Homework 3. Gradient Descent

Double Click here to edit this cell

- Name: 김동규
- Student ID: 201800615
- Submission date: 2023/05/05

You must run this homework code on Google Colab

- DON'T run on Google Colab Pro
- DON'T use GPU or TPU

You must run the following two cells to make sure you are running on Google Colab

In [ ]:

```
!cat /proc/cpuinfo
```

In [ ]:

```
!cat /proc/meminfo
```

Remark: gradient_descent.py, linear_algebra.py must be in the folder having this notebook file

In [1]:

```
# run this cell
from gradient_descent import *
from linear_algebra import *
```

Problem 1 (5 pts)

- The following function has a minimum at $(2,3)$

$$f(x_1,x_2) = (x_1-2)^2 + (x_2-3)^2$$

- We want to compute the minimum of f using the gradient descent algorithm

- Define a function (`f`) and gradient of function (`f_gradient`)
- **Do NOT use numpy functions to define f and f_gradient**
- **USE functions in linear_algebra.py** ### **You write two functions. Each function must have ONE line of code in function body; Otherwise, you get zero point (0점)**

In [2]:

```
# YOUR CODE MUST BE HERE
def f(init_x):
    return sum_of_squares(vector_subtract(init_x,[2.,3.]))
```

```
def f_gradient(init_x):
    return 2 * vector_subtract(init_x,[2.,3.])
```

In [3]:

```
# DO NOT EDIT THIS CELL
# RUN THIS CELL

init_x = [0.,0.]
%time solution = minimize_batch(f, f_gradient, init_x, tolerance=0.00001)

### correctness check
print('solution is {}'.format(solution))
EPSILON = 0.01
cond1 = math.fabs(solution[0] - 2.0) <= EPSILON
cond2 = math.fabs(solution[1] - 3.0) <= EPSILON
assert  all([cond1, cond2]), '-'*10 + ' Problem 1 check failed ' + '-'*10
print('+'*10 + ' Problem 1 check passed ' + '+'*10)
```

```
Wall time: 0 ns
solution is [2.0, 3.0]
++++++++++ Problem 1 check passed ++++++++++
```

Problem 2 (10 pts)

- The centroid of a finite set of k points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$ in $\mathbb{R}^n$ is

$$\mathbf{C} = \frac{\mathbf{x}_1 + \mathbf{x}_2 + \cdots + \mathbf{x}_k}{k}$$

- We want to compute a centroid by **minimizing the mean of squared Euclidean distances between itself and each point in the set**

$$x_{\text{centroid}} = \text{argmin}_{\text{c}} \frac{\sum_{i=1}^{n} d(\text{c}, x_i)^2}{n}$$

- Define a function (`sq_dist`) and gradient of function (`sq_dist_gradient`)
- **Do NOT use numpy functions to define sq_dist and sq_dist_gradient**
- **USE functions in linear_algebra.py** ### **You write two functions. Each function must have ONE line of code in function body; Otherwise, you get zero point (0점)**

In [4]:

```
# YOUR CODE MUST BE HERE
def sq_dist(center,X):
    return sum(sum_of_squares(vector_subtract(center,x)) for x in X)
def sq_dist_gradient(center,X):
    return vector_sum(scalar_multiply(2, vector_subtract(center,x)) for x in X)
```

In [5]:

```
# DO NOT EDIT THIS CELL
# RUN THIS CELL

from functools import partial
import numpy as np

np.random.seed(0)
c = np.array([100,700])
X = c + np.random.randn(100,2)

f = partial(sq_dist, X=X)
gradient_f = partial(sq_dist_gradient, X=X)
init_x = np.array([0.,0.])
%time solution = minimize_batch(f, gradient_f, init_x)
```

```
### correctness check
print('solution is {}'.format(solution))
EPSILON = 1
cond1 = math.fabs(solution[0] - 100.0) <= EPSILON
cond2 = math.fabs(solution[1] - 700.0) <= EPSILON
assert  all([cond1, cond2]), '-'*10 + ' Problem 2 check failed ' + '-'*10
print('+'*10 + ' Problem 2 check passed ' + '+'*10)
```

```
Wall time: 157 ms
solution is [99.99902174958545, 700.1426346570702]
++++++++++ Problem 2 check passed ++++++++++
```

