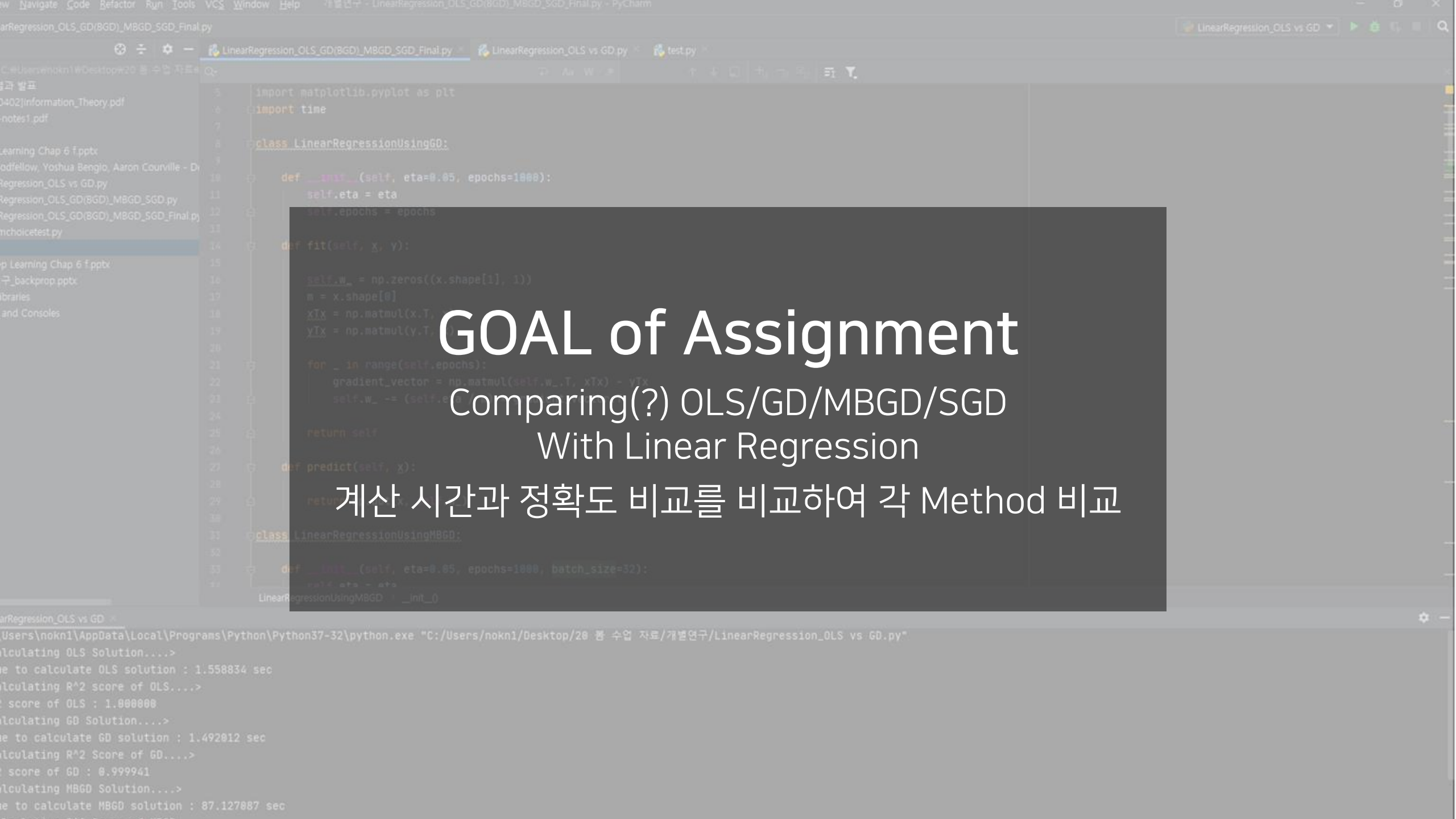


개별연구 숙제 수행 결과 보고

20170604 정지형

C:\Users\nokn1\AppData\Local\Programs\Python\Python37-32\python.exe "C:/Users/nokn1/Desktop/28. 품 수업 자료/개별연구/LinearRegression_OLS vs GD.py"

