



Deep Learning

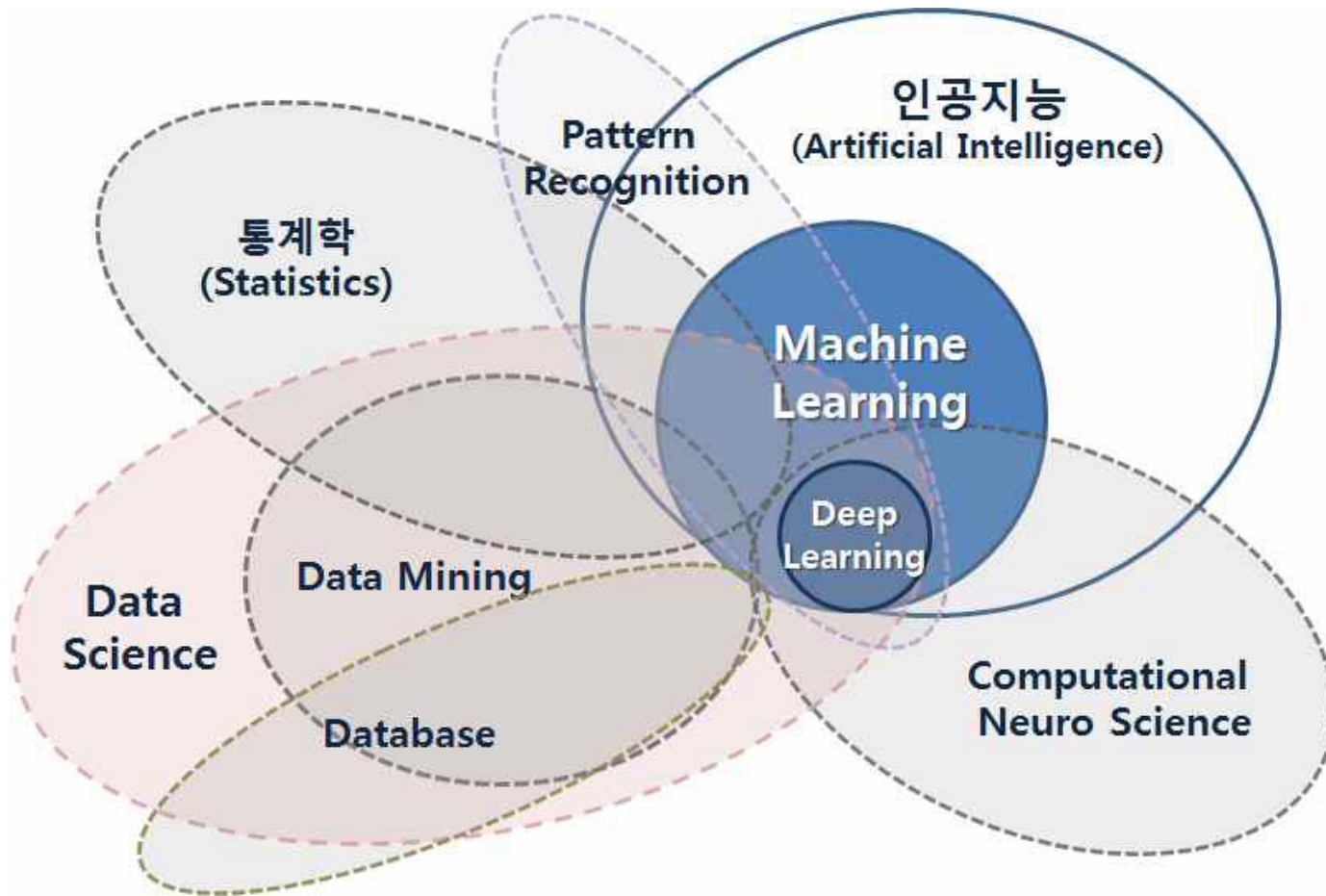
강사 : 구정은

1일차

0. 인공지능과 머신러닝 그리고 딥러닝

머신러닝

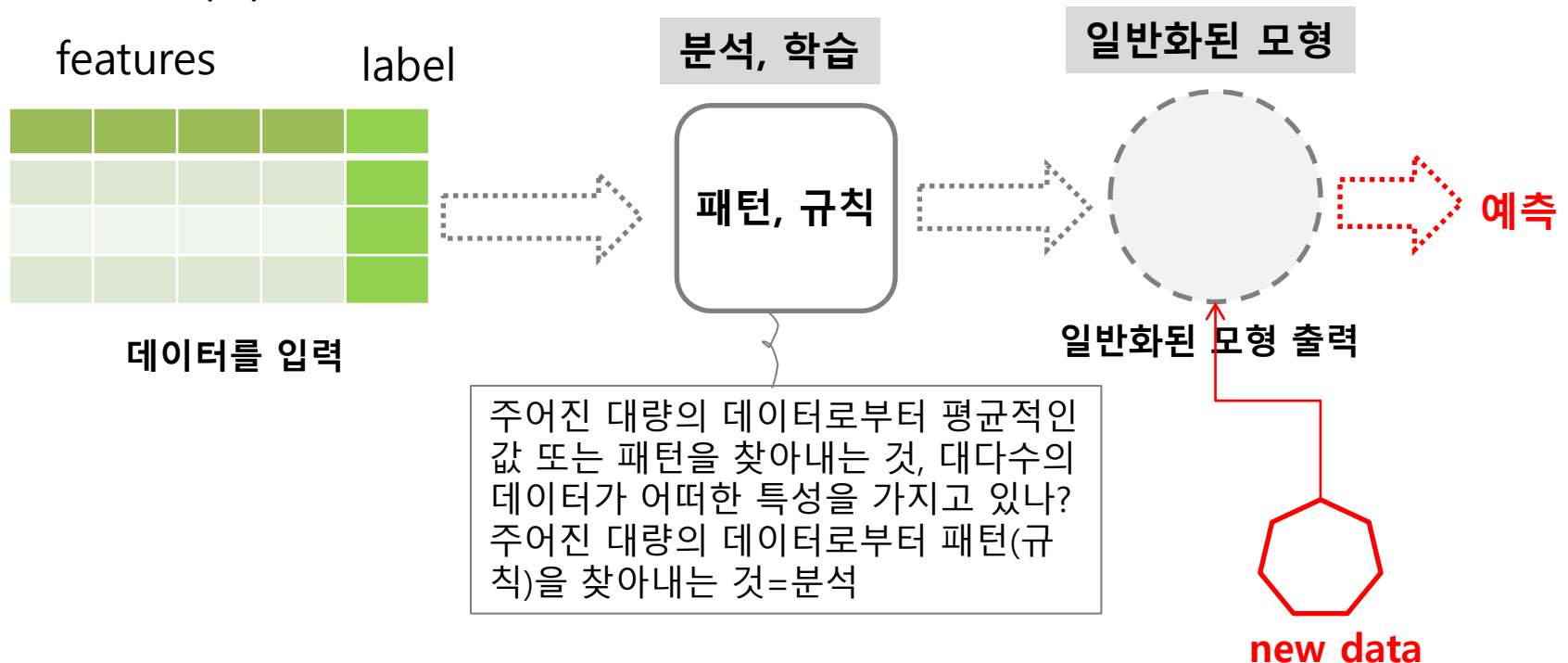
- 인공지능과 머신러닝



머신러닝