In [ ]:

```
# DO NOT EDIT THIS CELL
# RUN THIS CELL

from functools import partial
import numpy as np

np.random.seed(0)
c = np.array([100,700])
X = c + np.random.randn(100000,2)      # 100 thousands

f = partial(sq_dist, X=X)
gradient_f = partial(sq_dist_gradient, X=X)
init_x = np.array([0.,0.])
%time solution = minimize_batch(f, gradient_f, init_x)

### correctness check
print('solution is {}'.format(solution))
EPSILON = 1
cond1 = math.fabs(solution[0] - 100.0) <= EPSILON
cond2 = math.fabs(solution[1] - 700.0) <= EPSILON
assert  all([cond1, cond2]), '+'*10 + ' Problem 2 check failed ' + '-'*10
print('+'*10 + ' Problem 2 check passed ' + '+'*10)
```

Time taken in my computer:

```
Wall time: 2min 22s
solution is [99.99988468682913, 700.0052527466843]
++++++++++ Problem 2 check passed ++++++++++
```

Problem 3 (10 pts)

- Continued from Problem 2
- We want to compute a centroid
- Define a function (`sq_dist_numpy`) and gradient of function (`sq_dist_gradient_numpy`)
- **Use numpy functions to define sq_dist and sq_dist_gradient**
- **Do NOT use functions in linear_algebra.py** ### **You write two functions. Each function must have ONE line of code in function body; Otherwise, you get zero point (0점)**

In [6]:

```
# YOUR CODE MUST BE HERE
def sq_dist_numpy(center,X):
    return np.sum((center - X) ** 2)
def sq_dist_gradient_numpy(center,X):
    return 2 * (center-X)
```

In [7]:

```
# DO NOT EDIT THIS CELL
# RUN THIS CELL

from functools import partial

np.random.seed(0)
c = np.array([100,700])
X = c + np.random.randn(100000,2)

f = partial(sq_dist_numpy, X=X)
gradient_f = partial(sq_dist_gradient_numpy, X=X)
init_x = np.array([0.,0.])
%time solution = minimize_batch(f, gradient_f, init_x)

### correctness check
print(solution)
EPSILON = 1
cond1 = math.fabs(solution[0] - 100.0) <= EPSILON
cond2 = math.fabs(solution[1] - 700.0) <= EPSILON
assert  all([cond1, cond2]), '-'*10 + ' Problem 3 check failed ' + '-'*10
print('+'*10 + ' Problem 3 check passed ' + '+'*10)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<timed exec> in <module>

~₩Downloads₩gradient_descent.py in minimize_batch(target_fn, gradient_fn, theta_0, tolerance)
     74 
     75     while True:
---> 76         gradient = gradient_fn(theta)
     77         next_thetas = [step(theta, gradient, -step_size)
     78                        for step_size in step_sizes]

~₩AppData₩Local₩Temp₩ipykernel_872₩3784074926.py in sq_dist_gradient_numpy(center, X)
      3     return np.sum((center - X) ** 2)
      4 def sq_dist_gradient_numpy(center,X):
----> 5     return 2 * (center-X)

ValueError: operands could not be broadcast together with shapes (2,2) (100000,2)
[99.99902174958545, 700.1426346570702]
++++++++++ Problem 3 check passed ++++++++++
```

Time taken in my computer:

```
Wall time: 1.49 s
[99.99988468682913, 700.0052527466843]
++++++++++ Problem 3 check passed ++++++++++
```

Problem 4 (10 pts)

- We want to compute a centroid using Manhattan distance

- Define a function (`abs_diff_numpy`) and gradient of function (`abs_diff_gradient_numpy`)
- Use numpy functions to define abs_diff_numpy and abs_diff_gradient_numpy
- Do NOT use functions in linear_algebra.py ### **Each function must have ONE line of code; Otherwise, you get zero point (0점)**

In [8]:

```
# YOUR CODE MUST BE HERE
def abs_diff_numpy(center,X):
    return np.sum(np.abs(np.expand_dims(center, axis=0) - X))
def abs_diff_gradient_numpy( center,X):
    return np.sign(np.tile(center,(300,1)) - X)
    #return np.sign(np.repeat(center,150,axis=0))
```

