Python-based Machine Learning Algorithmic Thinking

KICS KoreaAI Tutorial
12/17/2020
Prof. Sang-Chul Kim

https://knu.webex.com/meet/see1
Tutorial Schedule

• Install Pycharm, interpreter Anaconda, and TensorFlow for machine learning
• Data Visualization
• Curve-fitting Strategy
• Debugging for Numerical Algorithm

• Understanding Linear Regression
  – With polyfilt() method
• Understanding Tensorflow
• Polynomial Regression
• Multivariable Regression
  – With scikit-learn package
Install PyCharm, interpreter Anaconda, and TensorFlow for machine learning

For installing PyCharm
https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/InstallPycharm
For installing Pycharm

• 1. Install Pycharm: Installing Pycharm IDE on Ubuntu
   – Pycharm installation video: https://youtu.be/ClQppYllGKE

• 2. Install anaconda: Installing anaconda interpreter on Ubuntu
   – Anaconda is an all-in-one package interpreter
   – Anaconda installation video: https://youtu.be/OeWBmxKzYOQ

• 3. Integration of Pycharm & anaconda: Integrating Pycharm IDE and Anaconda Interpreter in Ubuntu

• 4. Install TensorFlow: Installing Google TensorFlow platform for data analysis and machine learning on Ubuntu
   – TensorFlow installation video: https://youtu.be/5ahUc6zB3NI

• 5. How to Debugging in Pycharm: Introduction to Pycharm debugging through data visualization example
   – Introduction to the difference between console debugging and UI debugging
   – How to Debugging in Pycharm video: https://youtu.be/4dnSdcGT_mk
Google Colab
Google Colab

https://colab.research.google.com/

Table of contents
File
New Python 3 notebook
New Python 2 notebook
Open notebook...
Upload notebook...
Rename...
Move to trash
Save a copy in Drive.

This notebook provides an introduction to computing on a...

Colaboratory
What is Colaboratory?
Colaboratory is a research tool ...

google.com 검색결과 더보기 »
Google Colab

- Google colaboratory service, a web service called colab for short,
- runs Jupiter notebooks on Google servers
- provides free of charge for users to use.
- To use this service, you must have a Google Gmail account.
- If you have a Gmail account, you can use Colab.

https://datascienceschool.net/view-notebook/f9d9fdddb7cc7494a9e4be99f0e137be0/
Google Colab

• The COLab notebook has the following advantages.
• You can work like a Jupiter notebook using only a web browser without installing Python.
• Because it is easy to share with other users, it is widely used for research and education.
• Packages commonly used for data analysis such as Tensorflow, keras, matplotlib, scikit-learn, and pandas are preinstalled.
• You can use the GPU for free.
• It can be shared and edited in the same way as Google Docs or Google Spreadsheet.
• If more than one person edits the same file at the same time, the changes are immediately visible to everyone.

https://datascienceschool.net/view-notebook/f9d9fddb7cc7494a9e4be99f0e137be0/
Colaboratory에 오신 것을 환영합니다

Colaboratory는 설치가 필요 없으며 완전히 클라우드에서 실행되는 무료 Jupyter 노트 환경입니다.
Colaboratory를 사용하면 브라우저를 통해 무료로 코드를 작성 및 실행하고, 문서를 저장 및 공유하며, 강력한 컴퓨팅 리소스를 이용할 수 있습니다.

[ ] Colaboratory 소개
3분 간의 동영상을 통해 Colaboratory의 주요 기능을 간단하게 알아보세요.

Get started with Google Colaboratory (Co...)

Intro to Google Colab

Coding TensorFlow

시작하기
지금 엽니다 게시 문서는 Colaboratory에 호스팅된 Jupyter 노트입니다. 정적인 페이지가 아닌, Python 등의 언어로 코드를 작성하고 실행할
import tensorflow as tf
Learn how to approximate true values through repetition of trial and error.
Approximation Strategy

1. The continuous system is approximated as a discrete system.
2. Replace integral with sum
3. Replace the derivative with Infinite Difference.
4. Replace nonlinearity with linearity
5. Transform a problem into another problem
6. It approximates the true value through repetition of trial and error.
   - Graphical Method, Bracketing Method (구간법), Open Method (개방법)
Data fitting strategy

There is a choice between Too Small number of incremental length (Too small number of increments) and Too many number of incremental length (Too many number of increments).
Fitting of number of incremental length

• Under fitting
  – A phenomenon in which the number of increments that can represent the data is too small to properly determine the existing solution
  – 데이터를 표현할 수 있는 증분 수가 너무 적어서, 존재하는 해를 제대로 구해내지 못하는 현상
  – Too Small number of incremental length (증분 수가 너무 적음)

• Normal fitting
  – The situation where the number of increments that can represent the data is adequate and all existing solutions are found
  – 데이터를 표현할 수 있는 증분 수가 적절 하여, 존재하는 해를 모두 구하는 상황

• Over fitting:
  – A situation where the number of increments required to obtain all existing solutions is too large than necessary, which wastes computing resources.
  – Too many number of incremental lengths (증분 수가 너무 많은 것)
Let's understand the concept of under fitting through GUI debugging.

- [GitHub Link](https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/incremental1.py)

**Debugging Points**

- What does `np.size()-1` mean?
- What is the meaning of `np.sign(f[k]) != np.sign(f[k+1])`?
- What is the role of the `append()` method?
- What is the role of the `hstack()` method?
- What does the `reshape()` method do?

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

def incsearch(func, xmin, xmax, ns):
    x = np.linspace(xmin, xmax, ns)
    f = func(x)
    nb = 0; xb = []
    for k in range(np.size(x)-1):
        if np.sign(f[k]) != np.sign(f[k+1]):
            nb = nb + 1
            xb.append(x[k])
            xb.append(x[k+1])
    xbt = np.hstack(xb)
    xb = xbt.reshape(nb, 2)
    return nb, xb

if __name__ == '__main__':
    xmin = 3; xmax = 6
    ns = 50
    func = lambda x: np.sin(np.dot(10.0, x)) + np.cos(np.dot(3.0, x))
    nb, xb = incsearch(func, 3, 6, ns)
    print('number of brackets=', nb)
    print('root interval=', xb)
```
Debugging for Incremental with ns=50

- When k=4, it can be confirmed that the value of np.sign(f[k]) != np.sign(f[k+1]) is True
  - By xb.append(x[k]), the value of x[k] = 3.2448979591836733 is stored in xb
    • xb=[3.2448979591836733]
  - By xb.append(x[k+1]), the value of x[k+1] = 3.306122448979592 is stored in xb
    • xb=[3.2448979591836733, 3.306122448979592]

- When k=5, it can be confirmed that the value of np.sign(f[k]) != np.sign(f[k+1]) is True
  - By xb.append(x[k]), the value of x[k] = 3.306122448979592 is stored in xb
    • [3.2448979591836733, 3.306122448979592, 3.306122448979592]
  - By xb.append(x[k+1]), the value of x[k+1] = 3.36734693877551 is stored in 텔
    • [3.2448979591836733, 3.306122448979592, 3.306122448979592, 3.36734693877551]
Debugging for Incremental with ns=50

- Elements of xb are stacked horizontally by hstack(xb)
  - xbt=np.hstack(xb)
  - xbt
    - array([3.24489796, 3.30612245, 3.30612245, 3.36734694])

- Converted to [nb,2] matrix by reshape(nb, 2) method
  - xb=xbt.reshape(nb, 2)
    - array([[3.24489796, 3.30612245],
               [3.30612245, 3.36734694]])

```
number of brackets=  5
root interval= [[3.24489796  3.30612245]
                [3.30612245  3.36734694]
                [3.73469388  3.79591837]
                [4.65306122  4.71428571]
                [5.63265306  5.69387755]]
```
import numpy as np  

```python
def incsearch(func, xmin, xmax, ns):
    x = np.linspace(xmin, xmax, ns)
    f = func(x)
    nb = 0
    xb = []
    for k in np.arange(np.size(x) - 1):
        if np.sign(f[k]) != np.sign(f[k + 1]):
            nb += 1
            xb.append(x[k])
            xb.append(x[k + 1])
    xbt = np.hstack(xb)
    return nb, xbt
```

```python
if __name__ == '__main__':
    xmin = 3; xmax = 6
    func = lambda x: np.sin(np.dot(10.0, x)) + np.cos(np.dot(3.0, x))
```

```
<table>
<thead>
<tr>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmin</td>
</tr>
<tr>
<td>xmax</td>
</tr>
<tr>
<td>nb</td>
</tr>
<tr>
<td>ns</td>
</tr>
<tr>
<td>f</td>
</tr>
</tbody>
</table>
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```
import numpy as np  

```
def incsearch(func, xmin, xmax, ns):
    x = np.linspace(xmin, xmax, ns)  
    f = func(x)  
    for k in np.arange(ns-1):
        if np.sign(f[k]) != np.sign(f[k+1]):
            nb = nb + 1
            xb.append(x[k])
            xb.append(x[k+1])
            xbt = np.hstack(xb)
            xb = xbt.reshape(nb, 2)

    return nb, xb
```

```
if __name__ == '__main__':
    func = lambda x: np.sin(np.dot(10, x)) + np.cos(np.dot(3, x))
```

```
Variables

- np.sign(f[k]) != np.sign(f[k+1]) - (bool) True
- Special Variables
  - f = [1.89916189 -1.69099109 -1.18862088 -0.58354376 -0.09349245 0.11658812 0.00597801 -0.38325708 -0.83614125 -1.15082957 -1.15815531 -0.79688603 -0.13793851 0.63933185]  
  - k = (int) 4  
  - nb = (int) 0  
  - ns = (int) 50  
  - xb = [class 'list']:
    - xmax = (int) 6  
    - xmin = (int) 3  
```
import numpy as np

