雜訊變動值介於 -10 ～ 10 之間，產生 51 個資料點，經由程式自動產生，可得右圖。



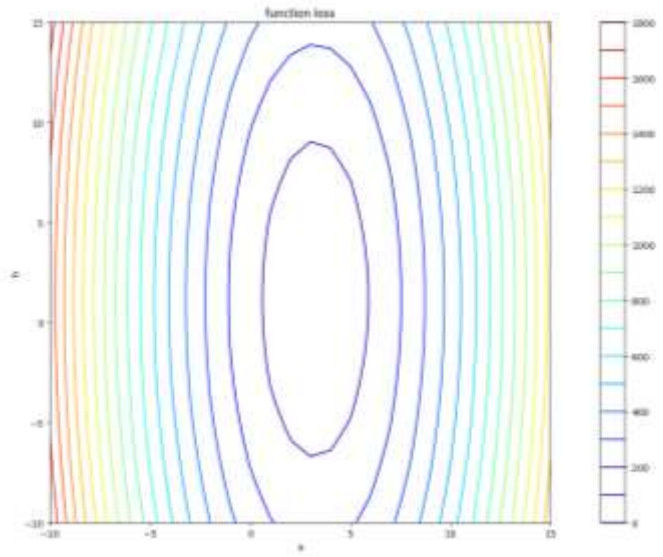
為了要對這組數據做線性回歸，先給定一個線性回歸式：f(x)=ax+b，x 就是我們所持有的資料，f(x)是模型的預測值。至於 a 和 b 則是待計算的參數。

接著設一個計算平均絕對差的損失函數，如此一來才能得到梯度下降的方向：

MAE $= \frac{1}{n}\sum_{i=1}^{n}|e_i|$ e_i是絕對誤差 $= |f_i - y_i|$

接著我們把預測模型帶入損失函數得到 MAE(a,b) = $\frac{1}{n}\sum_{i=1}^{n}|ax + b - y_i|$

我們可以先對 MAE 製圖
根據以下圖片可以發現最小值約落在
(a,b)=(3,2)，符合預期。



我們知道梯度下降公式為
$w^{j+1} = w^j - \alpha \nabla E(w)$

因為 MAE 函數為絕對值函數，我們知道不好微分，於是我先把它平方所以新的 Loss 函數為$\frac{1}{n}\sum_{i=1}^{n}(ax+b-y_i)^2$這時我們得到現在的目標改成 $\mathrm{Loss(a,b)}$，所以要分別計算 a 和 b 的梯度。$\nabla Loss(a,b) = \left(\frac{\partial Loss(a,b)}{\partial a}, \frac{\partial Loss(a,b)}{\partial b}\right)$

$$\frac{\partial\ \mathrm{Loss(a,\ b)}}{\partial \mathrm{a}} = \frac{2}{\mathrm{n}}\sum_{\mathrm{i=1}}^{\mathrm{n}}\left(\mathrm{ax^i + b - Y_t^i}\right) \times \left(\mathrm{x^i}\right)$$

$$\frac{\partial\ \mathrm{Loss(a,\ b)}}{\partial \mathrm{b}} = \frac{2}{\mathrm{n}}\sum_{\mathrm{i=1}}^{\mathrm{n}}\left(\mathrm{ax^i + b - Y_t^i}\right) \times (1)$$

於是可以把梯度寫成以下程式

```
def gradient(a, b, x, y):
    grad_a = 2 * np.mean((a*x + b - y) * (x))
    grad_b = 2 * np.mean((a*x + b - y) * (1))
    return grad_a, grad_b
```

有了梯度函數，我們還需要一個更新函數與均方差計算：

```
def update(a, b, grad_a, grad_b, alpha):
    a_new = a - alpha * grad_a
    b_new = b - alpha * grad_b
    return a_new, b_new
def mse(a, b, x, y):
    sqr_err = ((a*x + b) - y) ** 2
    loss = np.mean(sqr_err)
    return loss
```

最後經過程式(程式碼見附件)運算我們可以得到直線方程式為
y=3.238425711869309x+1.1639791894237836，我們取有效數字四位得到
Y=3.2385x+1.164 與我們用 excel 做的線性回歸 y = 3.238x + 1.164 完全一樣，故本方法有效，我們可以從附件中的 GIF 檔看到，在跌代次數到達 22 次之後所繪製出的回歸直線已無太大變動，可能是已經離 LOSS 最小值很接近。

肆、 結論

梯度下降演算法的精神在於經過很多運算得到一個最優解，而在最憂解產生之前我們必須先產生損失函數，損失函數 loss function 只能告訴我們參數的好壞，並不能直接給我們答案，因為我們通常不可能把所有的 loss 都計算出來，我們頂多知道要評估哪個點時，才會去計算那一個點的 loss，例如說我們可能只知道像上圖的兩個點。所以當我們計算第一個點的時候，我們仍然要知道接下來我們要往哪裡移動，才能找到最小值。於是運用到我們在高三數甲所學到的微分概念，透過微分來找尋梯度下降演算法的方向，再藉由調整學習率來控制步幅，因此梯度下降演算法是可以找到最憂解的一種方法。

伍、 心得

在這次學習之梯度下降演算法之中一開始只有用到一班的方程式微分，並且一開始所用到的只有簡單的二次函數圖形找最低點，但到後面隨機亂數產生，要來預測此數據的回歸直線方程式，因為直線方程式是 y=ax+b，所以我們有 a 跟 b 兩個變數要處理，所以延伸到平面方向的進展，而我所設計出的損失函數代有兩個未知數的方程式，所以一般的方程式微分並不能解決這問題，所以我用到的偏微分來計算梯度，也就是我們在梯度下降演算法所類比的方向，在這次學習完梯度下降演算法之後我解決了在高一時未解開的問題，並且從課本中的知識延伸出來，把微分學以致用。

陸、 反思

對於梯度下降演算法我還有許多疑問?

1. 在函數探討極值時，我們怎麼知道所探詢到的一定是極值，例如四次函數可能會出現兩次極值，但根據初始點味的不同，可能會掉入較大的局部極值，而非全域極值。
2. 在靠近局部極小值時速度減慢，若資料增加至 10 萬筆的時候會非常的緩慢
3. 直線搜索可能會產生一些問題。
4. 如果有一個地方是無法辨別方向，例如鞍點

以上的問題都值得思考是我以後可以繼續學習的。

柒、 附件與參考資料

附件: https://github.com/TingYueLai/Gradient-descent-

參考資料:

1. **ML Lecture 3-1: Gradient Descent**
2. **An overview of gradient descent optimization algorithms**
3. **SGD 算法比較**
4. **深度學習優化器總結**
5. **自己动手用 python 写梯度下降**
6. **深入浅出--梯度下降法及其实现**