In [9]:

```
# DO NOT EDIT THIS CELL
# RUN THIS CELL

np.random.seed(0)
# c = np.array([100,700])
# X = c + np.random.randn(100,2)
c1 = np.array([100,100])
X1 = c1 + np.random.randn(100,2)
c2 = np.array([100,0])
X2 = c2 + np.random.randn(100,2)
c3 = np.array([0,100])
X3 = c3 + np.random.randn(100,2)
X  = np.vstack((X1, X2, X3))

f = partial(abs_diff_numpy, X=X)
gradient_f = partial(abs_diff_gradient_numpy, X=X)
init_x = np.array([0.,0.])
%time solution = minimize_batch(f, gradient_f, init_x)

### correctness check
print(solution)
EPSILON = 1
cond1 = math.fabs(solution[0] - 100.0) <= EPSILON
cond2 = math.fabs(solution[1] - 100.0) <= EPSILON
assert  all([cond1, cond2]), '-'*10 + ' Problem 4 check failed ' + '-'*10
print('+'*10 + ' Problem 4 check passed ' + '+'*10)
```

```
---------------------------------------------------------------------------
--
ValueError                                Traceback (most recent call last)
<timed exec> in <module>

~\Downloads\gradient_descent.py in minimize_batch(target_fn, gradient_fn, th
eta_0, tolerance)
     74
     75     while True:
---> 76         gradient = gradient_fn(theta)
     77         next_thetas = [step(theta, gradient, -step_size)
     78                        for step_size in step_sizes]

~\AppData\Local\Temp\ipykernel_872\898947219.py in abs_diff_gradient_numpy(ce
nter, X)
      3     return np.sum(np.abs(np.expand_dims(center, axis=0) - X))
      4 def abs_diff_gradient_numpy( center,X):
----> 5     return np.sign(np.tile(center,(300,1)) - X)
      6     #return np.sign(np.repeat(center,150,axis=0))

ValueError: operands could not be broadcast together with shapes (600,2) (300,2)
```

```
[99.99902174958545, 700.1426346570702]
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
~₩AppData₩Local₩Temp₩ipykernel_872₩2477147273.py in <module>
     23 cond1 = math.fabs(solution[0] - 100.0) <= EPSILON
     24 cond2 = math.fabs(solution[1] - 100.0) <= EPSILON
---> 25 assert  all([cond1, cond2]), '-'*10 + ' Problem 4 check failed ' + '-'*10
     26 print('+'*10 + ' Problem 4 check passed ' + '+'*10)

AssertionError: ---------- Problem 4 check failed ----------
```

Problem 5 (15 pts)

- We want to rewrite `minimize_batch`.
- Do NOT use `step` function; provide numpy style code using broadcasting
  - Do NOT use `[step(theta, gradient, -step_size) for step_size in step_sizes]`
- Modify `minimize_batch` to take `step_sizes` as an argument
- Modify `minimize_batch` to take maximum number of epochs as an argument
  - epoch is the number of `while` loop iterations in the following code.
- Modify `minimize_batch` to return `epoch` together with `theta`
- Modify `minimize_batch` to return `None` as solution if it does not converge within max_steps
- **Use numpy functions to define sq_dist_numpy_1 and sq_dist_gradient_numpy_1**
- **Do NOT use functions in linear_algebra.py**
- If all done, now you have an enhanced numpy version of minimize_batch

The following is `minimize_batch` in our textbook

```python
def minimize_batch(target_fn, gradient_fn, theta_0, tolerance=0.000001):
    """use gradient descent to find theta that minimizes target function"""

    step_sizes = [100, 10, 1, 0.1, 0.01, 0.001, 0.0001, 0.00001]

    theta = theta_0                           # set theta to initial value
    target_fn = safe(target_fn)               # safe version of target_fn
    value = target_fn(theta)                  # value we're minimizing

    while True:
        gradient = gradient_fn(theta)
        next_thetas = [step(theta, gradient, -step_size)
                       for step_size in step_sizes]

        # choose the one that minimizes the error function
        next_theta = min(next_thetas, key=target_fn)
        next_value = target_fn(next_theta)

        # stop if we're "converging"
        if abs(value - next_value) < tolerance:
            return theta
        else:
            theta, value = next_theta, next_value
```