np.sign(f[k]) != np.sign(f[k+1]): True

```python
def incsearch(func, xmin, xmax, ns):
xmin: 3  xmax: 6  ns: 50

x = np.linspace(xmin, xmax, ns)
f = func(x)
f: [-1.89916189 -1.69099109 -1.18862088 -0.58354376 -0.09349245 0.11658812

nb = 0  nb: 1
xb = []  xb: <class 'list'>: [3.2448979591836733, 3.306122448979592]
```

```python
for k in np.arange(np.size(x)-1):  k: 4

if np.sign(f[k]) != np.sign(f[k+1]):
    nb = nb + 1
    xb.append(x[k])
    xb.append(x[k+1])

xbt = np.hstack(xb)
xb = xbt.reshape(nb, 2)

return nb, xb
```

```python
if __name__ == '__main__':
xmin = 3; xmax = 6
func = lambda x: np.sin(np.dot(10.0, x)) + np.cos(np.dot(3.0, x))
```

```python
incsearch()
```
Break Point for Debugging

```python
import numpy as np

np.sign(f[k]) != np.sign(f[k+1]): False

def incsearch(func, xmin, xmax, ns):
    x = np.linspace(xmin, xmax, ns)
    x = [3, 3.06122449, 3.12244898, 3.18367347, 3.24489796, 3.30612245

f = func(x)
    f = [-1.89916189, -1.69099109, -1.18862088, -0.58354376, -0.09349245, 0.11658812, -0.00597801, -0.35710273, -1.15815531, -0.79686032

nb = 0
    nb = 2

xb = []
    xb = <class 'list'>: [3.2448979591836733, 3.306122448979592, 3.306122448979592, 3.36734693877551]

for k in range(np.size(x)-1):
    k = 6

    if np.sign(f[k]) != np.sign(f[k+1]):
        nb = nb + 1
        xb.append(x[k])
        xb.append(x[k+1])

xb = np.hstack(xb)
xb = xb.reshape(nb, 2)

return nb, xb

if __name__ == '__main__':
    x = np.linspace(10.0, x) + np.cos(np.dot(3.0, x))

incsearch()
```
import numpy as np

np.sign(f[k]) != np.sign(f[k+1]): False

def incsearch(func, xmin, xmax, ns):
    x = np.linspace(xmin, xmax, ns)
    x = 3.06122449 3.1224498 3.18367347 3.24489796 3.30612245
    x = 3.36734649
    f = func(x)
    f = [-1.89916189 -1.69099109 -1.18662088 -0.58354376 -0.09349245 0.11658812
    x = 0.22530708 0.236122448979592, 3.36734693877551, 3.734693877551, 3.734693877551
    if np.sign(f[k]) != np.sign(f[k+1]):
        nb = nb + 1
        x.append(x[k])
        x.append(x[k+1])

    x = 4.65306122 4.71428571 5.63265306 5.693877551

    return nb, x

if __name__ == '__main__':
    x = lambda x: np.sin(np.dot(10.0, x)) + np.cos(np.dot(3.0, x))
xb = xbt.reshape(nb, 2)

def incsearch(func, xmin, xmax, ns):   xmin = 3   xmax = 6   ns = 50
    x = np.linspace(xmin, xmax, ns)   x: [3. 3.06122449 3.12244898 3.18367347 3.24489796 3.30612245]
    f = func(x)   f: [-1.89916189 -1.69099109 -1.18862088 -0.58354376 -0.09349245 0.11658812]
    nb = 5
    for k in np.arange(ns-1); k: 48
        if np.sign(f[k]) != np.sign(f[k+1]):
            nb = nb + 1
            xb.append(x[k])
            xb.append(x[k+1])

    xbt = xbt.reshape(nb, 2)

    return nb, xbt

if __name__ == '__main__':
    xmin = 3; xmax = 6
    func = lambda x: np.sin(np.dot(10.0, x)) + np.cos(np.dot(3.0, x))
```python
xb = xbt.reshape(nb, 2)
```
import numpy as np
import matplotlib.pyplot as plt

x=np.linspace(3, 6, 50)
func=lambda x: np.sin(np.dot(10.0, x))+np.cos(np.dot(3.0, x))
f1=func(x)
plt.figure(1)  plt.plot(x, f1, 'ro-')  plt.grid()  plt.show()
Plotting Incremental Search with ns=50
import numpy as np

def incsearch(func, xmin, xmax, ns):
    x=np.linspace(xmin, xmax, ns)
    f=func(x)
    nb=0
    xb=[]
    for k in np.arange(np.size(x)-1):
        if np.sign(f[k]) != np.sign(f[k+1]):
            nb=nb+1
            xb.append(x[k])
            xb.append(x[k+1])
    xbt=np.hstack(xb)
    xb=xbt.reshape(nb, 2)
    return nb, xb

if __name__ == '__main__':
    xmin=3; xmax=6
    func=lambda x: np.sin(np.dot(10.0, x))+np.cos(np.dot(3.0, x))
    nb, xb=incsearch(func, 3, 6, 100)
    print('number of brackets=', nb)
    print('root interval=', xb)
Incremental Search with $ns=100$

number of brackets = 9
root interval = [[3.24242424 3.27272727]
[3.36363636 3.39393939]
[3.72727273 3.75757576]
[4.21212121 4.24242424]
[4.24242424 4.27272727]
[4.6969697  4.72727273]
[5.15151515 5.18181818]
[5.18181818 5.21212121]
[5.66666667 5.6969697 ]]
import numpy as np
import matplotlib.pyplot as plt

x=np.linspace(3, 6, 100)
func=lambda x: np.sin(np.dot(10.0, x))+np.cos(np.dot(3.0, x))
f2=func(x)
plt.figure(2) plt.plot(x,f2, 'ro-') plt.grid() plt.show()
Plotting Incremental Search with ns=100
import numpy as np

def incsearch(func, xmin, xmax, ns):
    x=np.linspace(xmin, xmax, ns)
    f=func(x)
    nb=0;     xb=[]
    for k in np.arange(np.size(x)-1):
        if np.sign(f[k]) != np.sign(f[k+1]):
            nb=nb+1
            xb.append(x[k])
            xb.append(x[k+1])
    xbt=np.hstack(xb)
    xb=xbt.reshape(nb, 2)
    return nb, xb

xmin=3; xmax=6
func=lambda x: np.sin(np.dot(10.0, x))+np.cos(np.dot(3.0, x))

nb, xb=incsearch(func, 3, 6, 100)
print('number of brackets=', nb)
print('root interval=', xb)
import numpy as np

def incsearch(func, xmin, xmax, ns):
    x=np.linspace(xmin, xmax, ns)
    f=func(x)
    nb=0;     xb=[]
    for k in np.arange(np.size(x)-1):
        if np.sign(f[k]) != np.sign(f[k+1]):
            nb=nb+1
            xb.append(x[k])
            xb.append(x[k+1])
    xbt=np.hstack(xb)
    xb=xbt.reshape(nb, 2)
    return nb, xb

xmin=3; xmax=6
func=lambda x: np.sin(np.dot(10.0, x))+np.cos(np.dot(3.0, x))

nb, xb=incsearch(func, 3, 6, 100)
print('number of brackets= ', nb)
print('root interval= ', xb)

number of brackets= 9
root interval= [[ 3.24242424  3.27272727]
                [ 3.36363636  3.39393939]
                [ 3.72727273  3.75757576]
                [ 4.21212121  4.24242424]
                [ 4.24242424  4.27272727]
                [ 4.6969697  4.72727273]
                [ 5.15151515  5.18181818]
                [ 5.18181818  5.21212121]
                [ 5.66666667  5.6969697 ]]
Strategy for Curve-fitting
Curve fitting Strategy

- **Fitting room**
  - a room in a store in which one can try on clothes before deciding whether to purchase them.
  - (Where to try on clothes before buying clothes)
  - Using lines to represent data instead
  - Trying on data with lines

- **Curve fitting Strategy (곡선 접합 전략)**
  - Curve fitting is the process of finding the curve that best fits a set of data points.
  - Least squares is a curve fitting method that minimizes the sum of squared residuals (errors here).
  - Residuals are the difference between the actual and predicted values.
Example of Curve fitting

- Data from Taiwan Central Meteorological Bureau Typhoon Database (http://rdc28.cwb.gov.tw/)
- Temporal trend linear regression joint by least squares analysis of gust wind (km/h)
- Typhoon landing in Taiwan
  - (a) speed of gusts (km/h)
  - (b) Total precipitation (mm)
  - (c) Time trend of precipitation (mm) up to 24 hours
Example of Curve fitting

• What is the connection between an individual's annual income and the price of the house they own?
• If the income is high, do you live in an expensive house?
• Take the example of Houston, USA.

https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/houston.py
Understanding linear regression through polyfilt() method

Understand the concepts of linear regression and polynomial regression through the numpy method polyfilt()
If the point on the blue straight line or the modeled point $y'$ is too well modeled and does not have an error, $y' = y$ is the strategy of curve jointing.

You need to find a straight line where the error is zero. That line is $y' = y$ and this is an optimization.

$y' = a_0 + a_1 x + e$

The reference point $x$ is the same. For reference point $x$, actual point is $y$. The modeled point is $y'$.
What about regression analysis?

- Regression analysis is given an instance with a given X, Y value.
- It is a supervised learning problem where you need to learn a function to be able to predict Y for an unknown X.
- Linear regression analysis is to find a linear function $y'$ given inputs x, y.
Simple Linear Regression through Polyfit


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

x = np.array([0, 1, 2, 3, 4, 5])
y = np.array([0, 0.8, 0.9, 0.1, -0.8, -1])
n = np.size(x)
b = (n*np.sum(x*y) - (np.sum(x)*np.sum(y))) / (n*np.sum(x**2) - (np.sum(x))**2)
# b = -0.30285714285714288: slope
a = (np.sum(y) - b*np.sum(x)) / n
# 0.75714285714285723: intercept

p1 = np.polyfit(x, y, 1)  # 1: linear array([-0.30285714, 0.75714286]) slope and intercept
plt.figure(1)
plt.plot(x, y, 'o')
plt.grid()
plt.plot(x, np.polyval(p1, x), 'r-')  # p1 from np.polyfit, plot(x, p1 with polyval

plt.figure(2)
plt.plot(x, y, 'o')
plt.grid()
plt.plot(x, p1[0]*x+p1[1], 'r-')
```
Simple Linear Regression through Polyfit

- Understand the concept of linear regression through the numpy method polyfilt()
Polynomial Regression through Polyfit

- Understand the concept of polynomial regression through the numpy method polyfilt()

```python
p2=np.polyfit(x, y, 2)  # quadratic second-order
# array([-0.16071429,  0.50071429,  0.22142857]) coefficient of quadratic second-order

p3=np.polyfit(x, y, 3)  # cubic
# array([ 0.08703704, -0.81349206,  1.69312169, -0.03968254]) coefficient of cubic third-order

plt.figure(2)
plt.plot(x, y, 'o')
plt.grid()
plt.plot(x, np.polyval(p1,x), 'r-')  # p1 from np.polyfit, plot(x, p1 with polyval
plt.plot(x, np.polyval(p2,x), 'b--')  # ---: dached line
#np.polyval(p2,x)= array([ 0.22142857,  0.56142857,  0.58  ,  0.27714286, -0.34714286, -1.29285714])
plt.plot(x, np.polyval(p3,x), 'm:')  # : dotted line

https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/regression.py
```
Linear & Polynomial Regression

• Visualizing linear regression and polynomial regression
Prediction (when -1, -2, and 6?)