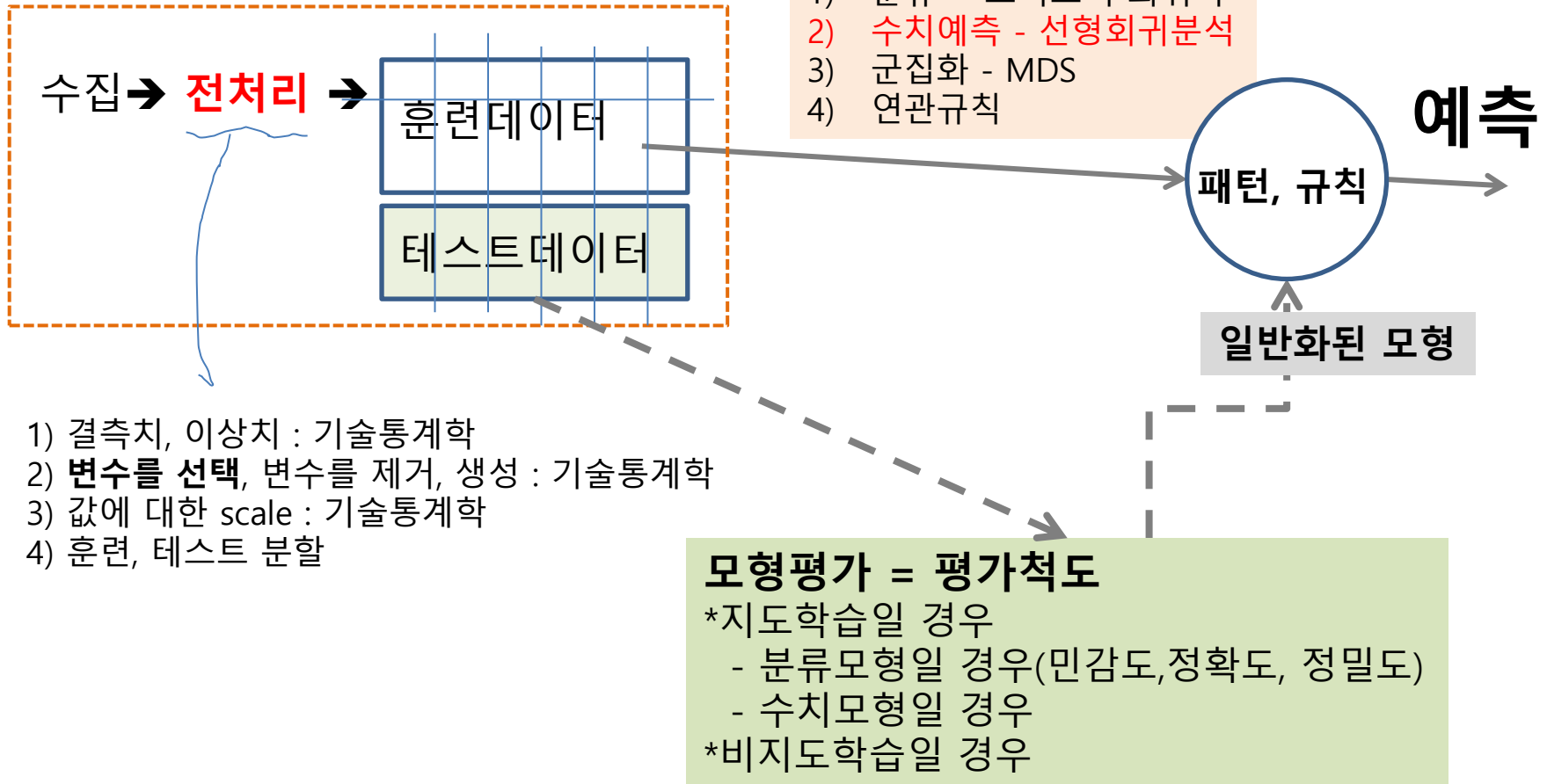
머신러닝의 개념

- 머신러닝(Machine Learning)은 컴퓨터 과학의 영역에 속하는 인공지능의 한 분야로서, 컴퓨터 프로그램이 어떤 것에 대한 학습을 통해 기존의 모델이나 결과물을 개선하거나 (자동으로) 예측 하게끔 구축하는 과정을 의미



