Homework 1

Implementation of the Gradient Descent Method

學號：404410030
系級：資工系大學部三年級
姓名：鄭光宇

環境設置

使用 Python 中的 jupyter notebook 來完成梯度下降的實驗。

主要使用 Python 中的 numpy 實作梯度下降法 (Gradient Descent Method)
並用 matplotlib 的視覺化工具將梯度下降的過程做成動畫、繪製圖表。

```
1 python 3.6
2 matplotlib
3 numpy
```

實作的程式碼檔名為 bs_n.ipynb 。

實驗的目標函數

要最小化的函數有三個：

1. $f(x) = x^4 - 3x^2 + 2$
2. $f(x_1, x_2) = 100(x_2 - x_1)^2 + (1 - x_1)^2$
3. $f(x_1, x_2) = sin(x_1^2 + x_2^2)$

三個函數分別在程式中命名為： fx, rosenbrock, basin

初始點、實驗設定

三個函數的初始點分別為：

1. $x = 0.05$
2. $x_1 = 100, x_2 = -80$
3. $x_1 = 0.9, x_2 = -0.8$

為了比較三個函數在使用不同類型的梯度下降法的收斂情形，我設定每個函數都迭代固定的次數。三個函數使用梯度下降法的迭代次數分別為：

1. 1300 次
2. 200 次
3. 1800 次

總結之，這次實驗有以下設定：

函數	初始點	迭代次數
$f(x) = x^4 - 3x^2 + 2$	$x = 0.05$	1300
$f(x_1, x_2) = 100(x_2 - x_1)^2 + (1 - x_1)^2$	$x_1 = 100, x_2 = -80$	200
$f(x_1, x_2) = sin(x_1^2 + x_2^2)$	$x_1 = 0.9, x_2 = -0.8$	1800

實作細節

為了更準確地計算梯度，我先推出三個函數對每各自每個變數的偏微分公式，這樣就可以直接求出每個函數給定變數值時的梯度方向、大小。

1. $\frac{\partial f}{\partial x} = 4x^3 - 6x$
2. $\frac{\partial f}{\partial x_1} = 202x_1 - 200x_2 - 2$
   $\frac{\partial f}{\partial x_1} = -200(x_1 - x_2)$
3. $\frac{\partial f}{\partial x_1} = 2x_1 cos(x_1^2 + x_2^2)$
   $\frac{\partial f}{\partial x_2} = 2x_2 cos(x_1^2 + x_2^2)$

這樣，當我們需要計算梯度時，只需要帶入變數值就可以求得。

在這次作業中，實作了三種梯度下降法，分別是一般的梯度下降法、Adam、Adagrad。三個方法實作出三個 class，三個 class 都有一個 update member function，輸入是變數值與函數在給定變數值時求得的梯度，會根據輸入的變數與梯度去更新變數值。

首先是一般的梯度下降法，固定更新步長為 0.001：

```
class Gradient_Descent(object):
    def __init__(self, lr=0.001, decay_rate=0.9, epsilon=1e-8):
        """
        x -= alpha * x_gradient
        """
        self.lr = lr
    def update(self, params, grads):
        f_param = params.ravel()
        f_grad  = grads.ravel()
        for i, (x, dx) in enumerate(zip(f_param, f_grad)):
            f_param[i] -= self.lr * dx
```

接下來是可以自適應調節 learning rate 的 Adagrad， learning rate 設為 0.001：

```
class Adagrad(object):
    def __init__(self, lr=0.001, decay_rate=0.9, epsilon=1e-8):
        """
        Ref from CS231n:
        http://cs231n.github.io/neural-networks-3
        cache = decay_rate * cache + (1 - decay_rate) * dx**2
        x += - learning_rate * dx / (np.sqrt(cache) + eps)
        """
```

```
        self.lr = lr
        self.decay = decay_rate
        self.eps = epsilon
    def update(self, params, grads):
        f_param = params.ravel()
        f_grad  = grads.ravel()
        if not hasattr(self, 'cache'):
            self.cache = np.zeros_like(f_param)
        for i, (x, dx, c) in enumerate(zip(f_param, f_grad, self.cache)):
            # Evaluate:
            c_t = self.decay * c + (1 - self.decay) * dx**2

            # Update:
            f_param[i] -= self.lr * dx / (np.sqrt(c_t) + self.eps)
            self.cache[i] = c_t # update cache
```