```python
cmplts.figure(3)
cmplts.plot(x, y, 'o')
cmplts.grid()
xp = np.linspace(-2, 6, 100)
cmplts.plot(xp, np.polyval(p1, xp), 'r-')
cmplts.plot(xp, np.polyval(p2, xp), 'b--')
cmplts.plot(xp, np.polyval(p3, xp), 'm:')</code>`
Different data sets with linear regression

- Several data sets have the same linear regression, but their data show very different patterns.
Curve-fitting Strategy

Error Optimization

Numerical analysis is the study of approximation
Ideal Curve-fitting Strategy for $e_1, e_2, e_3$

$y'_1 = a_0 + a_1x + e_1$
$y'_2 = a_0 + a_1x + e_2$
$y'_3 = a_0 + a_1x + e_3$

There is only one straight line, but it is assumed that different requirements must be accommodated and are acceptable.
Requirements for an ideal Curve-fitting

- When all the requirements from the error standpoint are met, three different straight lines must be made.
  - 에러 입장에서 요구하는 사항을 다 들어주게 되면, 세 개의 다른 직선을 만들어야 한다.
- Another straight line means that the slope ($a_1$) and the intercept ($a_0$) are different.
  - 다른 직선이란 기울기 ($a_1$) 와 절편 ($a_0$) 이 다르다는 얘기이다.
- The straight lines we defined $y'_1$, $y'_2$, $y'_3$ consist of an intercept ($a_0$) with the same slope ($a_1$), so it becomes one straight line.
  - 우리가 정의한 직선 $y'_1$, $y'_2$, $y'_3$ 는 같은 기울기 ($a_1$)와 같은 절편 ($a_0$)으로 구성되어 있어서 하나의 직선이 된다.
  - 즉, 기울기 ($a_1$) 와 절편 ($a_0$) 이 다른 세 개의 직선을 만들지 않았다.
- The error values ($e_1$, $e_2$, $e_3$) created by one straight line have changed.
  - 한 개의 직선에 의해 만들어지는 에러 값 ($e_1$, $e_2$, $e_3$)이 달라졌다.
- The goal is to create a straight line consisting of one slope ($a_1$) and one intercept ($a_0$).
  - 하나의 기울기 ($a_1$)와 하나의 절편 ($a_0$)으로 구성된 하나의 직선을 만드는 것이 목표이다.

\[
\begin{align*}
y'_1 &= a_0 + a_1x + e_1 \\
y'_2 &= a_0 + a_1x + e_2 \\
y'_3 &= a_0 + a_1x + e_3
\end{align*}
\]
Error Optimization

Regression analysis

Fits a straight line to this messy scatterplot.

$x$ is called the independent or predictor variable, and $y$ is the dependent or response variable. The regression or prediction line has the form

$$y = a + bx$$
Choosing the next best option : Minimize the sum of errors

\[ e_1 = y_1 - a_0 - a_1 x_1 \]

Best optimization to be \( e_1 = 0 \)

\[ e_2 = y_2 - a_0 - a_1 x_2 \]

Best optimization to be \( e_2 = 0 \)

\[ e_3 = y_3 - a_0 - a_1 x_3 \]

Best optimization to be \( e_3 = 0 \)

\[ y' = a_0 + a_1 x + e \]

You have to accommodate each and every requirement, but you can't. The best way to do your best in situations where you can't accept is to make small yields so that the sum of errors is minimized.

minimize the sum of error \( (e_1 + e_2 + \ldots + e_n) = \min \sum_{i=1}^{n} e_i \)

\[ = \min \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i) \]
Choosing the next best option : Minimize the sum of errors

- From the standpoint of \( e_1 \), the dotted line (blue) is more optimized than the solid line
  - \( e_1 \) 입장에서는 실선 직선보다 점선 직선(파란색)이 더 최적화된 직선이다
- From the standpoint of \( e_2 \), the dotted line (red) is more optimized than the solid line
  - \( e_2 \) 입장에서는 실선 직선보다 점선 직선(빨강색)이 더 최적화된 직선이다
- \( e_1, e_2, e_3 \) you have to accommodate all of the requirements, but you can't.
  - \( e_1, e_2, e_3 \) 각각의 모든 요구 조건을 수용해야 하는 상황이지만, 그렇게 하지 못한다.
- If you can't accommodate all the requirements of \( e_1, e_2, e_3 \), how to do your best if you can't meet all the requirements of \( e_1, e_2, e_3 \). From the standpoint, you should throw away the request to make sure there are no errors, and accept some errors.
- The best way to do this is to give \( e_1, e_2, e_3 \) little by little, In the position of \( e_1, e_2, e_3 \), you must accept the request that there are absolutely no errors, and accept the request for a little error.
  - \( e_1, e_2, e_3 \) 의 모든 요구 조건을 수용하지 못하는 상황에서 최선을 다하는 방법은 \( e_1, e_2 \) 가 조금씩 양보를 해서 \( e_1 \), \( e_2 \) 입장에서는 절대 에러가 없게 해 달라는 요구를 버리고, 조금의 에러를 수용해야 한다.
- Overall, it is to minimize the sum of errors \( e_1, e_2, e_3 \).
  - 그리고, 전체적으로 보면, \( e_1, e_2, e_3 \) 에러의 합이 최소화되도록 하는 것이다.
Least Squares Regression (최소 제곱 회귀)

• To minimize the sum of errors $e_1, e_2, e_3$
  – 에러의 합이 최소화 되도록 하는 것이다.

• How to perform minimization is a matter of optimization.
  – 최소화를 어떻게 수행할 것인가는 최적화의 문제이다.
  – Minimize the sum of the squares of the errors → optimization

• How to do the optimization?
  – When there are two variables (Multi variables, $a_0, a_1$), it is necessary to apply a partial derivative method to find the maximum and minimum.
    • Find min, max → differentiation
    • Calculate slope ($a_1$) and intercept ($a_0$) in order to minimize the sum of error
    • Multi variables ($a_0, a_1$) → Partial Derivative

$$S = \min \sum_{i=1}^{n} e_i^2 = \min \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i)^2$$

Least Error Square Form (최소 에러 제곱 회귀)
Why squared? Squared error

• The error is an error of size \( \text{abs}(4)=4 \) and \( \text{abs}(-3)=3 \)
• The total size of the error should be 7
• Without abs(), adding the error results in \( 4+(-3)=1 \), and cannot be a measure of the total size of the error.
• So, in order to remove the minus sign of -3, \((-3)^*2\), square it.

\[
S = \min \sum_{i=1}^{n} e_i^2 = \min \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i)^2
\]

\[
\frac{\partial S}{\partial a_0} = \frac{\partial}{\partial a_0} \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i)^2 = 0
\]

\[
\frac{\partial S}{\partial a_1} = \frac{\partial}{\partial a_1} \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i)^2 = 0
\]
Multiple Linear Regression

- Weight (variable xs), 100kg, age (variable ys), and blood fat (variable zs) of a 40-year-old person are predicted
Comparison of Regression Methods for Curve Fitting

• Linear Regression (선형 회귀)
  – Perform curve fitting using linear equation of 1 variable $x$
  – use method of Numpy, scikit-learn, tensorflow

• Polynomial Regression (다항 회귀)
  – Perform curve fitting using the quadratic and cubic equations of $x$ of one variable ($x^2, x^3$)
  – use method of Numpy, scikit-learn, tensorflow

• Multiple Linear Regression (다변수 선형 회귀)
  – Perform curve fitting using linear equations of two or more variables $x, y$ or $x_1, x_2$
  – use method of scikit-learn, tensorflow

• General Linear Regression (일반 선형 회귀)
  – Perform more than one function, $z_1, z_2, \ldots, z_n$ curve fitting with $n$ linear functions of $z_n$
Experience the abstraction of TensorFlow through coding through linear regression
Install TensorFlow (Ubuntu)

- sudo apt update
- sudo apt install python3-dev python3-pip
- sudo pip3 install -U virtualenv
- pip install --upgrade pip
- pip install --upgrade tensorflow

- https://www.tensorflow.org/install/pip
Data generation

- Let's create data with the following data distribution
- https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/regression_class.py
Data generation

- Use the Data_Generation() method
- 50 data are generated by randomly generating x and y values.

```python
def Data_Generation(num_points):
    # num_points = 50
    vectors_set = []
    for i in np.arange(num_points):
        x = np.random.normal(2, 2) + 10
        y = x * 5 + (np.random.normal(0, 3)) * 2
        vectors_set.append([x, y])

    x_data = [v[0] for v in vectors_set]
    y_data = [v[1] for v in vectors_set]

    return x_data, y_data
```
Data Visualization

- This is a method of drawing the created 50 points.

```python
def Data_Draw(x_data, y_data):
    plt.figure(100)
    plt.plot(x_data, y_data, 'ro')
    plt.ylim([0, 100])
    plt.xlim([0, 25])
    plt.xlabel('x')
    plt.ylabel('y')
    #plt.legend()
    plt.show()```
Graphical Method

- When looking at the 50 points generated by the data, the slope is about 4.
- The intercept is expected to be between -10 and 10.
• Learn through the Data_Learning() method.
• Compute the slope (W) and y-intercept (b) values that provide the minimum error value (loss value) for 50 points of x_data and y_data given.
• Use the least squares method.
• Use the GradientDecentOptimizer() method.

```python
def Data_Learning(x_data, y_data):
    W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W * x_data + b
loss = tf.reduce_mean(tf.square(y - y_data))
#lo.append(loss)
optimizer = tf.train.GradientDescentOptimizer(0.0015) # 0.1, 0.01 0.001 0.0015
train = optimizer.minimize(loss)
init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)
```
TensorFlow variable concept, learning rate

• Initialize $W$, which is a variable for explaining the correlation between the 50 generated $x$ and $y$ coordinates, to a random value with a uniform distribution from -1.0 to 1.0, and initialize $b$ to 0.

• The optimal learning rate is specified by changing the learning rate of the GradientDescentOptimizer() method to 0.1, 0.01, 0.001, etc.