머신러닝 & 알고리즘

머신러닝



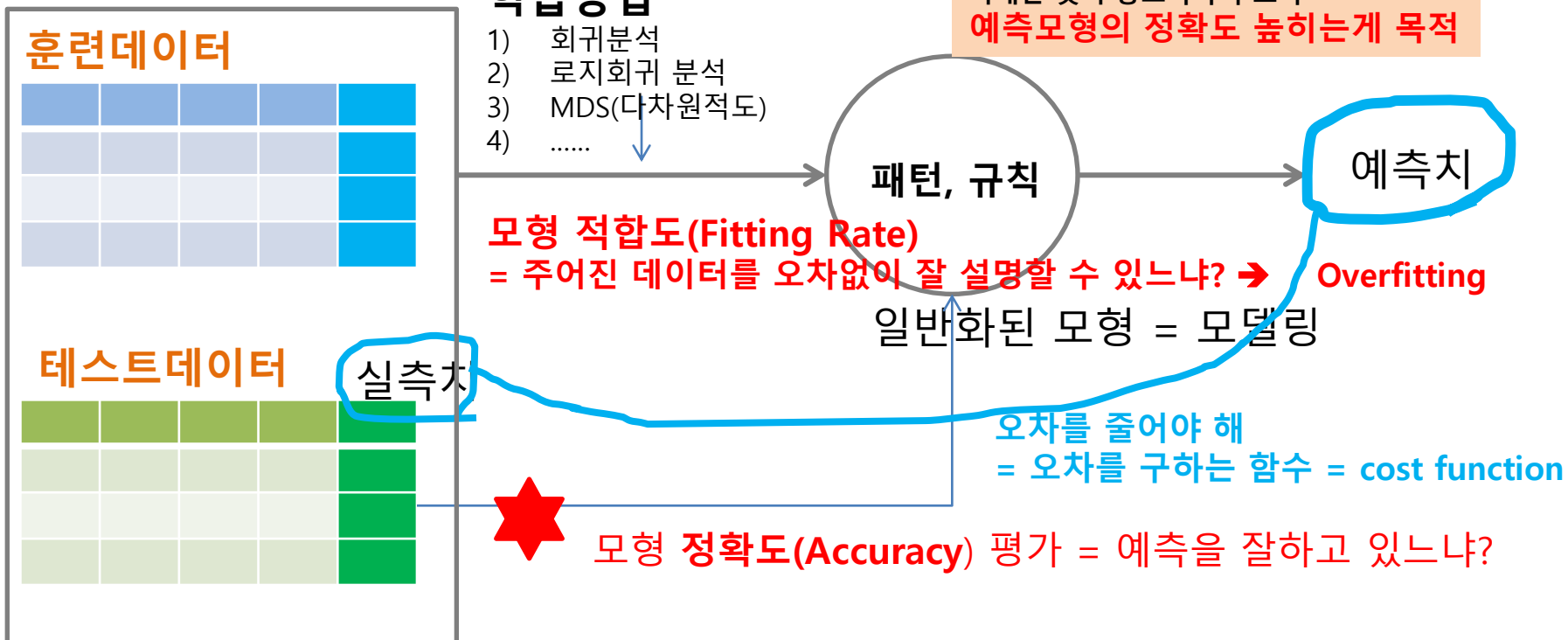
학습모형 평가

• 학습모형의 평가

대량의 주어진 데이터의 패턴, 규칙 찾아(fitting=학습)내어 → **예측**

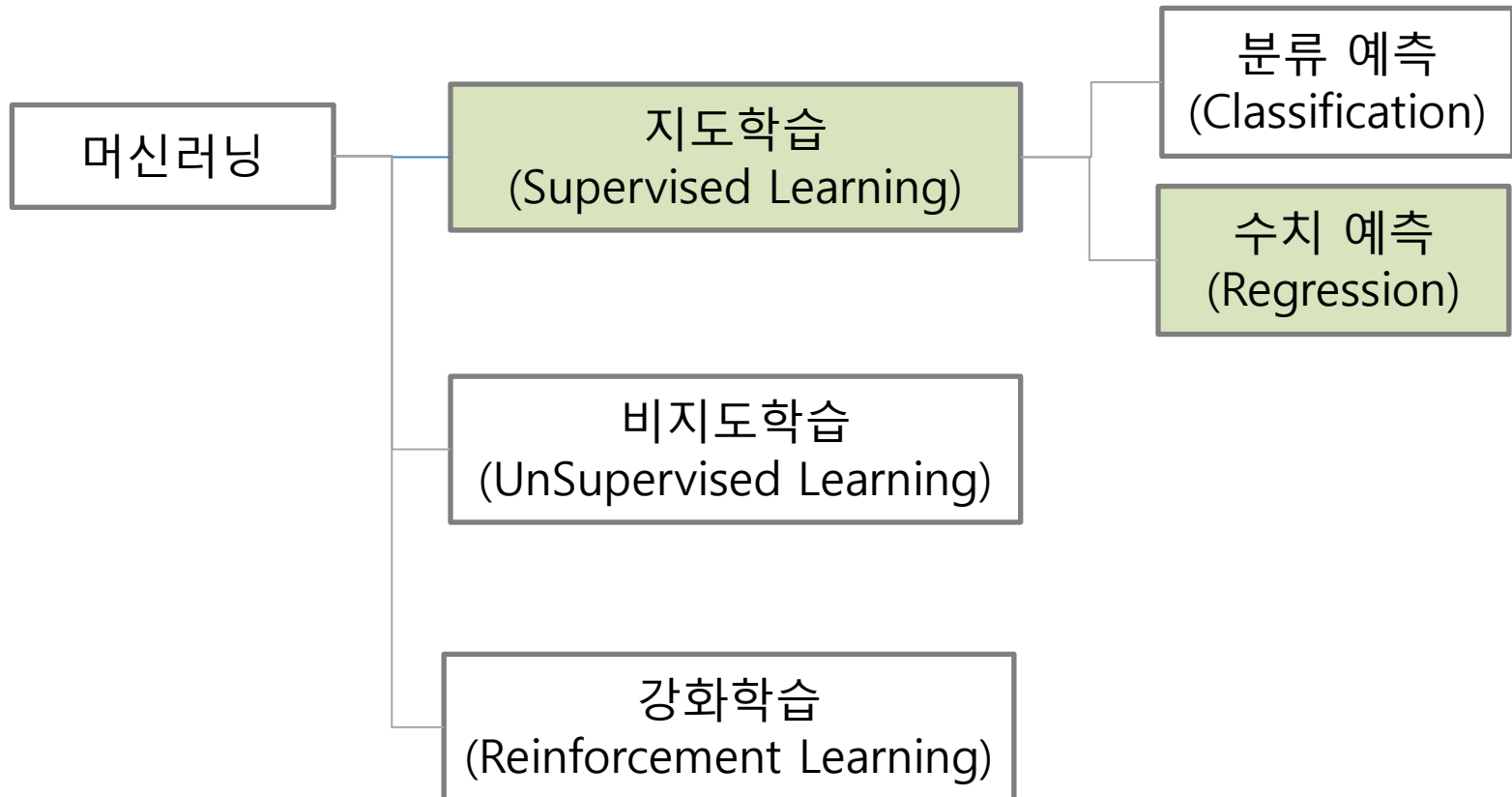
대략적으로 ~하다=평균 = **일반화**

• 지도 학습과정



머신러닝 학습방법의 종류

- 머신러닝 학습 방법



Framework for ML & Deep Learning

- DeepLearning Framework



(google)

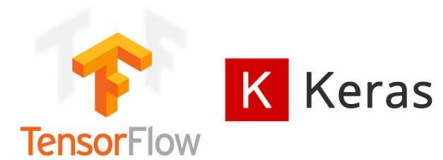
- Debugging 어려움
- 분산병렬처리에 우수
- 코드가 복잡



(facebook AI Rearch)

- Debugging 수월
- 코드간결
- 명령형 프로그래밍과 유사

Low-level API



TF v 2.0 + Keras

- 코드가 간결
- 가독성이 우수, 초보자에 적합

High-level API (python)

1일차

1-1 Tensorflow 개요

1-2 선형회귀분석



TensorFlow

Tensorflow

- Tensorflow 설치

1. Tensorflow 설치

- Window에 Tensorflow 설치(2016.11월 이후 가능)