Calculating OLS Solution....>
Time to calculate OLS solution : 1.558834 sec
Calculating R^2 score of OLS....>
R^2 score of OLS : 1.000000
Calculating GD Solution....>
Time to calculate GD solution : 1.492012 sec
Calculating R^2 Score of GD....>
R^2 score of GD : 0.999941
Calculating MBGD Solution....>
Time to calculate MBGD solution : 87.127087 sec



GOAL of Assignment

Comparing(?) OLS/GD/MBGD/SGD
With Linear Regression

계산 시간과 정확도 비교를 비교하여 각 Method 비교

Codes

- Imports & settings

```
2 # imports
3 import numpy as np
4 from numpy.linalg import inv
5 import matplotlib.pyplot as plt
6 import time
```

- Numpy 이용
- Time - 계산 시간 측정

```
91 # Generate random data-set
92 np.random.seed(0)
93 print("<Generating Data....>")
94 n = 100 # vector size
95 N = (2*5)*5*100 # number of data
96
97 x = np.random.rand(N, n) # Data matrix
98 w = 10 * np.random.rand(n, 1) # True parameter
99 yt = np.matmul(x, w) # y value matrix induced by data x
100 e = np.random.randn(N, 1) # error term
101 yd = np.add(yt, e) # y data matrix we have
102
103 a = 10 # number of calculations
```

Codes

- Calculating solutions (Ex. OLS)

```
def __init__(self, eta=0.05, epochs=1000):
    # Calculate the Ordinary Least Squares solution
    m_time_OLS = np.zeros(a)
    m_R2score_OLS = np.zeros(a)

    print("<Executing OLS method....>")
    for i in range(a):
        calstarted_OLS = time.time()

        b1 = inv(np.matmul(x.T, x))
        b2 = np.matmul(x.T, yd)
        B = np.matmul(b1, b2)          # solution matrix
        y_OLS = np.matmul(x, B)        # y value induced by x, B

        calended_OLS = time.time()
        caltime_OLS = calended_OLS - calstarted_OLS    # OLS calculation time

        m_time_OLS[i] = caltime_OLS      # Recording caltime to get average

        # Calculating R^2 score for OLS
        ssr = np.sum((yt - y_OLS)**2)
        sst = np.sum((yt - np.mean(yt))**2)
        r2score_OLS = 1 - (ssr/sst)

        m_R2score_OLS[i] = r2score_OLS    # Recording R^2 score to get average

    # Calculating average run-time of OLS
    avg_time_OLS = np.average(m_time_OLS)
    avg_R2score_OLS = np.average(m_R2score_OLS)
```

- 여러 번 돌려 평균 값으로 비교
- 계산 시간 측정 > 어느 method가 더 빠를까?
- R^2 Score을 통한 Evaluation

Spoiler?

- Shocking calculation results

```
C:\Users\nokn1\AppData\Local\Programs\Python\Python37-32\p
<Generating Data....>
<Executing OLS method....>
<Executing Gradient Descent....>
<Executing Mini-Batch Gradient Descent....>
<Executing Stochastic Gradient Descent....>
<Report> With n = 100, N = 16,000, a = 10
| Method || Average Run-time || Average R^2 Score |
| OLS    || 0.017554 sec          || 0.999978      |
| GD     || 0.071709 sec          || 0.999915      |
| MBGD   || 0.701330 sec          || 0.999685      |
| SGD    || 0.652762 sec          || 0.919777      |
```

4.2 times
9.9 times
Similar

Spoiler?

- Shocking calculation results

```
C:\Users\nokn1\AppData\Local\Programs\Python\Python37-32\python.exe "C:/Users/nokn1/Desktop/LinearRegression_OLS vs GD.py"
<Generating Data....>
<Executing OLS method....>
<Executing Gradient Descent....>
<Report> With n = 100, N = 1,440,000, a = 100
| Method || Average Run-time || Average R^2 Score |
| OLS    || 2.066714 sec          || 1.000000      |
| GD     || 1.696927 sec          || 0.999946      |

Process finished with exit code 0
```

* Another execution of methods gave a result that avg runtime GD > avg runtime OLS, so we need to check this in environment that our data has much higher dimension

Spoiler?