與 Adagrad 類似的 Adam，使用 learning rate 0.001 ，其他參數參考自它的 Paper：

```
class Adam(object):
    def __init__(self, lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-8):
        """
        Use recommended parameters from paper of Adam:
            -- https://arxiv.org/abs/1412.6980
        """
        self.lr = lr
        self.beta_1 = beta_1
        self.beta_2 = beta_2
        self.eps = epsilon
        self.iter = 1
    def update(self, params, grads):
        f_param = params.ravel()
        f_grad  = grads.ravel()
        if not hasattr(self, 'ms'):
            self.ms = np.zeros_like(f_param)
            self.vs = np.zeros_like(f_param)
        for i, (x, dx, m, v) in enumerate(zip(f_param, f_grad, self.ms, self.vs)):
            # Evaluate:
            m = self.beta_1*m + (1-self.beta_1)*dx # m_t = b1*m_t-1 + (1-b1)*g
            mt = m / (1-self.beta_1**self.iter) # m_t_h = m_t / (1-b1^t)
            v = self.beta_2*v + (1-self.beta_2)*(dx**2) # v_t = b2*v_t-1 + (1-b2)*g^2
            vt = v / (1-self.beta_2**self.iter) # v_t_h = v_t / (1-b2^t)

            # Update:
            f_param[i] -= self.lr * mt / (np.sqrt(vt) + self.eps) # theta = -lr * m_t_h / (sqrt(v_t_h) + eps)
            self.ms[i] = m # write m_t to memory (update from m_t-1 to m_t)
            self.vs[i] = v # write v_t to memory (update from v_t-1 to v_t)
        self.iter += 1
```

實驗結果

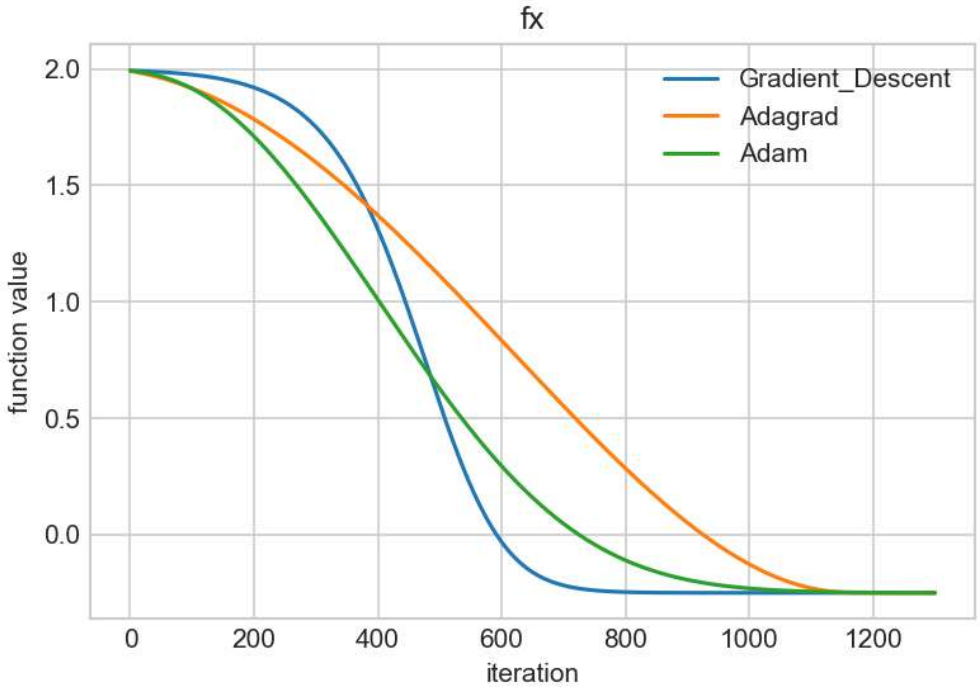
每一個函數以不同梯度下降法最佳化的過程，也就是梯度下降時，每個 iteration 的變數值與函數值，會被存放在 logs 資料夾中。