- Tensorflow 공식 설치 가이드

https://www.tensorflow.org/get_started/os_setup

- CPU-only version of TensorFlow

command prompt:> pip install --upgrade pip

command prompt:> pip install tensorflow=버전

- GPU version of TensorFlow

command prompt:> pip install tensorflow-gpu

conda 패키지 관리자를 이용하는 경우

➤ **conda install tensorflow**

➤ **conda update -all** #파이썬 최신 패키지로 업데이트

로 설치 가능하지만 아나콘다 배포판에는 텐서플로 최신 버전이 늦게 포함되므로
파이썬 기본 패키지 관리자인 pip로 텐서플로를 설치 권장

1. Tensorflow 설치

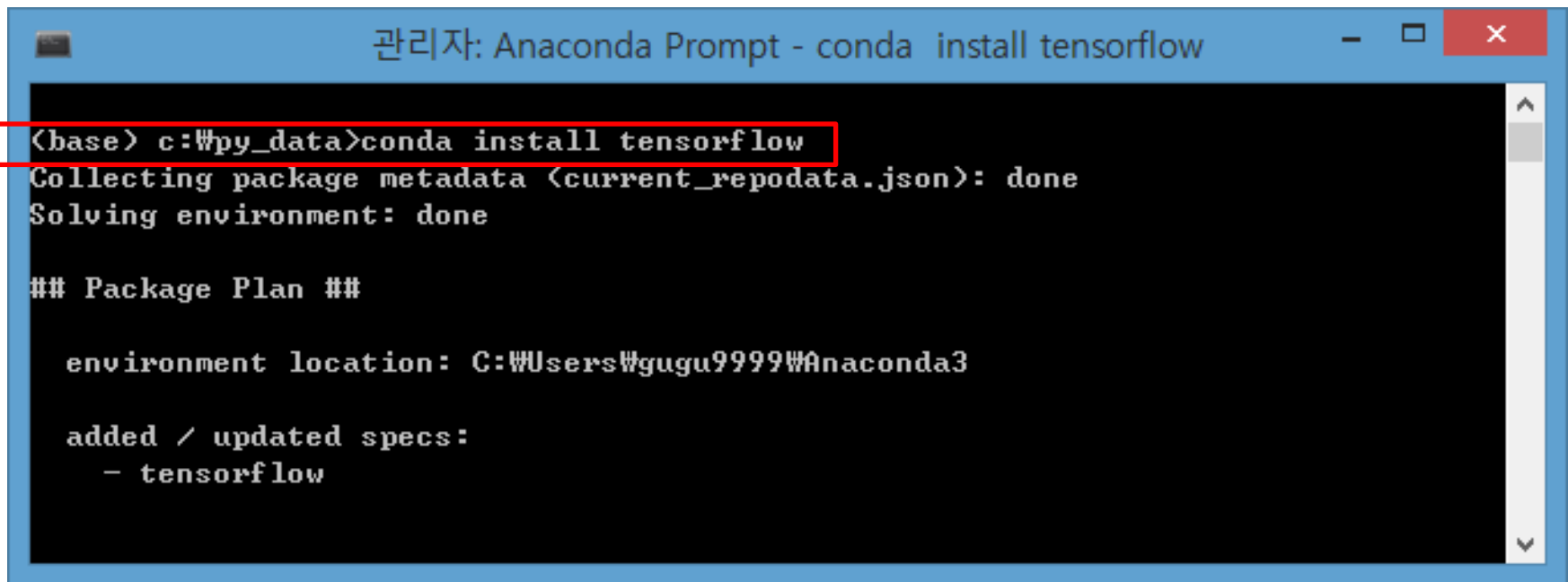
- 텐서플로 1.9.0 버전부터는 콘다를 사용하여 텐서플로를 설치하는 것이 권장됩니다. MKL-DNN 라이브러리에 최적화되어 있어서 CPU만을 사용하는 경우 보다 나은 성능을 기대할 수 있습니다.(콘다 패키지는 최신 텐서플로를 지원하지 않을 수 있습니다)

```
> conda install tensorflow
```

1. Tensorflow 설치

- CPU Only 버전으로 설치

① Anaconda Prompt를 관리자 권한으로 실행합니다.



```
관리자: Anaconda Prompt - conda install tensorflow

(base) c:\wpy_data>conda install tensorflow
Collecting package metadata (current_repodata.json): done
Solving environment: done

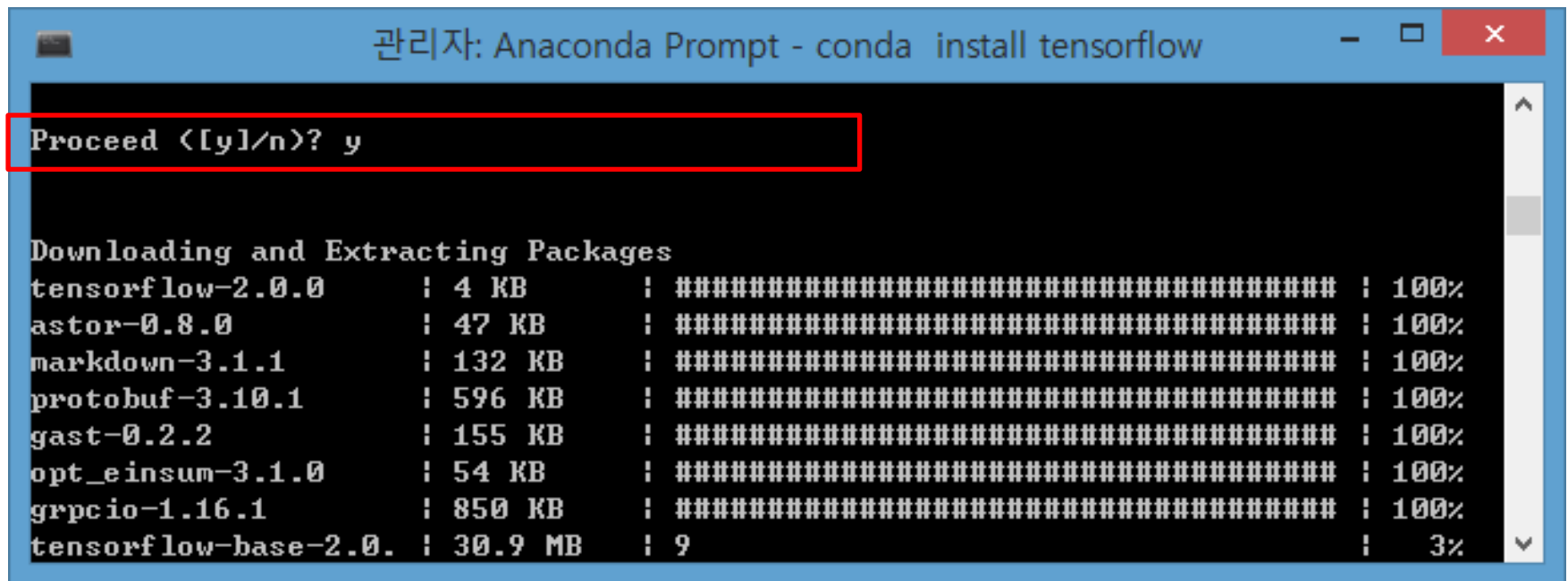
## Package Plan ##

  environment location: C:\Users\Wgugu9999\Anaconda3

added / updated specs:
- tensorflow
```

1. Tensorflow 설치

- CPU Only 버전으로 설치
- ② 설치를 묻는 메시지에 y로 대답합니다.