- Shocking calculation results

Why?

Gradient 계산이 복잡

$$\frac{\partial J}{\partial \mathbf{W}} = (\mathbf{W}^T \mathbf{X}^T \mathbf{X} - \mathbf{y}^T \mathbf{X})$$

TMI - Too Much Iterations?
Random Data Generation

C:\Users\nokn1\AppData\Local\Programs\Python\Python37-32\p				
<Generating Data....>				
<Executing OLS method....>				
<Executing Gradient Descent....>				
<Executing Mini-Batch Gradient Descent....>				
<Executing Stochastic Gradient Descent....>				
<Report>				
Method		Average	R^2 Score	
OLS		0.017554 sec	0.999978	
GD		0.071709 sec	0.999915	
MBGD		0.701955 sec	0.797885	
SGD		0.652762 sec	0.919777	

Codes

- OLS again

```
105 # Calculate the Ordinary Least Squares solution
106 m_time_OLS = np.zeros(a)
107 m_R2score_OLS = np.zeros(a)
108
109 print("<Executing OLS method....>")
110 for i in range(a):
111
112     calstarted_OLS = time.time()
113
114     b1 = inv(np.matmul(x.T, x))
115     b2 = np.matmul(x.T, yd)
116     B = np.matmul(b1, b2) # solution matrix
117     y_OLS = np.matmul(x, B) # y value induced by x, B
118
119     calended_OLS = time.time()
120     caltime_OLS = calended_OLS - calstarted_OLS # OLS calculation time
121
122     m_time_OLS[i] = caltime_OLS # Recording caltime to get average
123
124 # Calculating R^2 score for OLS
125 ssr = np.sum((yt - y_OLS)**2)
126 sst = np.sum((yt - np.mean(yt))**2)
127 r2score_OLS = 1 - (ssr/sst)
128
129 m_R2score_OLS[i] = r2score_OLS # Recording R^2 score to get average
130
131 # Calculating average run-time of OLS
132 avg_time_OLS = np.average(m_time_OLS)
133 avg_R2score_OLS = np.average(m_R2score_OLS)
```

- Inverse Calculation in numpy
Pseudo inverse를 계산
- 당연히 기존 inverse 계산보다
빠름
- So, 계산이 matrix 몇 번 곱하기
에 그침

Codes

- But Gradient Descent (BGD?)

```
LinearRegression_OLS_GD(BGD)_MBGD_SGD_Final.py
LinearRegression_OLS vs GD.py
test.py

import matplotlib.pyplot as plt
import time

def __init__(self, eta=0.05, epochs=1000):
    self.eta = eta

class LinearRegressionUsingGD:
    def __init__(self, eta=0.05, epochs=1000):
        self.eta = eta
        self.epochs = epochs

    def fit(self, x, y):
        self.w_ = np.zeros((x.shape[1], 1))
        m = x.shape[0]
        xTx = np.matmul(x.T, x)
        yTx = np.matmul(y.T, x)

        for _ in range(self.epochs):
            gradient_vector = np.matmul(self.w_.T, xTx) - yTx
            self.w_ -= (self.eta / m) * gradient_vector.T

        return self

    def predict(self, x):
        return np.dot(x, self.w_)

class LinearRegressionUsingMBGD:
```

- Learning Rate $\eta = 0.05$
 - 1,000 Epochs(iterations)
 - Initial $W_0 = 0$
-
- 주어진 Gradient 계산 식으로 Gradient 계산
 - OLS 보다 훨씬 많은 matrix 곱셈 수행
 - If Data size is Large, OLS/GD is Comparable.