```python
def Data_Learning(x_data, y_data):
    W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
    b = tf.Variable(tf.zeros([1]))
    y = W * x_data + b
    loss = tf.reduce_mean(tf.square(y - y_data))
    #lo.append(loss)
    optimizer = tf.train.GradientDescentOptimizer(0.0015)  # 0.1, 0.01 0.001 0.0015
    train = optimizer.minimize(loss)
    init = tf.initialize_all_variables()
    sess = tf.Session()
    sess.run(init)
```
• It means that we will explain the relationship between \( x \) and \( y \) through the product of \( W \) and the sum of \( b \).

• Given \( x \), it means to find \( W \) and \( b \) that can produce \( y \).

• \( W: \) weight, \( b: \) bias

```python
def Data_Learning(x_data, y_data):
    W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
    b = tf.Variable(tf.zeros([1]))
    y = W * x_data + b
    loss = tf.reduce_mean(tf.square(y - y_data))
    optimizer = tf.train.GradientDescentOptimizer(0.0015)  # 0.1, 0.01 0.001 0.0015
    train = optimizer.minimize(loss)
    init = tf.initialize_all_variables()
    sess = tf.Session()
    sess.run(init)
```
Loss function or cost function

- The loss function is a function that calculates the loss value for a pair of data \((x, y)\).
- The loss value is a value indicating how much difference is between the actual value and the value predicted by the model.
- The smaller the loss value, the better the model explains the relationship between \(x\) and \(y\), and it means that it can accurately predict the \(y\) value for a given \(x\) value.
- This loss is called the cost when it is calculated for the entire data.
- Cost (abstraction function): \(\text{tf.reduce_mean}\)

```python
loss = tf.reduce_mean(tf.square(y - y_data))

cost = tf.reduce_mean(tf.square(hypothesis - Y))
```
Loss

Console Debugging for Loss (when xPos=-0.5)

```python
# hypothesis = tf.mul(W, X)

hypothesis = W*X

cost = tf.reduce_sum(tf.pow(hypothesis-Y, 2)) / m
```

Python Console

```
In[224]: -0.5

In[225]: hypothesis = xPos*X

In[226]: hypothesis

Out[226]: array([-0.5, -1. , -1.5])

In[227]: Y

Out[227]: [1.0, 2.0, 3.0]

In[228]: hypothesis[0]

Out[228]: -0.5

In[229]: hypothesis-Y

Out[229]: array([-1.5, -3. , -4.5])

In[230]: (hypothesis-Y)**2

Out[230]: array([ 2.25,  9. , 20.25])

In[231]: tf.pow(hypothesis-Y, 2)

Out[231]: <tf.Tensor 'Pow_1:0' shape=(3,) dtype=float64>

In[232]: sess.run(tf.pow(hypothesis-Y, 2))

Out[232]: array([ 2.25,  9. , 20.25])

In[233]: tf.reduce_sum(tf.pow(hypothesis-Y, 2))

Out[233]: <tf.Tensor 'Sum_1:0' shape=() dtype=float64>

In[234]: sess.run(tf.reduce_sum(tf.pow(hypothesis-Y, 2)))

Out[234]: 31.5
```
Console Debugging for Loss (when xPos=-0.5)

```python
# -5에서 20 사이, -0.5에서 2 사이 0.1씩 증가하면 총 25
for i in range(-5, 20):  # -30, 50
    xPos = i*0.1
    yPos = sess.run(cost, feed_dict={W: xPos})
    print('{:.3f}, {:.3f}'.format(xPos, yPos))
    W_val.append(xPos)
    cost_val.append(yPos)

sess.close()
print('size(W_val)=', np.size(W_val))
print('W_val=', W_val)
```

Update
Console Debugging for Loss (when xPos=-0.4)
What is learning?

- Learning is to calculate the values of $W$ and $b$ that minimize this loss value while calculating by putting various values of the variables ($W$: weight, $b$: bias).
- As the loss value, the distance between the predicted value and the actual value is most often used.
- The loss value is obtained by subtracting the actual value from the predicted value and then squared, and the cost is obtained by taking the average of the loss values for all data.

```python
W_val = array([-0.5, -0.4, -0.3, -0.2, -0.1, 0., 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1., 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9])
cost_val = array([10.5, 9.147, 7.89, 6.72, 5.65, 4.67, 3.78, 2.99, 2.29, 1.68, 1.17, 0.75, 0.42, 0.19, 0.05, 0., 0.05, 0.19, 0.42, 0.75, 1.17, 1.68, 2.29, 2.99, 3.78])
```
Steps to create a TensorFlow graph

- The following code creates TensorFlow graphs required for learning.
  - Since W and b are TensorFlow graphs, y created by W and b is also a TensorFlow graph.
  - You can check the value of the randomly generated x_data variable, but since W, b, and y are the TensorFlow graphs in the stage of creating the TensorFlow graph, their values cannot be checked in console debugging yet. W, b, y values are initialized.

```python
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W * x_data + b
loss = tf.reduce_mean(tf.square(y - y_data))
#lo.append(loss)
optimizer = tf.train.GradientDescentOptimizer(0.0015)  # 0.1, 0.01 0.001 0.0015
train = optimizer.minimize(loss)
init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)
```
Steps to create a TensorFlow graph

• In machine learning, you can't determine what the current data is until you run it.
  – Loss, optimizer, and train have not been executed yet. So, the result of running loss, optimizer, and train is not displayed.
  – \( \text{loss} = \text{tf.reduce\_mean}(\text{tf.square}(y - y\_data)) \)
  – \( \text{loss} \)
  – Out[59]: <tf.Tensor 'Mean:0' shape=() dtype=float32>

• \( \text{optimizer} = \text{tf.train.}\text{GradientDescentOptimizer}(0.0015) \)
  – \( \text{optimizer} \)
  – Out[60]:
    <tensorflow.python.training.gradient_descent.GradientDescentOptimizer at 0x22571d30b38>

• \( \text{train} = \text{optimizer.minimize(} \text{loss} \) \)
  – \( \text{train} \)
  – Out[61]: <tf.Operation 'GradientDescent' type=NoOp>
Steps to create a TensorFlow graph

• Since the value is not known until the run function is called, it is correct to print out a summary of all tensor objects that tells who they are for consistent processing.
  – W
    • Out[46]: <tf.Variable 'Variable:0' shape=(1,) dtype=float32_ref>
  – b
    • Out[48]: <tf.Variable 'Variable_1:0' shape=(1,) dtype=float32_ref>
  – loss
    • Out[52]: <tf.Tensor 'Mean:0' shape=() dtype=float32>

• A session is required to run.
• TensorFlow can be started by calling the run function included in the session.
Graph execution must be done in Session. Graph execution uses Session object and run method

```python
init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)
train_set = []
for step in np.arange(20):
sess.run(train)
print(step, sess.run(W), sess.run(b))
print(step, sess.run(loss))
train_set.append([sess.run(W), sess.run(b), sess.run(loss)])
plt.figure(step)
plt.plot(x_data, y_data, 'ro')
plt.plot(x_data, sess.run(W) * x_data + sess.run(b))
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

```
W_data = [t[0] for t in train_set]
v_data = [t[1] for t in train_set]
Loss_data = [t[2] for t in train_set]
return W_data, v_data, Loss_data
```
TensorFlow graph Execution steps

• Initialize TensorFlow variables.
  – `init = tf.initialize_all_variables()`
  – `init`
  – Out[63]: <tf.Operation 'init_1' type=NoOp>

• Create a Session object.
  – `sess = tf.Session()`
tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX AVX2

• Execute the init method through the run method of the Session object.
  – `sess.run(init)`
Through the run method of the Session object, execute the train method.

```
sess.run(train)
- loss = tf.reduce_mean(tf.square(y - y_data))
- train = optimizer.minimize(loss)
```

The train() method outputs W and b values, which are the 0th training results after executing the loss() method.

```
print(step, sess.run(W), sess.run(b))
```

- 0 [3.553465] [0.31483752]
  - step 0th: W value is 3.553465, b value is 0.314
  - Gradient 4 degree in Graphical Method
  - Intercept matches expected values between -
The loss value at Step 0 is output.
  – print(step, sess.run(loss))
  – 0 367.4683
  – The loss value is 367.46.

Store the W, b, and loss values in Step 0 in train_set.
  – train_set.append([sess.run(W), sess.run(b), sess.run(loss)])
  – train_set
    • [[array([3.553465], dtype=float32), array([0.31483752], dtype=float32), 367.4683]]
Console debugging of loss value at Step 0

- 0th Step learning result, W, b, and loss values are calculated and output.

```python
sess.run(train)
print(step, sess.run(W), sess.run(b))
print(step, sess.run(loss))
train_set.append([sess.run(W), sess.run(b), sess.run(loss)])
```

![Graph](image.png)
Using the values of $W$ and $b$, which are the 0th learning results, the predicted line ($y$) is output on the $x_{\text{data}}$ and $y_{\text{data}}$ coordinates.

```python
plt.figure(step)
plt.plot(x_data, y_data, 'ro')
plt.plot(x_data, sess.run(W) * x_data + sess.run(b))
plt.xlabel('x')
plt.ylabel('y')
# plt.legend()
plt.show()
```
Console debugging of loss value at Step 0

• \( y = x_{\text{data}} \times W + b \)
  - `<tf.Tensor 'add_2:0' shape=(50,) dtype=float32>`
• If you execute `sess.run(y)`, you can check the \( y \) graph value.
• If \( y = \text{sess.run}(y) \) is executed, the \( y \) value is stored in the \( y \) variable.

```
array([42.92975 , 40.00953 , 46.08135 , 45.447483, 49.918053, 51.624714,
  34.05071 , 52.764053, 57.15245 , 42.98017 , 51.312218, 57.04682 ,
  43.72928 , 41.14911 , 43.657482, 39.395763, 44.090137, 38.637604,
  45.117508, 46.29372 , 51.215588, 44.00339 , 50.25373 , 36.61917 ,
  38.0987 , 36.795113, 45.256386, 34.173832, 35.144196, 45.60984 ,
  53.855278, 46.03917 , 40.39592 , 32.584106, 53.79027 , 49.385914,
  50.0337 , 47.026936, 36.007835, 52.042915, 39.187145, 48.742886,
  46.10148 , 30.20313 , 43.508713, 47.80955 , 59.52558 , 56.052826,
  38.24937 , 50.064648], dtype=float32)
```
Console debugging of loss value at Step 0

- tf.square(y-y_data) is executed.
  - Out[97]: <tf.Tensor 'Square_5:0' shape=(50,) dtype=float64>
- To check the result value of tf.square(y-y_data), execute 
  `sess.run(tf.square(y-y_data))`.

```python
array([3.55887660e+02, 2.19515568e+02, 7.65990154e+02, 2.00963266e+02,
       4.24603277e+02, 1.86404216e+02, 2.86510325e+02, 8.83120084e+02,
       3.75882401e+02, 2.44313768e+02, 2.68821511e+02, 5.83241280e+02,
       7.93462849e+02, 9.42321300e+01, 3.23208629e+02, 1.20379282e+02,
       8.60770028e+01, 3.3349419e+02, 2.02680634e+02, 4.52072610e+02,
       3.65716953e+02, 4.54534894e+02, 2.9317497e+02, 9.05798690e+01,
       7.37774252e+02, 2.20189581e+02, 4.01763774e+02, 1.37117623e+02,
       5.06256283e+01, 3.52435534e+02, 3.98363520e+02, 1.35701994e+02,
       4.68060278e+02, 1.47089562e+02, 3.93036451e+02, 3.36543665e+02,
       3.00211507e+02, 3.66502536e+02, 7.8853182e+01, 2.48265203e+02,
       8.26627877e+02, 4.21276920e+02, 8.77037917e+02, 1.29749703e+02,
       6.73178502e+02, 5.18197669e+02, 7.35094790e+02, 5.42157841e+02,
       1.54648405e+02, 4.32132662e+02])
```
Console debugging of loss value at Step 0