The screenshot shows a terminal window titled "관리자: Anaconda Prompt - conda install tensorflow". The prompt "Proceed <[y]/n>? y" is highlighted with a red box, indicating the user's response to the installation confirmation. Below the prompt, the progress of downloading and extracting packages is shown, including tensorflow-2.0.0, astor-0.8.0, markdown-3.1.1, protobuf-3.10.1, gast-0.2.2, opt_einsum-3.1.0, grpcio-1.16.1, and tensorflow-base-2.0.0.

```
관리자: Anaconda Prompt - conda install tensorflow

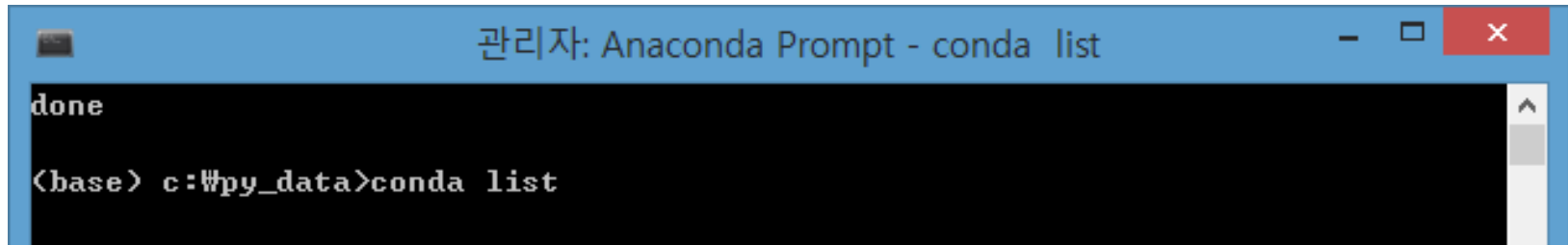
Proceed <[y]/n>? y

Downloading and Extracting Packages
tensorflow-2.0.0           | 4 KB           | ##### | 100%
astor-0.8.0               | 47 KB          | ##### | 100%
markdown-3.1.1           | 132 KB         | ##### | 100%
protobuf-3.10.1          | 596 KB         | ##### | 100%
gast-0.2.2               | 155 KB         | ##### | 100%
opt_einsum-3.1.0         | 54 KB          | ##### | 100%
grpcio-1.16.1            | 850 KB         | ##### | 100%
tensorflow-base-2.0.0    | 30.9 MB        | 9      | 3%
```

1. Tensorflow 설치

- CPU Only 버전으로 설치
- ② 설치를 묻는 메시지에 y로 대답합니다.

<설치 완료 후>

A screenshot of an Anaconda Prompt window. The title bar is blue and contains the text '관리자: Anaconda Prompt - conda list'. The window has standard Windows window controls (minimize, maximize, close) on the right. The main area is black with white text. It shows the word 'done' on the first line. On the second line, the command prompt shows '(base) c:\wpy_data>conda list'.

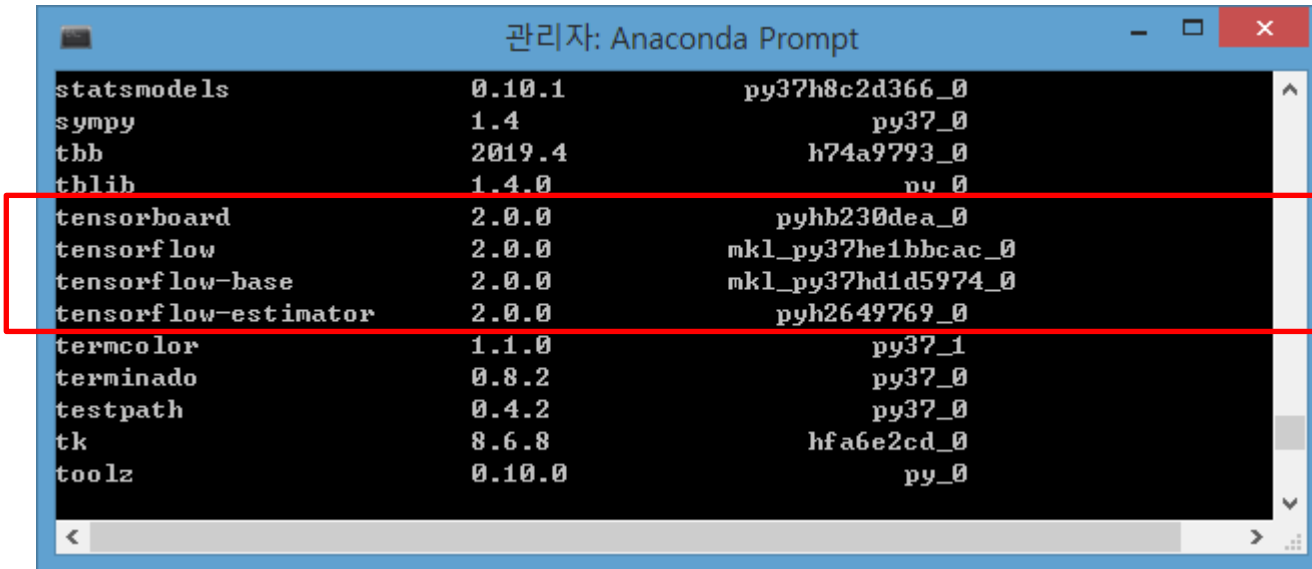
```
done  
(base) c:\wpy_data>conda list
```

2019.11.22

1. Tensorflow 설치

- CPU Only 버전으로 설치
- ② 설치를 묻는 메시지에 y로 대답합니다.