Codes

- Mini-batch Gradient Descent

```
31 class LinearRegressionUsingMBGD:
32
33     def __init__(self, eta=0.05, epochs=1000, batch_size=32):
34         self.eta = eta
35         self.epochs = epochs
36         self.batch_size = batch_size
37
38     def fit(self, x, y):
39         self.w_ = np.zeros((x.shape[1], 1))
40         m = self.batch_size
41
42         for _ in range(self.epochs):
43
44             # Choosing Mini-Batch with size m
45             numdata = np.arange(x.shape[0])
46             J = np.random.choice(numdata, size=m, replace=False)
47             xb = x[J]
48             yb = y[J]
49             xbTxb = np.matmul(xb.T, xb)
50             ybTxb = np.matmul(yb.T, xb)
51             # Calculating & Updating Gradient
52             gradient_vector = np.matmul(self.w_.T, xbTxb) - ybTxb
53             self.w_ -= (self.eta / m) * gradient_vector.T
54
55         return self
56
57     def predict(self, x):
58
59         return np.dot(x, self.w_)
```

- Learning Rate $\eta = 0.05$
 - 1,000 Epochs(iterations)
 - Batch size = 32
 - Initial $W_0 = 0$
-
- 1개의 Mini Batch Random 추출
 - 주어진 Gradient 계산 식으로 Gradient 계산(하기 위해...)
 - OLS/GD 보다 훨씬 더더 많은 matrix 곱셈 수행

Codes

- Stochastic Gradient Descent

```
61 class LinearRegressionUsingSGD:
62
63     def __init__(self, eta=0.05, epochs=1000, batch_size=1):
64         self.eta = eta
65         self.epochs = epochs
66         self.batch_size = batch_size
67
68     def fit(self, X, y):
69         self.w_ = np.zeros((X.shape[1], 1))
70         m = self.batch_size
71
72         for _ in range(self.epochs):
73
74             # Choosing a data randomly
75             numdata = np.arange(X.shape[0])
76             J = np.random.choice(numdata, size=m, replace=False)
77             xb = X[J]
78             yb = y[J]
79             xBTxb = np.matmul(xb.T, xb)
80             yBTyb = np.matmul(yb.T, yb)
81             # Calculating & Updating Gradient
82             gradient_vector = np.matmul(self.w_.T, xBTxb) - yBTyb
83             self.w_ -= (self.eta / m) * gradient_vector.T
84
85         return self
86
87     def predict(self, X):
88
89         return np.dot(X, self.w_)
```

- Learning Rate $\eta = 0.05$
- 1,000 Epochs(iterations)
- Batch size = 1
- Initial $W_0 = 0$

-
- 1개의 Data Random 추출
 - 주어진 Gradient 계산식으로 Gradient 계산(하기 위해...)
 - OLS/GD 보다 훨씬 더더 많은 matrix 곱셈 수행, MBGD와는 비슷.