- `tf.reduce_mean(tf.square(y-y_data))` is executed.
- To check the result of `tf.reduce_mean(tf.square(y-y_data))`, execute `sess.run(tf.reduce_mean(tf.square(y-y_data)))`.
  - `sess.run(tf.reduce_mean(tf.square(y - y_data)))`
  - Out[106]: 367.46829244768355
- The loss output value in Step 0 was checked by console debugging.
  - `print(step, sess.run(loss))`
  - 0 367.4683
  - The loss value is 367.46.
Console debugging of loss value at Step 0

- The loss value at Step 0 is stored as the output value, W, b, and loss through the code below.
  - \( W_{\text{data}} = [t[0] \text{ for } t \text{ in train_set}] \)
  - \( b_{\text{data}} = [t[1] \text{ for } t \text{ in train_set}] \)
  - \( \text{Loss}_{\text{data}} = [t[2] \text{ for } t \text{ in train_set}] \)

- \( W_{\text{data}} \)
  - Out[112]: [array([3.553465], dtype=float32)]

- \( b_{\text{data}} \)
  - Out[113]: [array([0.31483752], dtype=float32)]

- \( \text{Loss}_{\text{data}} \)
  - Out[114]: 367.4683
Console debugging of loss value in Step 1

• The training results $W$, $b$, and loss of Step 1 are calculated and output.

```python
sess.run(train)
print(step, sess.run(W), sess.run(b))
print(step, sess.run(loss))
train_set.append([sess.run(W), sess.run(b), sess.run(loss)])
```

```
step=1
sess.run(train)
print(step, sess.run(W), sess.run(b))
1 [4.2541637] [0.3692001]
print(step, sess.run(loss))
1 118.703514
```
Console debugging of loss value in Step 1

- Using the values of $W$ and $b$, which are the learning results of Step 1, the predicted line ($y$) is output on the $x_data$ and $y_data$ coordinates. The straight line moved more toward the data than in the Step 0.

```python
plt.figure(step)
plt.plot(x_data, y_data, 'ro')
plt.plot(x_data, sess.run(W) * x_data + sess.run(b))
plt.xlabel('x')
plt.ylabel('y')
# plt.legend()
plt.show()

cp = 1
sess.run(train)
print(step, sess.run(W), sess.run(b))
1 [4.2541637] [0.3692001]
print(step, sess.run(loss))
1 118.703514
```
Console debugging of loss value in Step 1

- \( y = x_{\text{data}} \times W + b \)
  
  - `<tf.Tensor 'add_2:0' shape=(50,) dtype=float32>`

- If you execute `sess.run(y)`, you can check the y graph value.
- If `y = sess.run(y)` is executed, the y value is stored in the y variable.

### y value: Step 0

```python
array([42.92975 , 40.00953 , 46.08135 , 45.447483, 49.918053, 51.624714, 34.05071 , 52.764053, 57.15245 , 42.98017 , 51.312218, 57.04682 , 43.72928 , 41.14911 , 43.657482, 39.395763, 44.090137, 38.637604, 45.117508, 46.29372 , 51.215588, 44.00339 , 50.25373 , 36.61917 , 38.0987 , 36.795113, 45.256386, 34.173832, 35.144196, 45.60984 , 53.855278, 46.03917 , 40.39592 , 32.584106, 53.79027, 49.835914, 50.0337 , 47.026936, 36.007835, 52.042915, 39.187145, 48.742886, 46.10148 , 30.20313 , 43.508713, 47.80955 , 59.52558 , 56.052826, 38.24937 , 50.064648], dtype=float32)
```

### y value: Step 1

```python
array([51.387238, 47.89119 , 55.160294, 54.401443, 59.75355 , 61.796745, 40.757362, 63.160744, 68.41448 , 51.447605, 61.422626, 68.288025, 52.344245, 49.25548 , 52.258472, 47.1564 , 52.776443, 46.248737, 54.006397, 55.414543, 61.30694 , 52.672592, 60.155422, 43.832294, 45.603573, 44.042927, 54.17266 , 40.904762, 42.066475, 54.595814, 64.46715 , 55.109802, 48.35377 , 39.001564, 64.38932 , 59.116478, 59.892 , 56.29234 , 43.10041 , 62.29741 , 46.906643, 58.346653, 55.184395, 36.151093, 52.08037 , 57.22928 , 71.25556 , 67.09802 , 45.78395 , 59.92905 ], dtype=float32)
```

### y_data

```python
y_data = np.array(y_data)
```
### Console debugging of loss value in Step 1

- Execute `tf.square(y - y_data)`
  - `Out[97]: <tf.Tensor 'Square_5:0' shape=(50,) dtype=float64>`
- To check the result value of `tf.square(y-y_data)`, execute `sess.run(tf.square(y - y_data))`

#### y-y_data of Step 0

```python
array([3.55887660e+02, 2.19515568e+02, 7.6590154e+02, 2.00963266e+02, 4.24603277e+02, 1.86404216e+02, 2.86510325e+02, 8.83120084e+02, 3.75882401e+02, 2.44313768e+02, 2.6821511e+02, 5.83241280e+02, 7.93462849e+02, 9.42321300e+01, 3.23208629e+02, 1.20379282e+02, 8.60770028e+01, 3.3349419e+02, 2.02680634e+02, 4.52072610e+02, 3.65716953e+02, 4.54534894e+02, 2.29317497e+02, 9.05798690e+01, 7.3774252e+02, 2.20189581e+02, 4.01763774e+02, 1.37117623e+02, 5.06256283e+01, 3.52435534e+02, 3.98363520e+02, 1.04447629e+02, 3.73287364e+02, 6.60262740e+01, 5.13100828e+01, 3.95058567e+01, 1.66647071e+02, 3.82332557e+02, 2.56304312e+00, 8.79283866e+01, 1.03112524e+01, 3.49828809e+01, 1.13352765e+02, 2.59795459e+01, 1.27081626e+02, 8.15838329e+01, 1.60038264e+02, 2.74737550e+01, 5.30944073e+00, 3.86402712e+02, 5.76228590e+01, 1.23827284e+02, 2.47883809e+01, 5.82748451e+01, 9.57911065e+01, 8.73697041e+01, 6.48594880e+00, 1.87056209e+02, 3.26109493e+01, 8.51208901e+01, 7.42106844e+01, 5.57757055e+01, 9.75917209e+01, 3.20111370e+00, 3.02713436e+01, 4.42329851e+02, 1.19274011e+02, 4.21559316e+02, 2.96241908e+01, 3.01856731e+02, 1.78068425e+02, 2.36625955e+02, 1.49795241e+02, 2.40216727e+01, 1.19320624e+02])
```

#### y-y_data of Step 1

```python
array([1.08315973e+02, 4.80858661e+01, 3.45870055e+02, 2.72712776e+01, 1.16001627e+02, 1.21170931e+01, 1.04447629e+02, 3.73287364e+02, 6.60262740e+01, 5.13100828e+01, 3.95058567e+01, 1.66647071e+02, 3.82332557e+02, 2.56304312e+00, 8.79283866e+01, 1.03112524e+01, 3.49828809e-01, 1.13352765e+02, 2.59795459e+01, 1.27081626e+02, 8.15838329e+01, 1.60038264e+02, 2.74737550e+01, 5.30944073e+00, 3.86402712e+02, 5.76228590e+01, 1.23827284e+02, 2.47883809e+01, 5.82748451e+01, 9.57911065e+01, 8.73697041e+01, 6.48594880e+00, 1.87056209e+02, 3.26109493e+01, 8.51208901e+01, 7.42106844e+01, 5.57757055e+01, 9.75917209e+01, 3.20111370e+00, 3.02713436e+01, 4.42329851e+02, 1.19274011e+02, 4.21559316e+02, 2.96241908e+01, 3.01856731e+02, 1.78068425e+02, 2.36625955e+02, 1.49795241e+02, 2.40216727e+01, 1.19320624e+02])
```
Console debugging of loss value in Step 1

• \( \text{tf.reduce\_mean} (\text{tf.square}(\ y - y\_\text{data})) \) is executed.

• To check the result of \( \text{tf.reduce\_mean} (\text{tf.square}(\ y - y\_\text{data})) \), execute \( \text{sess.run} (\text{tf.reduce\_mean} (\text{tf.square}(\ y - y\_\text{data}))) \)

• \( \text{sess.run} (\text{tf.reduce\_mean} (\text{tf.square}(\ y - y\_\text{data}))) \)

  – Out[106]: **367.46829244768355**  
  – Out[131]: **118.70350861527437**

• loss value in Step 1 was confirmed by console debugging.