梯度下降過程的動畫，存放在 visualization 資料夾。

實驗結果（一）

對於第一個函數（$f(x) = x^4 - 3x^2 + 2$），三種方法得到的 local minimizer 如下：

方法	變數值	函數值
Gradient Descent	1.224685	-0.250000
Adagrad	1.224338	-0.249999
Adam	1.217681	-0.249702



比較三種方法，可以發現自適應調節 learning rate 的方法 (Adagrad, Adam) 前期下降的速度很快，但是最後收斂到的 local minimizer 沒有比一般的 Gradient Descent 方法好。

每種方法的梯度下降過程動畫可見：

```
./visualization/Gradient_Descent_fx.mp4
./visualization/Adagrad_fx.mp4
./visualization/Adam_fx.mp4
```

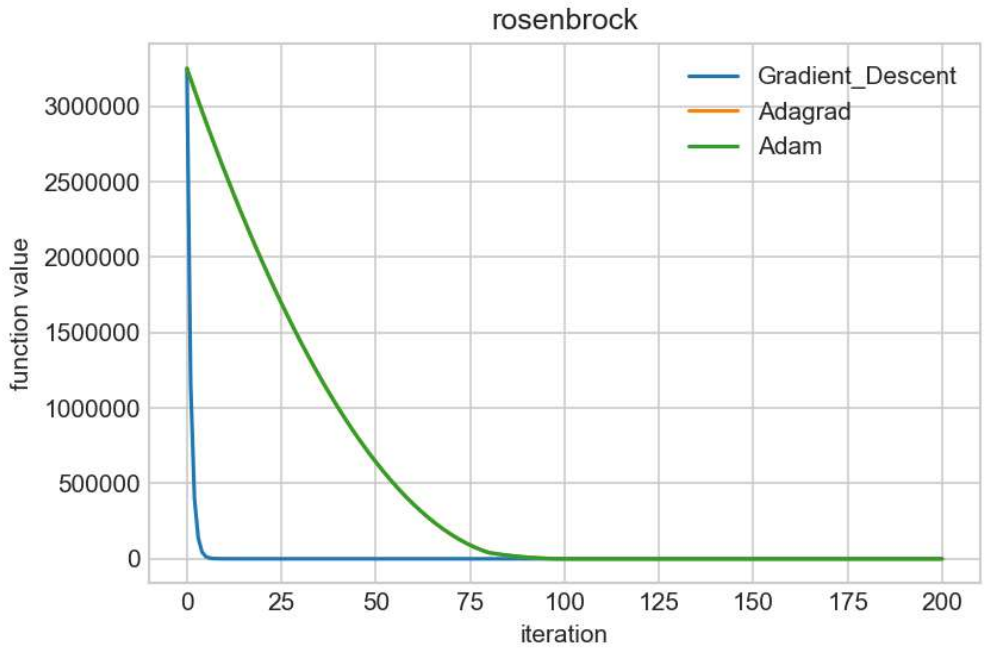
完整過程記錄檔可見：

```
./logs/Gradient_Descent_fx_log.csv
./logs/Adagrad_fx_log.csv
./logs/Adam_fx_log.csv
```

實驗結果（二）

對於第二個函數（$f(x_1, x_2) = 100(x_2 - x_1)^2 + (1 - x_1)^2$），三種方法得到的 local minimizer 如下：

方法	變數值	函數值
Gradient Descent	[1, 1]	0.000000
Adagrad	[0, 0]	1.000000
Adam	[0, 0]	1.000000



比較三種方法，一般的 Gradient Descent 方法效果不論哪一方面都較好。

每種方法的梯度下降過程動畫可見：

```
./visualization/Gradient_Descent_rosenbrock.mp4
./visualization/Adagrad_rosenbrock.mp4
./visualization/Adam_rosenbrock.mp4
```