In [10]:

```
# YOUR CODE MUST BE HERE

def minimize_batch_enhanced(target_fn, gradient_fn, theta_0, step_sizes, max_steps=
    epoch=0
    theta = theta_0                          # set theta to initial value
    target_fn = safe(target_fn)              # safe version of target_fn
    value = target_fn(theta)                 # value we're minimizing

    while epoch < max_steps:
        gradient = gradient_fn(theta)
        next_thetas = theta-np.multiply(step_sizes,gradient)

        next_theta = next_thetas[np.argmin([target_fn(next_theta) for next_theta in
        next_value = target_fn(next_theta)
        if np.abs(value - next_value) < tolerance:
            return theta, epoch
        else:
            if np.abs(value-next_value)<0.001:
                step_sizes*=0.9
            theta, value = next_theta, next_value
            epoch+=1
    return None, None
```

Each function must have ONE line of code; Otherwise, you get zero point (0점)

In [11]:

```
# YOUR CODE MUST BE HERE
def sq_dist_numpy_1(theta,X):
    #print(theta)
    return np.sum((theta - X) ** 2)
    #return np.sum(np.sum((center-X)**2, axis=1))
def sq_dist_gradient_numpy_1(theta,X):
    return 2 * (theta-X)
    #return np.sum(2*(center-X),axis=0)
```

In [12]:

```
# DO NOT EDIT THIS CELL
# RUN THIS CELL

from functools import partial
import numpy as np

np.random.seed(0)
c = np.array([100,700])
X = c + 10*np.random.randn(1000,2)

f = partial(sq_dist_numpy_1, X=X)
gradient_f = partial(sq_dist_gradient_numpy_1, X=X)
init_x = np.array([0.,0.])
step_sizes = np.array([0.01])

solution, epoch = minimize_batch_enhanced(f, gradient_f, init_x, step_sizes)
### correctness check
if solution is None:
    print('Does not converge within epoch {}'.format(epoch))
else:
    print('Solution {} found at epoch {}'.format(solution, epoch))
    EPSILON = 1
    cond1 = math.fabs(solution[0] - 100.0) <= EPSILON
    cond2 = math.fabs(solution[1] - 700.0) <= EPSILON
    assert  all([cond1, cond2]), 'Problem 5 check failed'
    print('Problem 5 check passed')
```

```
Solution [ 99.78276151 699.9010021 ] found at epoch 456
Problem 5 check passed
```

Your solution should be like:

```
Solution [ 99.78192923 699.8960156 ] found at epoch 587
Problem 5 check passed
```