  – print(step, sess.run(loss))

  – 0  367.4683  
  – 1  118.703514

  – loss 값은 118.70 이다.
Console debugging of loss value in Step 2

- Calculate and output learning result of the 2\textsuperscript{nd} step, W, b, and loss values.

```
sess.run(train)
print(step, sess.run(W), sess.run(b))
print(step, sess.run(loss))
train_set.append([sess.run(W), sess.run(b), sess.run(loss)])
```

<table>
<thead>
<tr>
<th>Step</th>
<th>W</th>
<th>b</th>
<th>Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.2541637</td>
<td>0.3692001</td>
<td>118.703514</td>
</tr>
<tr>
<td>2</td>
<td>4.6121655</td>
<td>0.3969517</td>
<td>53.766617</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Console debugging of loss value in Step 2

• Using the values of W and b, which are the second learning results, the predicted line (y) is output on the x_data and y_data coordinates. The straight line moved more towards the data.

    plt.figure(step)
    plt.plot(x_data, y_data, 'ro')
    plt.plot(x_data, sess.run(W) * x_data + sess.run(b))
    plt.xlabel('x')
    plt.ylabel('y')
    #plt.legend()
    plt.show()

step=1
sess.run(train)
print(step, sess.run(W), sess.run(b))
1 [4.2541637] [0.3692001]
print(step, sess.run(loss))
1 118.703514

step=2
sess.run(train)
print(step, sess.run(W), sess.run(b))
2 [4.6121655] [0.39695176]
print(step, sess.run(loss))
2 53.766617
Loss according to W

- Loss according to W
Tensor operation

- If you do print(c), you can think that 42 will come out, but it will print in the form of a tensor.
  - The reason is that the TensorFlow program structure is divided into two types: 1. graph generation and 2. graph execution.

```python
a=tf.constant(10)
<tf.Tensor 'Const_1:0' shape=() dtype=int32>

b=tf.constant(32)
<tf.Tensor 'Const_2:0' shape=() dtype=int32>

c=tf.add(a,b)
<tf.Tensor 'Add:0' shape=() dtype=int32>
```

When to actually perform the operation

```python
print(sess.run(hello))
b'Hello, TensorFlow!'

print(sess.run([a, b, c]))
[10, 32, 42]
```

10+32, 42

b'Hello, TensorFlow!'
Graphing and Lazy Evaluation

• Graph: A collection of tensors' operations
  – Create a graph by first defining a tensor and its operations
  – After that, the actual operation is performed at the 'desired point' by putting the code that executes the operation when necessary.
  – Called Lazy Evaluation

• Graph execution must be done in Session

• Using Session object and run method

```python
sess=tf.Session()
print(sess.run(hello))
'b'Hello, TensorFlow!'

print(sess.run([a, b, c]))
[10, 32, 42]
sess.close()
```
```python
import tensorflow as tf

def hello():
    a = tf.constant('hello, tensorflow!')
    print(a)  # Tensor("Const:0", shape=(), dtype=string)

    sess = tf.Session()
    result = sess.run(a)

    # 2.x 버전에서는 문자열로 출력되지만, 3.x 버전에서는 byte 자료형
    # 문자열로 변환하기 위해 decode 함수로 변환
    print(result)  # b'hello, tensorflow!'
    print(type(result))  # <class 'bytes'>
    print(result.decode(encoding='utf-8'))  # hello, tensorflow!
    print(type(result.decode(encoding='utf-8')))  # <class 'str'>

    # 세션 닫기
    sess.close()

if '__name__' == '__main__':
    hello()

Tensor("Const_5:0", shape=(), dtype=string)
b'hello, tensorflow!'
<class 'bytes'>
hello, tensorflow!
<class 'str'>
```

https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/tf_type.py
Gradient Descent Optimization Function

- Using the gradient descent optimization function provided by TensorFlow, an operation graph that minimizes the loss value is created.

\[
\text{optimizer}=\text{tf.train.GradientDescentOptimizer}(\text{learning\_rate}=0.1) \\
\text{train\_op}=\text{optimizer}.\text{minimize}(\text{cost})
\]
Gradient Descent Optimization

• The optimization function is a function that finds the most optimized weight (W) and bias (b) values that minimize the loss value while changing the weight (W) and bias (b) values.
• Randomly changing the weight (W) and bias (b) values takes too long and it is difficult to predict the learning time.
• One of the fast optimization methods is gradient descent.
• Gradient descent is the most basic algorithm among optimization methods, and it is a method of finding the optimal value while continuing to move in the negative slope direction.
Learning rate : hyperparameter

• The learning rate is a value that sets how urgent learning is done.
• If the learning rate is too large, the optimum loss value cannot be found and it is excessive, and if the value is too small, the learning speed is very slow.
• If a variable that affects the learning process is called a hyperparameter, the learning speed or the performance of the neural network greatly varies according to this value.
• In machine learning, tuning hyperparameters well is a big challenge.
Now that the linear regression model has been created, let's run the graph to train it and check the result.

Create a session block using the Python with function, and handle session termination automatically.

Execute train_op, a graph that performs optimization, and output the loss value that changes with each execution.

When training is performed 100 times, input x_data and y_data, which are data to find out the correlation, through the feed_dict parameter.
Perform Learning

- Put the test values in the optimized model and check if the results come out well.

```python
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(100):
        _, cost_val=sess.run([train_op, cost], feed_dict={X: x_data, Y: y_data})
        print(step, cost_val, sess.run(W), sess.run(b))

print("\n=== Test ===")
print("X: 5, Y:", sess.run(hypothesis, feed_dict={X: 5}))
print("X: 2.5, Y:", sess.run(hypothesis, feed_dict={X: 2.5}))
```
Output of learning progress

• Step, loss value, [W value], [b value]
  – 0 0.869386 [0.9130715] [0.3043009]
  – 1 0.02205333 [0.87248445] [0.27821213]
  – 2 0.011377501 [0.88021415] [0.27357593]
  – 3 0.010722056 [0.8825839] [0.26677507]
  – 4 0.010211366 [0.8854622] [0.2603865]
  – 5 0.009726319 [0.8882096] [0.2541243]
  – 6 0.009264297 [0.8908976] [0.24801561]
  – 7 0.008824232 [0.89352024] [0.24205346]
  – 8 0.008405092 [0.89607996] [0.23623468]
  – 9 0.008005846 [0.8985781] [0.23055576]
  – 10 0.00762555 [0.90101624] [0.22501338]
Test Learning

- Step, loss value, [W value], [b value]
  - 99 0.000100282195 [0.98864883] [0.02580387]

- By learning, the W value that makes x=[1,2,3], y=[1,2,3] is 0.988, which is close to 1, and the b value is 0.025, which is close to 0.

- In the test, we have a set [W value] and [b value], and when X: 5 comes in, the Y value is predicted, and Y: [4.969048] is predicted. (Close to 5)
  - When X: 2.5 comes in, the Y value is predicted, and Y: [2.4974258] is predicted. (Close to 2.5)

  === Test ===
  - X: 5, Y: [4.969048]
  - X: 2.5, Y: [2.4974258]
Polynomial Regression

https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/regression_polynomial.py
Polynomial Regression: Solving through Matrix Expansion
Matrix expansion: np.linalg.solve() method

• Try Curve Fitting with Polynomial Regression
• https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/regression_polynomial.py

Therefore, the simultaneous linear equations are

\[
\begin{bmatrix}
6 & 15 & 55 \\
15 & 55 & 225 \\
55 & 225 & 979 \\
\end{bmatrix}
\begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
\end{bmatrix}
= 
\begin{bmatrix}
152.6 \\
585.6 \\
2488.8 \\
\end{bmatrix}
\]

These equations can be solved to evaluate the coefficients. For example, using MATLAB:

```matlab
>> N = [6 15 55; 15 55 225; 55 225 979];
>> r = [152.6 585.6 2488.8];
>> a = N \ r
a = 
 2.4786
 2.3593
 1.8607
```

Therefore, the least-squares quadratic equation for this case is

\[ y = 2.4786 + 2.3593x + 1.8607x^2 \]
Matrix expansion: np.linalg.solve() method

- Try Curve Fitting with Polynomial Regression

\[
\begin{bmatrix}
    n & \sum x_i & \sum x_i^2 \\
    \sum x_i & \sum x_i^2 & \sum x_i^3 \\
    \sum x_i^2 & \sum x_i^3 & \sum x_i^4 
\end{bmatrix} \begin{bmatrix}
    a_0 \\
    a_1 \\
    a_2 
\end{bmatrix} = \begin{bmatrix}
    \sum y_i \\
    \sum x_i y_i \\
    \sum x_i^2 y_i 
\end{bmatrix}
\]

# 편미분을 이용한 다항 회귀
a=np.array([[np.size(x), np.sum(x), np.sum(x**2)],
            [np.sum(x), np.sum(x**2), np.sum(x**3)],
            [np.sum(x**2), np.sum(x**3), np.sum(x**4) ] ])
# array([[  6,  15,  55],
#        [ 15,  55, 225],
#        [ 55, 225, 979]])
b=np.array([np.sum(y), np.sum(x*y),
            np.sum(x**2*y)])
# array([ 152.6,  585.6, 2488.8])
t=np.linalg.solve(a, b)
# array([2.47857143, 2.35928571, 1.86071429])
x1=np.linspace(0,5,10)
y1=t[2]*x1**2+t[1]*x1+t[0]
# derived from polynomial regression
plt.figure(2)
plt.plot(x,y,'ro', x1, y1, 'b*:')
plt.legend(['Real Data','Polynomial Regression by Partial Derivative'])
plt.grid()
np.linalg.solve() method

- The np.linalg.solve() method is a method that gets the solution of a matrix.
- Solve a linear matrix equation, or system of linear scalar equations.

\[ y_1 = 1.8607x_1^2 + 2.3593x_1 + 2.4786 \]

```python
# 편미분을 이용한 다항 회귀
a=np.array([[np.size(x), np.sum(x), np.sum(x**2)], [np.sum(x), np.sum(x**2), np.sum(x**3)], [np.sum(x**2), np.sum(x**3), np.sum(x**4)]]
#array([[  6,  15,  55],
#       [ 15,  55, 225],
#       [ 55, 225, 979]])
b=np.array([np.sum(y), np.sum(x*y), np.sum(x**2*y)])
# array([ 152.6,  585.6, 2488.8])
t=np.linalg.solve(a, b)
# array([2.47857143, 2.35928571, 1.86071429])
```
Solving through Matrix Expansion

- Python Code for Polynomial Regression
  - Try Curve Fitting with Polynomial Regression

```python
x1=np.linspace(0,5,10)
y1=t[2]*x1**2+t[1]*x1+t[0]
# derived from polynomial regression
plt.figure(2)
plt.plot(x,y,'ro', x1, y1, 'b*:
plt.legend([ 'Real Data', 'Polynomial Regression by Partial Derivative' ])
plt.grid()
```
Polynomial Regression: Solving through Matrix Expansion

- Solving polynomial regression through matrix expansion with partial derivative

https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/regression_polynomial.py
np.polyfit(x, y, 2): Polynomial Regression

- Polynomial Regression: Try Curve Fitting with np.polyfit(x, y, 2)
- Try Curve Fitting with np.polyfit(x, y, 2)

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

# np.polyfit(x, y, 2)를 이용한 다항 회귀
x = np.array([0, 1, 2, 3, 4, 5])
y = np.array([2.1, 7.7, 13.6, 27.2, 40.9, 61.1])

p2 = np.polyfit(x, y, 2)
# array([1.86071429, 2.35928571, 2.47857143])
plt.figure(1)
plt.plot(x, y, 'ro', np.polyval(p2, x), 'b*-' ) # x, p2[0]*x**2+p2[1]*x+p2[2]
plt.legend(['Real Data', 'Polynomial Regression by np.polyfit(x,y,2)'])
plt.grid()
```

https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/regression_polynomial.py
np.polyfit(x, y, 2): Polynomial Regression

- Polynomial Regression: Try Curve Fitting with np.polyfit(x, y, 2)
- Try Curve Fitting with np.polyfit(x, y, 2)
Polynomial Regression: Try Curve Fitting with Partial Differentiation

- Try polynomial regression with partial derivative