完整過程記錄檔可見：

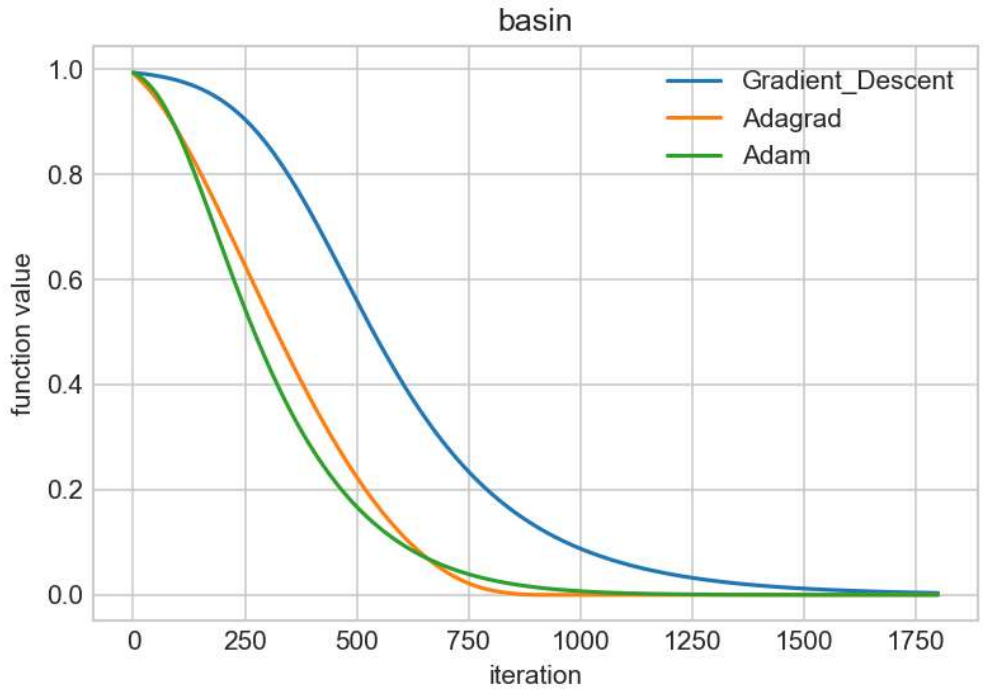
```
./logs/Gradient_Descent_rosenbrock_log.csv
./logs/Adagrad_rosenbrock_log.csv
./logs/Adam_rosenbrock_log.csv
```

實驗結果（三）

對於第三個函數（$f(x_1, x_2) = sin(x_1^2 + x_2^2)$），三種方法得到的 local minimizer 如下：

方法	變數值	函數值
Gradient Descent	[0.044746, -0.039774]	0.003584
Adagrad	[0.0005, 0.0005]	0.000000

方法	變數值	函數值
Adam	[0.001564, -0.000253]	0.000003



比較三個方法， Adam 的收斂速度前期較快，但是最後得到的 local minimizer 並不好，而 Adagrad 最後最早收斂，並且在三個方法中，有最好的 local minimizer。

但是觀察 ./logs/Adagrad_basin_log.csv 的內容可以發現，Adagrad 方法下降到接近 local minimum 的地方會有震盪的情形，這是一個問題。

```
0.000500, 0.000500, 0.000000
-0.000500, -0.000500, 0.000000
0.000500, 0.000500, 0.000000
-0.000500, -0.000500, 0.000000
0.000500, 0.000500, 0.000000
-0.000500, -0.000500, 0.000000
0.000500, 0.000500, 0.000000
-0.000500, -0.000500, 0.000000
0.000500, 0.000500, 0.000000
-0.000500, -0.000500, 0.000000
0.000500, 0.000500, 0.000000
-0.000500, -0.000500, 0.000000
0.000500, 0.000500, 0.000000
-0.000500, -0.000500, 0.000000
0.000500, 0.000500, 0.000000
```

每種方法的梯度下降過程動畫可見：

```
1 ./visualization/Gradient_Descent_basin.mp4
2 ./visualization/Adagrad_basin.mp4
3 ./visualization/Adam_basin.mp4
```

完整過程記錄檔可見：

```
./logs/Gradient_Descent_basin_log.csv
./logs/Adagrad_basin_log.csv
./logs/Adam_basin_log.csv
```