<설치 완료 후>



```
관리자: Anaconda Prompt
statsmodels      0.10.1      py37h8c2d366_0
sympy            1.4        py37_0
tbb              2019.4     h74a9793_0
tblib            1.4.0      nv_0
tensorboard      2.0.0      pyhb230dea_0
tensorflow       2.0.0      mkl_py37he1bbcac_0
tensorflow-base  2.0.0      mkl_py37hd1d5974_0
tensorflow-estimator 2.0.0      pyh2649769_0
termcolor       1.1.0      py37_1
terminado        0.8.2      py37_0
testpath        0.4.2      py37_0
tk               8.6.8      hf a6e2cd_0
toolz            0.10.0     py_0
```

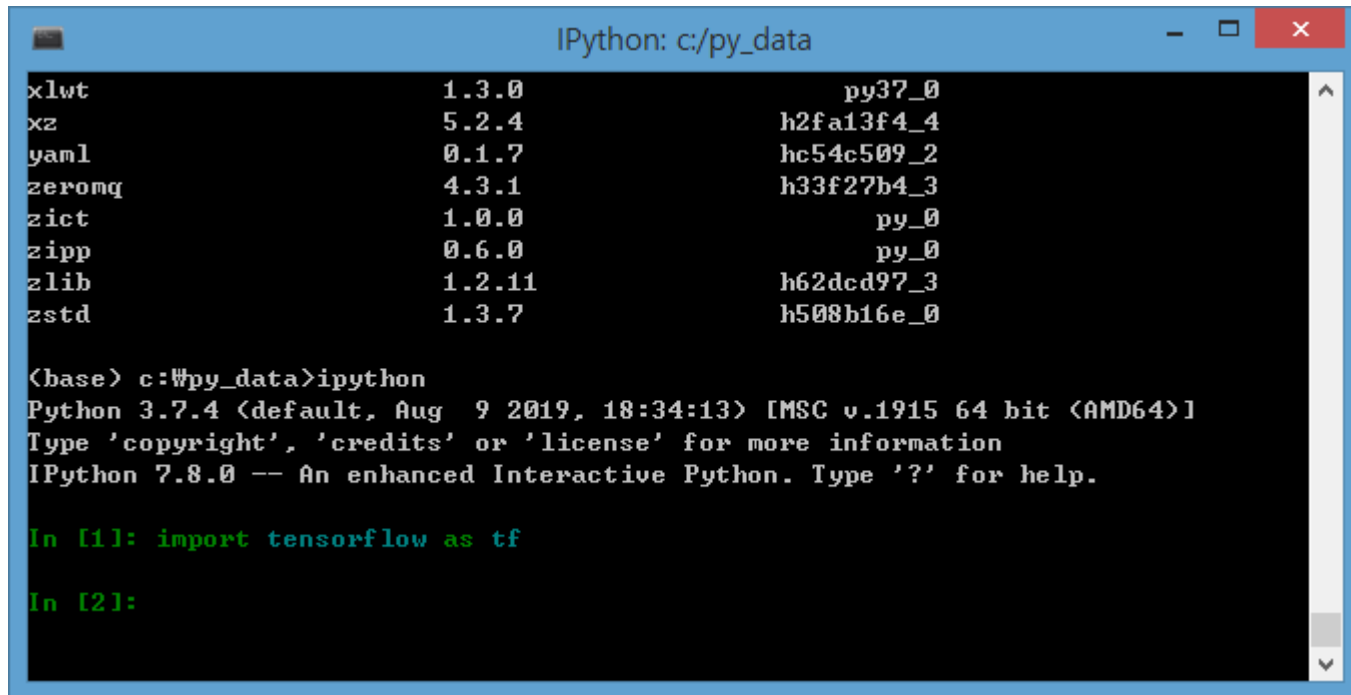
2019.11.22

1. Tensorflow 설치

- CPU Only 버전으로 설치

[Tensorflow 설치]

- ③ 설치가 완료된 후 IPython 쉘을 실행하여 tensorflow 모듈을 임포트합니다.
아무런 메시지가 뜨지 않으면 정상적으로 설치에 성공한 것입니다.



```
IPython: c:/py_data

xlwt                1.3.0                py37_0
xz                  5.2.4                h2fa13f4_4
yaml                0.1.7                hc54c509_2
zeromq              4.3.1                h33f27b4_3
zict                1.0.0                py_0
zipp                0.6.0                py_0
zlib                1.2.11               h62dcd97_3
zstd                1.3.7                h508b16e_0

(base) c:\py_data>ipython
Python 3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.8.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import tensorflow as tf

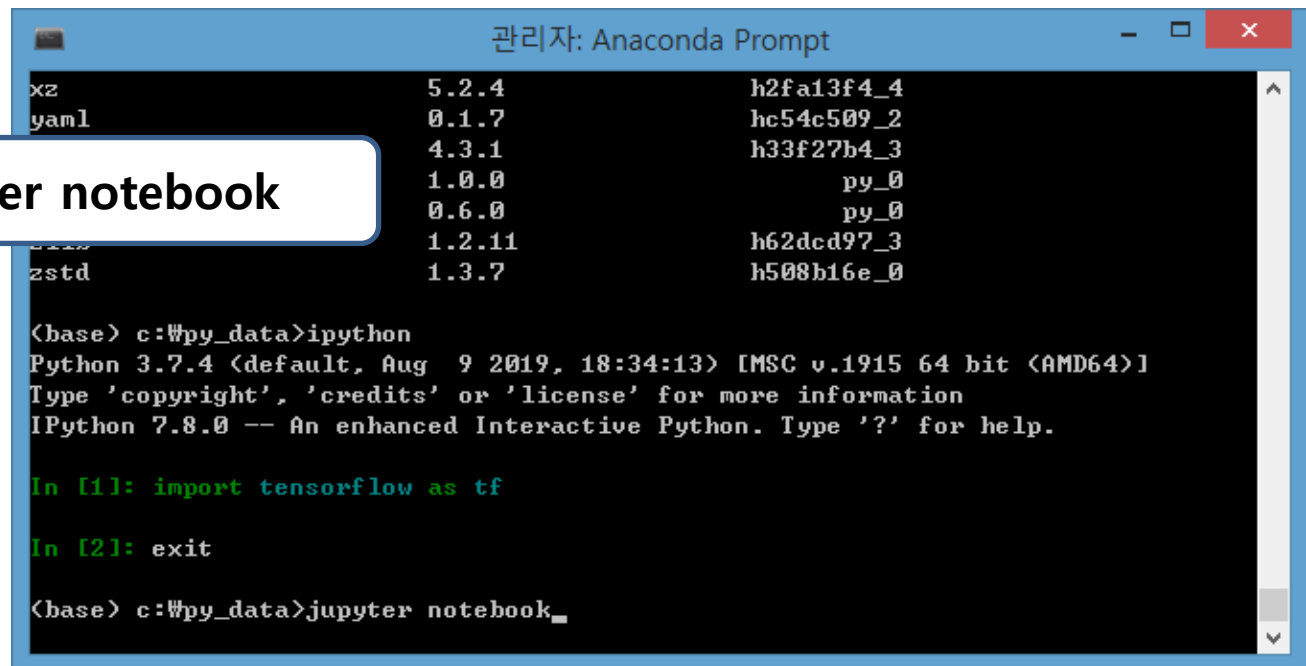
In [2]:
```

