機器學習 HW1

班級:電機三甲
姓名:林士恩
學號:B043011031

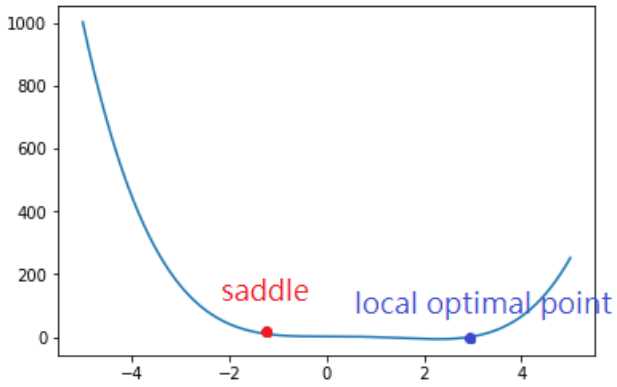
一、完整程式碼:

```
#import the packages
import numpy as np
import matplotlib.pyplot as plt
#畫出原f(theta)方程式，從-5到5之間，間距為0.0001
theta = np.arange(-5.0, 5.0, 10**-4)
func = theta**4 - 3*theta**3 + 2
plt.plot(theta, func)
plt.show()
#產生隨機的初始位置
currentPosition = np.random.uniform(high = 5.0, low = -5.0, size = None)
currentPosition = 0.04
initialPosition = currentPosition
epsilon = 10**-10   #設定最小優化區間
lambdA = 10**-5    #設定lambda大小
ValueOfFunction = currentPosition**4 - 3*currentPosition**3 + 2 #利用f'(theta)去尋找某範圍中的最低點(saddle)
trackOfValue = [ValueOfFunction]                                #並利用Tuple紀錄函數值
finding = False
#利用whhile迴圈去不斷地逼近saddle，並到了一定的位置後會自動停止，達到early stop的效果
while not finding:
    #利用 x2 = x1 - f'(theta) * lambda的公式去逼近saddle
    currentPosition = currentPosition - lambdA * (4 * currentPosition ** 3 - 9 * currentPosition ** 2)
    ValueOfFunction = currentPosition ** 4 - 3 * currentPosition ** 3 + 2
    trackOfValue.append(ValueOfFunction) #函數值加到Tuple中
    if trackOfValue[-2] - trackOfValue[-1] < epsilon:   #當位置與位置間的差距比epsilon還小時就停止逼近
        print("Early Stop")
        finding = True


print("Start at %s" %initialPosition)
print("End at %s" %currentPosition)
print("With %s approaching" %len(trackOfValue))
print("The local optimal is 2.25")
#畫出函數值的變化
plt.plot(np.arange(len(trackOfValue)),trackOfValue)
plt.show()
```

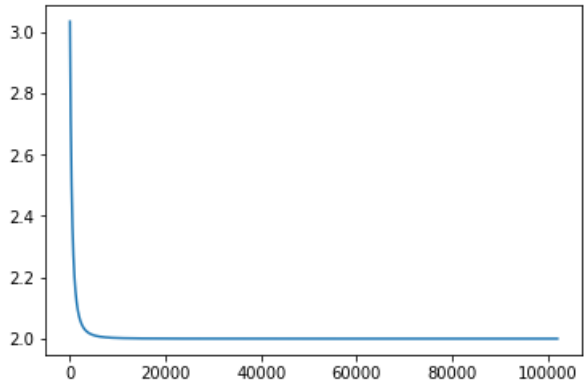
更正:tuple -> list

Console:



此圖為 f(Θ)的函數圖，Θ 區間為-5~5

```
Early Stop
Start at -0.6564958841549586
End at -0.010516198909156083
With 102044 approaching
The local optimal is 2.25
```



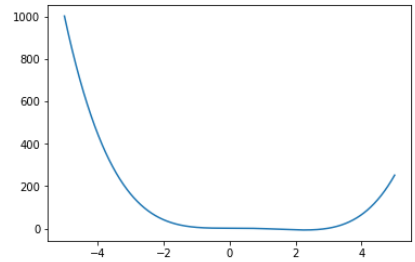
此圖為尋找 saddle 或 optimal point 的函數值變化圖

二、問題討論:

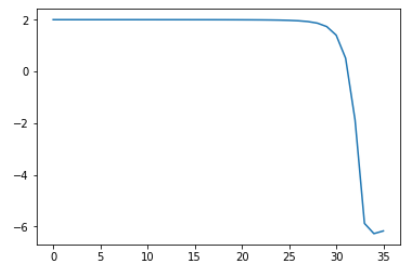
1. 在 Early Stop 之下，lambda 值對於收斂過程的影響:

起點都同樣設定在 0.04

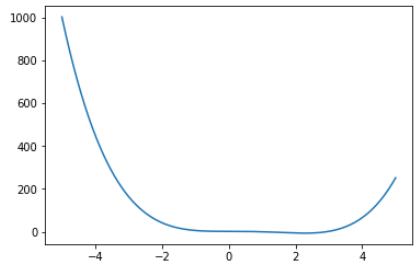
lambda = 0.01



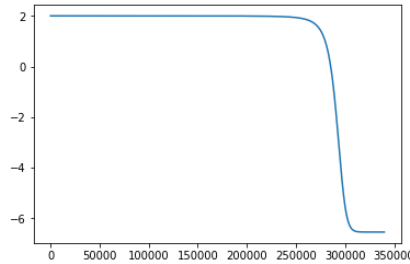
```
Early Stop
Start at 0.04
End at 2.047311642919764
With 36 approaching
The local optimal is 2.25
```



lambda = 0.00001



```
Early Stop
Start at 0.04
End at 2.2498438564008825
With 339938 approaching
The local optimal is 2.25
```