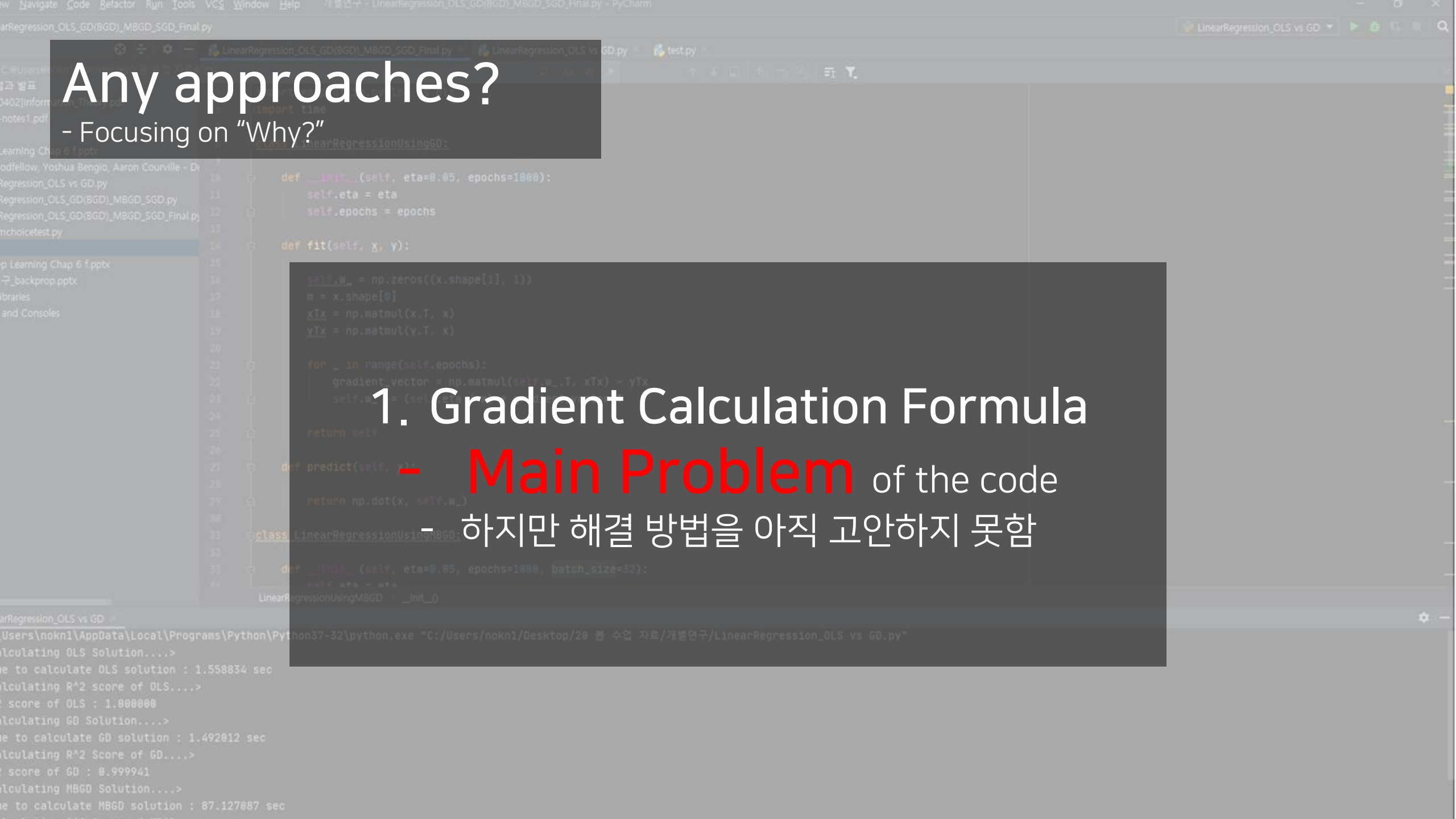
Any approaches?

- Focusing on "Why?"

1. Gradient Calculation Formula

- **Main Problem** of the code

- 하지만 해결 방법을 아직 고안하지 못함



Any approaches?

- Focusing on "Why?"

2. TMI – Too Much Iterations

1,000 epochs(iterations) > Isn't it too large?

Stopping Criteria – gradient가 적당히 작아졌을 때
멈추도록 하면 어떨까?

```
C:\Users\nokn1\AppData\Local\Programs\Python\Python37-32\python.exe "C:/Users/nokn1/Desktop/28 품 수업 자료/개별연구/LinearRegression_OLS vs GD.py"
```

```
Calculating OLS Solution....>
Time to calculate OLS solution : 1.558834 sec
Calculating R^2 score of OLS....>
R^2 score of OLS : 1.000000
Calculating GD Solution....>
Time to calculate GD solution : 1.492012 sec
Calculating R^2 Score of GD....>
R^2 score of GD : 0.999941
Calculating MBGD Solution....>
Time to calculate MBGD solution : 87.127087 sec
```