```python
# 편미분을 이용한 다항 회귀
a=np.array([[np.size(x), np.sum(x), np.sum(x**2)], [np.sum(x), np.sum(x**2), np.sum(x**3)], [np.sum(x**2), np.sum(x**3), np.sum(x**4)]]))
# array([[  6,  15,  55],
# #       [ 15,  55, 225],
# #       [ 55, 225, 979]])
b=np.array([np.sum(y), np.sum(x*y), np.sum(x**2*y)])
# array([ 152.6,  585.6, 2488.8])
t=np.linalg.solve(a, b)
# array([2.47857143, 2.35928571, 1.86071429])
x1=np.linspace(0,5,10)
y1=t[2]*x1**2+t[1]*x1+t[0]
# derived from polynomial regression
plt.figure(2)
plt.plot(x,y,'ro', x1, y1, 'b*:')
plt.legend(['Real Data','Polynomial Regression by Partial Derivative'])
plt.grid()
```

https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/regression_polynomial.py
Polynomial Regression: Try Curve Fitting with Partial Differentiation

- Try polynomial regression with partial derivative

https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/regression_polynomial.py
Polynomial regression of the np.polyfit() method

- Let's try np.polyfit(x,y,7) (7th order term)

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

# 구간 [0,4*pi]에 사인 곡선을 따라 균일한 간격의 점 100개를 생성합니다.
x = np.linspace(0,4*np.pi,100)
y=np.sin(x)

# 구간 [0,4*pi]에 따라 사인 곡선을 시각화한다.
plt.figure(1)
plt.plot(x, y)
# np.polyfit를 사용하여 이들 점에 7차 다항식을 피팅합니다.
p = np.polyfit(x,y,7)
# 좀 더 촘촘한 그리드에서 다항식을 계산하고 결과를 플로팅합니다.
x1 = np.linspace(0, 4*np.pi)  
y1 = np.polyval(p,x1)
plt.figure(2)
plt.plot(x, y, x1, y1, 'r*')
plt.grid()
```
Polynomial regression of the np.polyfit() method

- Let's try np.polyfit(x,y,7) (7th order term)
Multivariable linear regression

Understand multivariable linear regression through TensorFlow
Multivariable Linear Regression

- Perform a curve joint using a linear equation of two variables ($x$, $y$ or $x_1, x_2$)

\[ y = a_0 + a_1 x_1 + a_2 x_2 + e \]
Comparison of polynomial and multivariable regressions

- Similar to the derivation coefficient of the polynomial regression $y_i - a_0 - a_1 x_i - a_2 x_i^2$, the coefficient of multivariable regression $y_i - a_0 - a_1 x_i - a_2 z_i$ can be derived

- the derivation coefficient of polynomial regression
  
  $S = \min \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 x_i^2)^2$
  
  $\frac{\partial S}{\partial a_0} = -2 \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 x_i^2)$
  
  $\frac{\partial S}{\partial a_1} = -2 \sum_{i=1}^{n} x_i (y_i - a_0 - a_1 x_i - a_2 x_i^2)$
  
  $\frac{\partial S}{\partial a_2} = -2 \sum_{i=1}^{n} x_i^2 (y_i - a_0 - a_1 x_i - a_2 x_i^2)$

- System of polynomial regression equations
  
  $(n)a_0 + (\sum_{i=1}^{n} x_i) a_1 + (\sum_{i=1}^{n} x_i^2) a_2 = \sum_{i=1}^{n} y_i$

  $(\sum_{i=1}^{n} x_i) a_0 + (\sum_{i=1}^{n} x_i^2) a_1 + (\sum_{i=1}^{n} x_i^3) a_2 = \sum_{i=1}^{n} x_i y_i$

  $(\sum_{i=1}^{n} x_i^2) a_0 + (\sum_{i=1}^{n} x_i^3) a_1 + (\sum_{i=1}^{n} x_i^4) a_2 = \sum_{i=1}^{n} x_i^2 y_i$
Comparison of polynomial and multivariable regressions

- Similar to the derivation coefficient of the polynomial regression $y_i - a_0 - a_1 x_i - a_2 x_i^2$, the coefficient of multivariable regression $y_i - a_0 - a_1 x_i - a_2 z_i$ can be derived

- the derivation coefficient of multivariable regression
  \[
  S = \min \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 z_i)^2
  \]
  \[
  \frac{\partial S}{\partial a_0} = -2 \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 z_i)
  \]
  \[
  \frac{\partial S}{\partial a_1} = -2 \sum_{i=1}^{n} x_i (y_i - a_0 - a_1 x_i - a_2 z_i)
  \]
  \[
  \frac{\partial S}{\partial a_2} = -2 \sum_{i=1}^{n} z_i (y_i - a_0 - a_1 x_i - a_2 z_i)
  \]

- System of multivariable regression equations
  \[
  na_0 + (\sum x_i)a_1 + (\sum z_i)a_2 = \sum y_i
  \]
  \[
  (\sum x_i)a_0 + (\sum x_i^2)a_1 + (\sum x_i \cdot z_i)a_2 = \sum x_i \cdot y_i
  \]
  \[
  (\sum z_i)a_0 + (\sum x_i \cdot z_i)a_1 + (\sum z_i^2)a_2 = \sum_{i=1}^{n} z_i \cdot y_i
  \]
Multi-variable Regression (다변수 회귀)

- 편미분으로 Multi-variable Regression (다변수 회귀) 유도하기

\[
\frac{\partial S}{\partial a_0} = \frac{\partial}{\partial a_0} \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 z_i)^2 = 0
\]

\[y = \{f(x)\}^n\]
\[y' = n \cdot \{f(x)\}^{n-1} \cdot f'(x)\]

\[
= 2 \cdot \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 z_i)^{2-1} \cdot \frac{\partial (y_i - a_0 - a_1 x_i - a_2 z_i)}{\partial a_0}
\]

\[
= 2 \cdot \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 z_i) \cdot (-1)
\]

\[
= -2 \cdot \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 z_i) = 0
\]

\[
= \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} a_0 - \sum_{i=1}^{n} a_1 x_i - \sum_{i=1}^{n} a_2 z_i = 0
\]

- \(S = \min \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 z_i)^2\)
- \(\frac{\partial S}{\partial a_0} = -2 \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 z_i)\)
- \(\frac{\partial S}{\partial a_1} = -2 \sum_{i=1}^{n} x_i (y_i - a_0 - a_1 x_i - a_2 z_i)\)
- \(\frac{\partial S}{\partial a_2} = -2 \sum_{i=1}^{n} z_i (y_i - a_0 - a_1 x_i - a_2 z_i)\)
Multi-variable Regression

Derive Multi-variable Regression with Partial Differentiation

\[ \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} a_0 - \sum_{i=1}^{n} a_1 x_i - \sum_{i=1}^{n} a_2 z_i = 0 \]

\[ \sum_{i=1}^{n} a_0 + \sum_{i=1}^{n} a_1 x_i + \sum_{i=1}^{n} a_2 z_i = \sum_{i=1}^{n} y_i \]

\[ n \cdot a_0 + \sum_{i=1}^{n} a_1 x_i + \sum_{i=1}^{n} a_2 z_i = \sum_{i=1}^{n} y_i \]

\[ \sum_{i=1}^{n} a_0 = a_0 \cdot \sum_{i=1}^{n} 1 = a_0 \cdot (1 + 1 + \cdots + 1) = a_0 \cdot n = n \cdot a_0 \]

\[ n a_0 + \left( \sum x_i \right) a_1 + \left( \sum z_i \right) a_2 = \sum y_i \quad \text{①} \]

If you keep developing equation ① is derived for \( a_0 \)
Multi-variable Regression

- Derive Multi-variable Regression with Partial Differentiation

\[
\frac{\partial S}{\partial a_1} = \frac{\partial}{\partial a_1} \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 z_i)^2 = 0
\]

\[
y = \{f(x)\}^n
\]

\[
y' = n \cdot \{f(x)\}^{n-1} \cdot f'(x)
\]

\[
= 2 \cdot \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 z_i)^{2-1} \cdot \frac{\partial (y_i - a_0 - a_1 x_i - a_2 z_i)}{\partial a_1}
\]

\[
= -2x_i \cdot \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 z_i) = 0
\]

\[
= x_i \cdot \sum_{i=1}^{n} y_i - x_i \cdot \sum_{i=1}^{n} a_0 - x_i \cdot \sum_{i=1}^{n} a_1 x_i - x_i \cdot \sum_{i=1}^{n} a_2 z_i = 0
\]

\[
= \sum_{i=1}^{n} x_i \cdot y_i - \sum_{i=1}^{n} x_i \cdot a_0 - \sum_{i=1}^{n} a_1 x_i \cdot x_i - \sum_{i=1}^{n} a_2 x_i \cdot z_i = 0
\]

solve the \( a_1 \) when \( \frac{\partial S}{\partial a_1} = 0 \) with the partial differentiation at \( a_1 \)
Multi-variable Regression

\[
\sum_{i=1}^{n} x_i \cdot y_i - \sum_{i=1}^{n} x_i \cdot a_0 - \sum_{i=1}^{n} a_1 x_i \cdot x_i - \sum_{i=1}^{n} a_2 x_i \cdot z_i = 0
\]

\[
\sum_{i=1}^{n} x_i \cdot y_i - \sum_{i=1}^{n} a_0 x_i - \sum_{i=1}^{n} a_1 x_i^2 - \sum_{i=1}^{n} a_2 x_i \cdot z_i = 0
\]

\[
\sum_{i=1}^{n} a_0 x_i + \sum_{i=1}^{n} a_1 x_i^2 + \sum_{i=1}^{n} a_2 x_i \cdot z_i = \sum_{i=1}^{n} x_i \cdot y_i
\]

\[
(\sum x_i) a_0 + (\sum x_i^2) a_1 + (\sum x_i \cdot z_i) a_2 = \sum x_i \cdot y_i \tag{2}
\]

equation (2) is derived for \(a_1\)
If you write down the equation ① and ② together, it is as follows:

\[ na_0 + \left( \sum x_i \right) a_1 + \left( \sum z_i \right) a_2 = \sum y_i \] ①

\[ \left( \sum x_i \right) a_0 + \left( \sum x_i^2 \right) a_1 + \left( \sum x_i \cdot z_i \right) a_2 = \sum x_i \cdot y_i \] ②
Multi-variable Regression