從迴圈進行次數中，可以發現當 lambda 較大時，能較快的找到 optimal point(但也可能會發散)；反之，lambda 較小時會比較慢收斂。但是；也可以發現 lambda 大時收斂後的最後位置會較不精確，lambda 小時較接近 optimal point。

2. Early Stop 機制對於求解過程的影響:

在起始位置為 0.04 之下

如果將 Early Stop statement 給關掉，如下圖

```
while not finding:
    #利用 x2 = x1 - f(theta) * lambda的公式去逼近saddle
    currentPosition = currentPosition - lambdA * (4 * currentPosition ** 3 - 9 * currentPosition ** 2)
    ValueOfFunction = currentPosition ** 4 - 3 * currentPosition ** 3 + 2
    trackOfValue.append(ValueOfFunction) #函數值加到Tuple中
    #if trackOfValue[-2] - trackOfValue[-1] < epsilon:   #當位置與位置間的差距比epsilon還小時就停止逼近
    #    print("Early Stop")
    #    finding = True
    print("%s approaching" %len(trackOfValue))
```

接著再跑程式，會發現程式會一直進行，即不斷地逼近，沒辦法順利收斂

```
136411 approaching
136412 approaching
136413 approaching
136414 approaching
136415 approaching
136416 approaching
136417 approaching
136418 approaching
136419 approaching
136420 approaching
136421 approaching
136422 approaching
136423 approaching
136424 approaching
136425 approaching
136426 approaching
136427 approaching
136428 approaching
136429 approaching
136430 approaching
136431 approaching
136432 approaching
136433 approaching
136434 approaching
136435 approaching
136436 approaching
136437 approaching
136438 approaching
136439 approaching
136440 approaching
136441 approaching
136442 approaching
136443 approaching
136444 approachingTraceback (most recent call last):
```

若再將 Early Stop 打開，會發現其實只需要 34000 次 iterations 就逼近 optimal point 了

```
Early Stop
Start at 0.04
End at 2.2498438564008825
With 339938 approaching
The local optimal is 2.25
```

會發生這種問題是因為運算過程中 f’(Θ)會愈來愈小(往 f’(Θ)=0)，且 lambda 不變，造成 x2 = x1 - lambda * f’(x1) 移動的距離隨著越靠近 f’(Θ)而越來越小而沒辦法真正落在 optimal point 上。

當起始位置設置為 2.00 時，且沒有 Early Stop 時，可以發現越接近 optimal point，斜率不斷向 0 靠近，且幅度增加幅度越來越小

```
Now at 2.232809251454l664
f'(theta) : -0.3427462754218382
Now at 2.232812679590026
f'(theta) : -0.3426789776626151
Now at 2.23281610705278
f'(theta) : -0.34261169269927905
Now at 2.232819533842557
f'(theta) : -0.34254442052954914
Now at 2.232822959959484
f'(theta) : -0.342477161151173
Now at 2.232826385403689
f'(theta) : -0.34240991456186975
Now at 2.232829810175301
f'(theta) : -0.342342680759387
Now at 2.2328332342744464
f'(theta) : -0.3422754597414439
Now at 2.232836657701254
f'(theta) : -0.34220825150578094
Now at 2.2328400804558517
f'(theta) : -0.34214105605013856
Now at 2.232843502538367
f'(theta) : -0.34207387337225015
Now at 2.2328469239489275
f'(theta) : -0.34200670346984197
Now at 2.232850344687661
f'(theta) : -0.3419395463406616
Now at 2.232853764754696
f'(theta) : -0.3418724019824424
Now at 2.232857184150159
f'(theta) : -0.34180527039292485
Now at 2.232860602874179
f'(theta) : -0.3417381515698281
Now at 2.232864020926883
f'(theta) : -0.3416710455108998
```

3. 如何使用 GD 方法來收斂於 Global Optimal

利用 recursion 的概念修改程式碼，假設此 f(x)函數的區間大小為 n，則把區間分成 1/2，之後繼續分成 1/2，當區間小到個程度時再尋找此區間內的 local minimum，最後比較各個區間內的 local minimum，找出其中最小值即為 global optimal。

```
36 def findGlobalMinimum(left, right, trackOfValues) : #trackOfValues用來儲存所有local minimum，最後只要在其中找到最小值就好
37     if((right - left) > 0.0001) :                  #right, left個代表區間的上下限
38         #分成兩半去進行同個function
39         findGlobalMinimum(left, (left + right) / 2, trackOfValues)
40         findGlobalMinimum((left + right) / 2, right, trackOfValues)
41     else :
42         current = np.random.uniform(high = right, low = left, size = None)
43         epsilon = 10**-10
44         lambdA = 10**-5
45         Value = current**4 - 3*current**3 + 2 #利用f'(theta)去尋找某範圍中的最低點(saddle)
46         trackOfValues.append(Value)
47         finding = False
48         while not finding:
49             #利用 x2 = x1 - f(theta) * lambda的公式去逼近saddle
50             current = current- lambdA * (4 * current ** 3 - 9 * current ** 2)
51             ValueOfFunc = current ** 4 - 3 * current ** 3 + 2
52             trackOfValues.append(ValueOfFunc) #函數值加到list中
53             if trackOfValues[-2] - trackOfValues[-1] < epsilon:   #當位置與位置間的差距比epsilon還小時就停止逼近
54                 print("Early Stop")
55                 finding = True
56
```

最後只要在 trackOfValues 中找出最小值，就是 global optimal。

時間複雜度: T(n) = T(n/2) + a ， 其中 a 的時間由 line 48 開始的 while 迴圈 dominate