Stopping Criteria

- Handling "TMI" Problem on GD case

```
def __init__(self, eta=0.05, epochs=1000):
    self.eta = eta
    self.epochs = epochs

def fit(self, X, y):
    self.W_ = np.zeros((X.shape[1], 1))
    m = X.shape[0]
    XTX = np.matmul(X.T, X)
    YTX = np.matmul(y.T, X)

    for _ in range(self.epochs):
        gradient_vector = np.matmul(self.W_.T, XTX) - YTX
        self.W_ -= (self.eta / m) * gradient_vector

    return self

def predict(self, X):
    return np.dot(X, self.W_)

class LinearRegressionUsingMBGD:
    def __init__(self, eta=0.05, epochs=1000):
        self.eta = eta
        self.epochs = epochs
```

```
# Reference : https://towardsdatascience.com/linear-regression-using-python-t
# imports
import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plt
import time

class LinearRegressionUsingGD:
    def __init__(self, eta=0.05, epochs=1000):
        self.eta = eta
        self.epochs = epochs

    def fit(self, X, y):
        self.W_ = np.zeros((X.shape[1], 1))
        m = X.shape[0]
        XTX = np.matmul(X.T, X)
        YTX = np.matmul(y.T, X)

        for _ in range(self.epochs):
            gradient_vector = np.matmul(self.W_.T, XTX) - YTX
            self.W_ -= (self.eta / m) * gradient_vector.T
            # Adding Stopping Criteria - with hyper-parameter 10^-3
            if np.max(np.abs(gradient_vector)) < 10**(-3):
                break

        return self

    def predict(self, X):
        return np.dot(X, self.W_)
```

Run: LinearRegression_OLS_GD(BGD)_MBGD_SGD_Final_...

Method	Average Run-time	Average R^2 Score
OLS	1.314596 sec	1.000000
GD	1.357392 sec	0.999946

Process finished with exit code 0

Relative error = 0.032554

Stopping Criteria

- Handling "TMI" Problem on GD case

```
def __init__(self, eta=0.05, epochs=1000):
```

```
    self.eta = eta
```

```
    self.epochs = epochs
```

```
def fit(self, X, y):
```

```
    self.W_ = np.zeros((X.shape[1], 1))
```

```
    m = X.shape[0]
```

```
    XTX = np.matmul(X.T, X)
```

```
    YTX = np.matmul(y.T, X)
```

```
    for _ in range(self.epochs):
```

```
        gradient_vector = np.matmul(self.W_.T, XTX) - YTX
```

```
        self.W_ -= (self.eta / m) * gradient_vector
```

```
    return self
```

```
def predict(self, X):
```

```
    return np.dot(X, self.W_)
```

```
class LinearRegressionUsingMBGD:
```

```
    def __init__(self, eta=0.05, epochs=1000):
```

```
        self.eta = eta
```

```
        self.epochs = epochs
```

```
    def fit(self, X, y):
```

```
        self.W_ = np.zeros((X.shape[1], 1))
```

```
        m = X.shape[0]
```

```
        XTX = np.matmul(X.T, X)
```

```
        YTX = np.matmul(y.T, X)
```

```
1  # Reference : https://towardsdatascience.com/linear-regression-using-python-b13
2  # imports
3  import numpy as np
4  from numpy.linalg import inv
5  import matplotlib.pyplot as plt
6  import time
7
8  class LinearRegressionUsingGD:
9
10     def __init__(self, eta=0.05, epochs=1000):
11         self.eta = eta
12         self.epochs = epochs
13
14     def fit(self, X, y):
15
16         self.W_ = np.zeros((X.shape[1], 1))
17         m = X.shape[0]
18         XTX = np.matmul(X.T, X)
19         YTX = np.matmul(y.T, X)
20
21         for _ in range(self.epochs):
22             gradient_vector = np.matmul(self.W_.T, XTX) - YTX
23             self.W_ -= (self.eta / m) * gradient_vector.T
24             # Adding Stopping Criteria - with hyper-parameter 10^-2
25             if np.max(np.abs(gradient_vector)) < 10**(-2):
26                 break
27
28         return self
29
30     def predict(self, X):
31         return np.dot(X, self.W_)
```

Run: LinearRegression_OLS_GD(MBGD)_SGD_Final_...
C:\Users\nokn1\AppData\Local\Programs\Python\Python37-32\python.exe "C:\Users\nokn1\Desktop\20 봄 수업 자료\개발연구\LinearRegression_OLS_GD(MBGD)_SGD_Final.py"

```
<Generating Data....>
<Executing OLS method....>
<Executing Gradient Descent....>
<Report / n=100, N=1440000, a=100>
| Method | Average Run-time | Average R^2 Score |
| OLS    | 1.259443 sec     | 1.000000          |
| GD     | 1.362797 sec     | 0.999946          |
Process finished with exit code 0
```