• Derive Multi-variable Regression with Partial Differentiation

\[
\frac{\partial S}{\partial a_2} = \frac{\partial}{\partial a_2} \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 z_i)^2 = 0
\]

\[
y = \{f(x)\}^n
\]

\[
y' = n \cdot (f(x))^{n-1} \cdot f'(x)
\]

\[
= 2 \cdot \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 z_i)^{2-1} \cdot \frac{\partial (y_i - a_0 - a_1 x_i - a_2 z_i)}{\partial a_2}
\]

\[
= -2z_i \cdot \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 z_i) = 0
\]

\[
= z_i \cdot \sum_{i=1}^{n} y_i - z_i \cdot \sum_{i=1}^{n} a_0 - z_i \cdot \sum_{i=1}^{n} a_1 x_i - z_i \cdot \sum_{i=1}^{n} a_2 z_i = 0
\]

\[
= \sum_{i=1}^{n} z_i \cdot y_i - \sum_{i=1}^{n} z_i \cdot a_0 - \sum_{i=1}^{n} a_1 z_i \cdot x_i - \sum_{i=1}^{n} a_2 z_i^2 = 0
\]
Multi-variable Regression (다변수 회귀)

\[
\sum_{i=1}^{n} z_i \cdot y_i - \sum_{i=1}^{n} z_i \cdot a_0 - \sum_{i=1}^{n} a_1 z_i \cdot x_i - \sum_{i=1}^{n} a_2 z_i^2 = 0
\]

\[
\sum_{i=1}^{n} z_i \cdot y_i - \sum_{i=1}^{n} a_0 z_i - \sum_{i=1}^{n} a_1 z_i \cdot x_i - \sum_{i=1}^{n} a_2 z_i^2 = 0
\]

\[
\sum_{i=1}^{n} a_0 z_i + \sum_{i=1}^{n} a_1 z_i \cdot x_i + \sum_{i=1}^{n} a_2 z_i^2 = \sum_{i=1}^{n} z_i \cdot y_i
\]

\[
\left( \sum z_i \right) a_0 + \left( \sum z_i \cdot x_i \right) a_1 + \left( \sum z_i^2 \right) a_2 = \sum_{i=1}^{n} z_i \cdot y_i \quad (3)
\]

Equation (3) is derived for \( a_2 \)
Multi-variable Regression

• For the unknown number $a_0$, $a_1$, $a_2$, the three equations ①, ②, and ③ can be calculated based on

$$\frac{\partial s}{\partial a_0} = 0, \frac{\partial s}{\partial a_1} = 0, \frac{\partial s}{\partial a_2} = 0.$$  

• where

$$S = \sum_{i=1}^{n}(y_i - a_0 - a_1 x_i - a_2 z_i)^2.$$  

– It is the sum of the errors of all actual and model values.

\[
\begin{align*}
na_0 + \left(\sum x_i\right)a_1 + \left(\sum z_i\right)a_2 &= \sum y_i \quad ① \\
\left(\sum x_i\right)a_0 + \left(\sum x_i^2\right)a_1 + \left(\sum x_i z_i\right)a_2 &= \sum x_i y_i \quad ② \\
\left(\sum z_i\right)a_0 + \left(\sum z_i x_i\right)a_1 + \left(\sum z_i^2\right)a_2 &= \sum_{i=1}^{n} z_i y_i \quad ③
\end{align*}
\]
Find Solution through Matrix Expansion

- The existing method of multiplying each of the ①, ②, and ③ equations to find the solution \((a_0, a_1, a_2)\) is very difficult. → Finding the solution through matrix expansion

\[
na_0 + \left( \sum x_i \right) a_1 + \left( \sum x_i^2 \right) a_2 = \sum y_i \quad ①
\]

\[
\left( \sum x_i \right) a_0 + \left( \sum x_i^2 \right) a_1 + \left( \sum x_i^3 \right) a_2 = \sum_{i=1}^{n} x_i \cdot y_i \quad ②
\]

\[
\left( \sum x_i^2 \right) a_0 + \left( \sum x_i^3 \right) a_1 + \left( \sum x_i^4 \right) a_2 = \sum_{i=1}^{n} x_i^2 \cdot y_i \quad ③
\]
Multivariable Linear Regression and TensorFlow

- Learn multiple weights ($w_1, w_2$) instead of one weight
- In GradientDecentOptimizer(), the process of obtaining a solution through matrix expansion is included. The abstraction is well done.

```python
import tensorflow as tf
#data
x1_data = [1,0,3,0,5]
x2_data = [0,2,0,4,0]
y_data = [1,2,3,4,5]
W1 = tf.Variable(tf.random_uniform([1],-1.0,1.0))
W2 = tf.Variable(tf.random_uniform([1],-1.0,1.0))
b = tf.Variable(tf.random_uniform([1],-1.0,1.0))
#hypothesis
hypothesis = W1 * x1_data + W2 * x2_data + b
cost = tf.reduce_mean(tf.square(hypothesis - y_data))
#minimize
a = tf.Variable(0.1) #alpha, learning rate
optimizer = tf.train.GradientDescentOptimizer(a)
train = optimizer.minimize(cost)
```
Multivariable Linear Regression and TensorFlow

- Learn multiple weights \((w_1, w_2)\) instead of one weight
- In GradientDecentOptimizer(), the process of obtaining a solution through matrix expansion is included. The abstraction is well done.

```python
# before starting, initialize variables
init = tf.initialize_all_variables()
# launch
sess = tf.Session()
sess.run(init)

# fit the line
for step in range(2001):
    sess.run(train)
    if step % 20 == 0:
        print (step, sess.run(cost), sess.run(W1), sess.run(W2), sess.run(b))
```
Multivariable linear regression with scikit-learn

Understand multivariable linear regression through scikit-learn
https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/regression_multivariable.py
Multivariable Linear Regression

Univariable linear regression is the process of creating a regression model for predicting dependent variables with only one independent variable.

Multivariate linear regression analysis is to create a regression model for predicting dependent variables with multiple independent variables.

- Independent variables of weight and age are used to predict blood fat.

https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/regression_multivariable.py
Multivariable Linear Regression

- These are weight, age, and blood fat content data.
import numpy as np
import matplotlib.pyplot as plt
import sklearn.linear_model
from mpl_toolkits.mplot3d import Axes3D

raw_data = np.genfromtxt('x09.txt', skip_header=36)
xs = np.array(raw_data[:,2], dtype=np.float32)
ys = np.array(raw_data[:,3], dtype=np.float32)
zs = np.array(raw_data[:,4], dtype=np.float32)

fig = plt.figure(figsize=(12,12))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(xs, ys, zs)
ax.set_xlabel('Weight')
ax.set_ylabel('Age')
ax.set_zlabel('Blood fat')
ax.view_init(15, 15)
plt.show()
Multivariable Linear Regression

- Understand multivariable linear regression through scikit-learn
- `fit()` method for multivariable linear regression.

```python
X = np.array(raw_data[:,2:4], dtype=np.float64)
# 두 개의 변수가 사용된다
# ax.set_xlabel('Weight')
# ax.set_ylabel('Age')
y = np.array(raw_data[:,4], dtype=np.float64)
model = sklearn.linear_model.LinearRegression()
model.fit(X, y)
print(model)
print('Est [100,40] : ', model.predict([[100,40]]))
print('Est [60,25] : ', model.predict([[60,25]]))
knn = sklearn.neighbors.KNeighborsRegressor(n_neighbors=3)
knn.fit(X, y)
print(knn)
print('Est [100,40] : ', knn.predict([[100,40]]))
Console Debugging : np.genfromtxt() method

- raw_data = np.genfromtxt('x09.txt', skip_header=36)
- np.genfromtxt() method:
  - Import data sets of weight (Weight, kilograms), age (Age, Years), blood fat content as raw_data

```
array([[ 1.,  1.,  84.,  46., 354.],
       [ 2.,  1.,  73.,  20., 190.],
       [ 3.,  1.,  65.,  52., 405.],
       [ 4.,  1.,  70.,  30., 263.],
       [ 5.,  1.,  76.,  57., 451.],
       [ 6.,  1.,  69.,  25., 302.],
       [ 7.,  1.,  63.,  28., 288.],
       [ 8.,  1.,  72.,  36., 385.],
       [ 9.,  1.,  79.,  57., 402.],
```

Console debugging: check the contents of variables xs, ys, zs

- **Store weight (kilograms) in variable xs.**
  - $xs = \text{np.array}(\text{raw_data}[:,2], \text{dtype} = \text{np.float32})$

- **Store age (Age, Years) in variable ys**
  - $ys = \text{np.array}(\text{raw_data}[:,3], \text{dtype} = \text{np.float32})$

- **Stored in the blood fat content variable zs**
  - $zs = \text{np.array}(\text{raw_data}[:,4], \text{dtype} = \text{np.float32})$
Console Debugging: Drawing a 3-D Graph

- Specify the figure size with `fig = plt.figure(figsize=(12,12))`.
- Let `ax = fig.add_subplot(111, projection='3d')` draw a 3D graph.
Console Debugging: Drawing a 3-D Graph

- Use `ax.scatter(xs, ys, zs)` function to scatter weight (variable `xs`), age (variable `ys`), and blood fat (variable `zs`).
Console debugging: creating a predictive model

• From the 3D graph, we can see that we are trying to predict blood fat (variable zs) according to body weight (variable xs) and age (variable ys).

• $X = \text{np.array}(\text{raw\_data}[:,:2:4], \text{dtype}=\text{np.float64})$
  – Two variables are used for $X$: weight (variable xs) and age (variable ys).
  – array([[84., 46.],
  [73., 20.],
  [65., 52.],
  [70., 30.]])

• $y = \text{np.array}(\text{raw\_data}[:,4], \text{dtype}=\text{np.float64})$
  – Blood fat (variable zs), variable is used for $y$.
  274., 303., 244.])]
You create a linear regression object through sklearn's `LinearRegression()` method.

- `model = sklearn.linear_model.LinearRegression()`
- `LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)`

Linear regression is performed through the `model.fit(X, y)` method.
Prediction

- Weight (variable xs), 100kg, age (variable ys), and blood fat (variable zs) of a 40-year-old person are predicted.
- The 100kg weight is currently not shown in the data set.
- Age 40 is shown.
Console debugging: prediction via `model.predict()` method

- Weight (variable `xs`), 100kg, age (variable `ys`), and blood fat (variable `zs`) of a 40-year-old person are predicted
  - The prediction result is 328.38.
  - `print('Est [100,40] : ', model.predict([[100,40]]))`.
  - `Est [100,40] : [328.38238085]`

- Weight (variable `xs`), 60kg, age (variable `ys`), and blood fat (variable `zs`) of a 25-year-old person are predicted
  - The prediction result is 233.43
  - `Est [60,25] : [233.43903476]`
Draw the prediction model line

- https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/regression_multivariable_graph.py
Thank you for your listening

If you have any questions, please send e-mail via sckim7@kookmin.ac.kr