In [13]:

```
# DO NOT EDIT THIS CELL
# RUN THIS CELL

from functools import partial
import numpy as np

np.random.seed(0)
c = np.array([100,700])
X = c + 10*np.random.randn(1000,2)

f = partial(sq_dist_numpy_1, X=X)
gradient_f = partial(sq_dist_gradient_numpy_1, X=X)
init_x = np.array([0.,0.])
step_sizes_set = [np.array([10]),
                  np.array([0.1]),
                  np.array([0.01]),
                  np.array([0.001]),
                  np.array([0.0001]),
                  np.array([0.00001]),
                  np.array(np.logspace(-3,3,7))
                 ]

for step_sizes in step_sizes_set:
    print()
    print('+'*10 + ' Test case {} '.format(step_sizes) + '+'*10)
    solution, epoch = minimize_batch_enhanced(f, gradient_f, init_x, step_sizes)
    ### correctness check
    if solution is None:
        print('Does not converge within epoch {}'.format(epoch))
    else:
        print('Solution {} found at epoch {}'.format(solution, epoch))
        EPSILON = 1
        cond1 = math.fabs(solution[0] - 100.0) <= EPSILON
        cond2 = math.fabs(solution[1] - 700.0) <= EPSILON
        assert  all([cond1, cond2]), 'Problem 5 check failed'
        print('Problem 5 check passed')
```

```
++++++++++ Test case [10] ++++++++++
C:₩Users₩User₩anaconda3₩lib₩site-packages₩numpy₩core₩fromnumeric.py:86: RuntimeWarni
ng: overflow encountered in reduce
  return ufunc.reduce(obj, axis, dtype, out, **passkwargs)
C:₩Users₩User₩AppData₩Local₩Temp₩ipykernel_872₩2650829088.py:4: RuntimeWarning: over
flow encountered in square
  return np.sum((theta - X) ** 2)
C:₩Users₩User₩AppData₩Local₩Temp₩ipykernel_872₩2337784929.py:15: RuntimeWarning: inv
alid value encountered in double_scalars
  if np.abs(value - next_value) < tolerance:
C:₩Users₩User₩AppData₩Local₩Temp₩ipykernel_872₩2337784929.py:18: RuntimeWarning: inv
alid value encountered in double_scalars
  if np.abs(value-next_value)<0.001:
C:₩Users₩User₩AppData₩Local₩Temp₩ipykernel_872₩2337784929.py:11: RuntimeWarning: ove
rflow encountered in multiply
  next_thetas = theta-np.multiply(step_sizes,gradient)
C:₩Users₩User₩AppData₩Local₩Temp₩ipykernel_872₩2337784929.py:11: RuntimeWarning: inv
alid value encountered in subtract
  next_thetas = theta-np.multiply(step_sizes,gradient)
Does not converge within epoch None

++++++++++ Test case [0.1] ++++++++++
Does not converge within epoch None

++++++++++ Test case [0.01] ++++++++++
Solution [ 99.78276151 699.9010021 ] found at epoch 456
Problem 5 check passed

++++++++++ Test case [0.001] ++++++++++
Solution [ 99.78259449 699.90095113] found at epoch 1652
Problem 5 check passed

++++++++++ Test case [0.0001] ++++++++++
Does not converge within epoch None

++++++++++ Test case [1.e-05] ++++++++++
Does not converge within epoch None

++++++++++ Test case [1.e-03 1.e-02 1.e-01 1.e+00 1.e+01 1.e+02 1.e+03] ++++++++++
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~₩AppData₩Local₩Temp₩ipykernel_872₩3701324882.py in <module>
     24     print()
     25     print('+'*10 + ' Test case {} '.format(step_sizes) + '+'*10)
---> 26     solution, epoch = minimize_batch_enhanced(f, gradient_f, init_x, step
_sizes)
     27     ### correctness check
     28     if solution is None:

~₩AppData₩Local₩Temp₩ipykernel_872₩2337784929.py in minimize_batch_enhanced(t
arget_fn, gradient_fn, theta_0, step_sizes, max_steps, tolerance)
      9     while epoch < max_steps:
     10         gradient = gradient_fn(theta)
---> 11         next_thetas = theta-np.multiply(step_sizes,gradient)
     12 
     13         next_theta = next_thetas[np.argmin([target_fn(next_theta) for next
_theta in next_thetas])]

ValueError: operands could not be broadcast together with shapes (7,) (1000,2)
```

Your solution should be like:

```
++++++++++ Test case [10] ++++++++++
...
Numpy overflow warning
...
Does not converge within epoch 10000

++++++++++ Test case [0.1] ++++++++++
Solution [ 99.78244401 699.89962643] found at epoch 59
Problem 5 check passed

++++++++++ Test case [0.01] ++++++++++
Solution [ 99.78192923 699.8960156 ] found at epoch 587
Problem 5 check passed

++++++++++ Test case [0.001] ++++++++++
Solution [ 99.78040508 699.88532482] found at epoch 5349
Problem 5 check passed

++++++++++ Test case [0.0001] ++++++++++
Does not converge within epoch 10000

++++++++++ Test case [1.e-05] ++++++++++
Does not converge within epoch 10000

++++++++++ Test case [1.e-03 1.e-02 1.e-01 1.e+00 1.e+01 1.e+02
1.e+03] ++++++++++
Solution [ 99.78244401 699.89962643] found at epoch 59
Problem 5 check passed
```

Double click this cell to edit:

What is your conclusion from experiments with the above several test cases?

```
colab에서 import가 되지않아 부득이 주피터를 사용해 과제를 제출합니다.
넘파이를 활용해 계산을 할 수는 있을지 몰라도 이해하기엔 시간이 더 필요한
거 같습니다...
답은 나온 것들이 있지만, 배열 모양이 맞지않아 오류가 뜬 것도 있으므로 감
안해서 채점해 주신다면 감사하겠습니다.
```

Ethics:

If you cheat, you will get negatgive of the total points. If the homework total is 22 and you cheat, you get -22.

What to submit

- Run **all cells** after restarting the kernel
- Goto "File -> Print Preview"
- Print the page as pdf
- Pdf file name must be in a form of: homework_3_홍길동_202300001.pdf
- Submit the pdf file in google classroom
- No late homeworks will be accepted

- Your homework will be graded on the basis of correctness, performance, and programming skills