Relative error = 0.082063

Stopping Criteria

- Handling "TMI" Problem on GD case

```
def __init__(self, eta=0.05, epochs=1000):
    self.eta = eta
    self.epochs = epochs

def fit(self, X, y):
    self.w_ = np.zeros((X.shape[0], 1))
    m = X.shape[0]
    XTX = np.matmul(X.T, X)
    YTX = np.matmul(y.T, X)

    for _ in range(self.epochs):
        gradient_vector = np.matmul(self.w_.T, XTX) - YTX
        self.w_ -= (self.eta / m) * gradient_vector

    return self

def predict(self, X):
    return np.dot(X, self.w_)
```

```
class LinearRegressionUsingMBGD:
    def __init__(self, eta=0.05, epochs=1000):
        self.eta = eta
        self.epochs = epochs
```

Project Structure:

- 개별연구
- 과제 결과 발표
- [20200402]Information_Theory.pdf
- cs229-notes1.pdf
- d.py
- Deep Learning Chap 6 f.pptx
- Ian Goodfellow, Yoshua Bengio, Aaron Courville - D
- LinearRegression_OLS vs GD.py
- LinearRegression_OLS_GD(BGD)_MBGD_SGD.py
- LinearRegression_OLS_GD(BGD)_MBGD_SGD_Final.py
- LinearRegression_OLS_GD(BGD)_MBGD_SGD_Final_w
- randomchoicetest.py
- test.py
- ~\$Deep Learning Chap 6 f.pptx
- 개별연구_backprop.pptx
- External Libraries
- Scratches and Consoles

```
# Reference : https://towardsdatascience.com/linear-regression-using-pytl
# imports
import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plt
import time

class LinearRegressionUsingGD:
    def __init__(self, eta=0.05, epochs=1000):
        self.eta = eta
        self.epochs = epochs

    def fit(self, X, y):
        self.w_ = np.zeros((X.shape[1], 1))
        m = X.shape[0]
        XTX = np.matmul(X.T, X)
        YTX = np.matmul(y.T, X)

        for _ in range(self.epochs):
            gradient_vector = np.matmul(self.w_.T, XTX) - YTX
            self.w_ -= (self.eta / m) * gradient_vector.T
            # Adding Stopping Criteria - with hyper-parameter 10^-2
            if np.max(np.abs(gradient_vector)) < 10**(-1):
                break

        return self

    def predict(self, X):
        return np.dot(X, self.w_)
```

Run: LinearRegression_OLS_GD(BGD)_MBGD_SGD_Final.py

C:\Users\nokn1\AppData\Local\Programs\Python\Python37-32\python.exe "C:/Users/nokn1/Desktop/20 품 수업 자료/개별연구/L

```
<Generating Data....>
<Executing OLS method....>
<Executing Gradient Descent....>
<Report / n=100, N=1440000, a=100>
| Method || Average Run-time || Average R^2 Score |
| OLS    || 1.336497 sec          || 1.000000    |
| GD     || 1.333568 sec          || 0.999946    |
```

Process finished with exit code 0

Relative error = 0.002191

"Why?" again

- Kind of Error Analysis?

1. Still we have "Formula Problem"

- Maybe we can calculate gradient more conveniently.
- And we can improve our algorithm; our algorithm of MBGD, SGD has serious problem.(Because of extracting a batch)

2. "TMI" Handling

- We have to check that Stopping Criteria is valid; maybe we need more "time".

3. "Random Number Problem"

- Randomly generated 'float' data causes MemoryError and kind of noises – for every "run", data differs, so that it might effect noises on our results.

4. "Laptop Problem" <=> "Second Problem"

- Poor performance of my laptop regulates the size of the data, so we can't check our algorithms/calculations in Large data environment.

As a result of Laptop problem, each of calculations are done in just 2 seconds.

This is also a big problem because of noises from randomly generated data

Reference

- Linear regression using matrix derivatives - Vivek Yadav's blog

URL : <http://vxy10.github.io/2016/06/25/lin-reg-matrix/>

- Linear Regression using Python - Animesh Agarwal

URL : <https://towardsdatascience.com/linear-regression-using-python-b136c91bf0a2>

이외 다수...

Special thanks to 김 아무개 군

Thank you.