1. Tensorflow 설치

- CPU Only 버전으로 설치

④ IPython 셸을 종료하려면 `exit` 명령을 입력합니다. 데이터 분석을 위해 IPython 셸도 좋지만 이보다 코드와 실행 결과를 함께 관리할 수 있는 jupyter notebook을 사용하도록 하겠습니다. 주피터 노트북은 로컬 컴퓨터에서 실행되는 웹 서버 프로그램과 비슷합니다. 브라우저로 코드를 실행하면 IPython 커널에게 실행을 명령하고 그 결과를 브라우저로 전달해 줍니다. 주피터 노트북을 실행하려면 아나콘다 프롬프트 `py35` 환경에서 jupyter notebook 명령을 사용합니다.

(base)...>jupyter notebook



```
관리자: Anaconda Prompt

xz                5.2.4                h2fa13f4_4
yaml              0.1.7                hc54c509_2
                  4.3.1                h33f27b4_3
                  1.0.0                py_0
                  0.6.0                py_0
                  1.2.11               h62dcd97_3
zstd              1.3.7                h508b16e_0

(base) c:\wpy_data>ipython
Python 3.7.4 <default, Aug  9 2019, 18:34:13> [MSC v.1915 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.8.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import tensorflow as tf

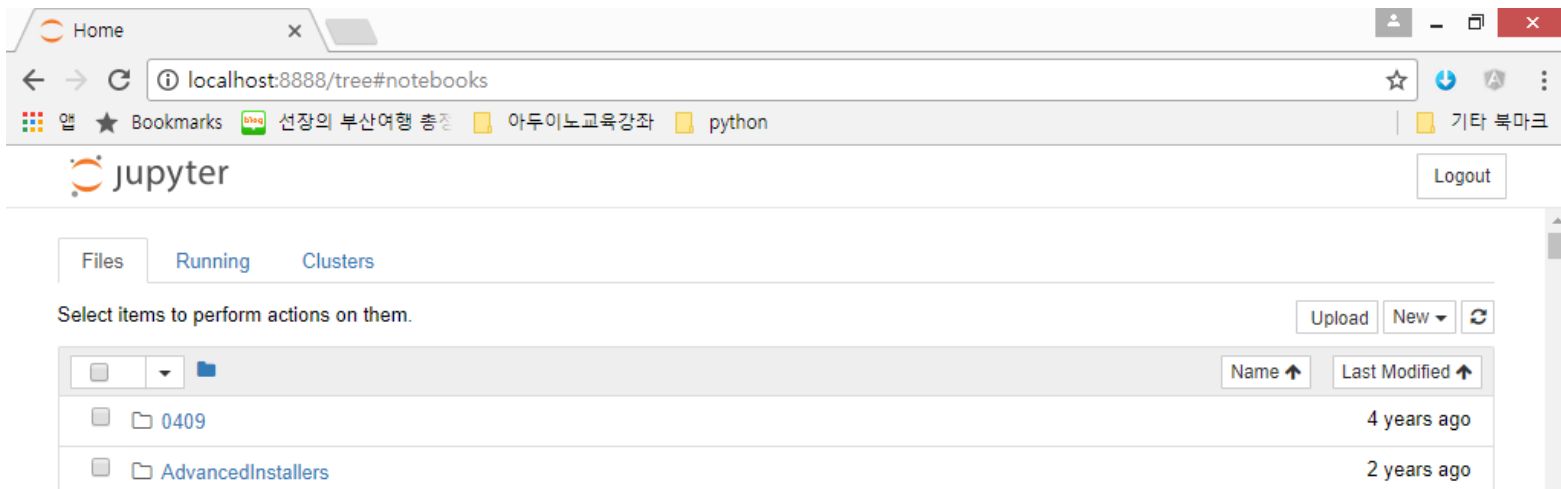
In [2]: exit

(base) c:\wpy_data>jupyter notebook_
```

1. Tensorflow설치

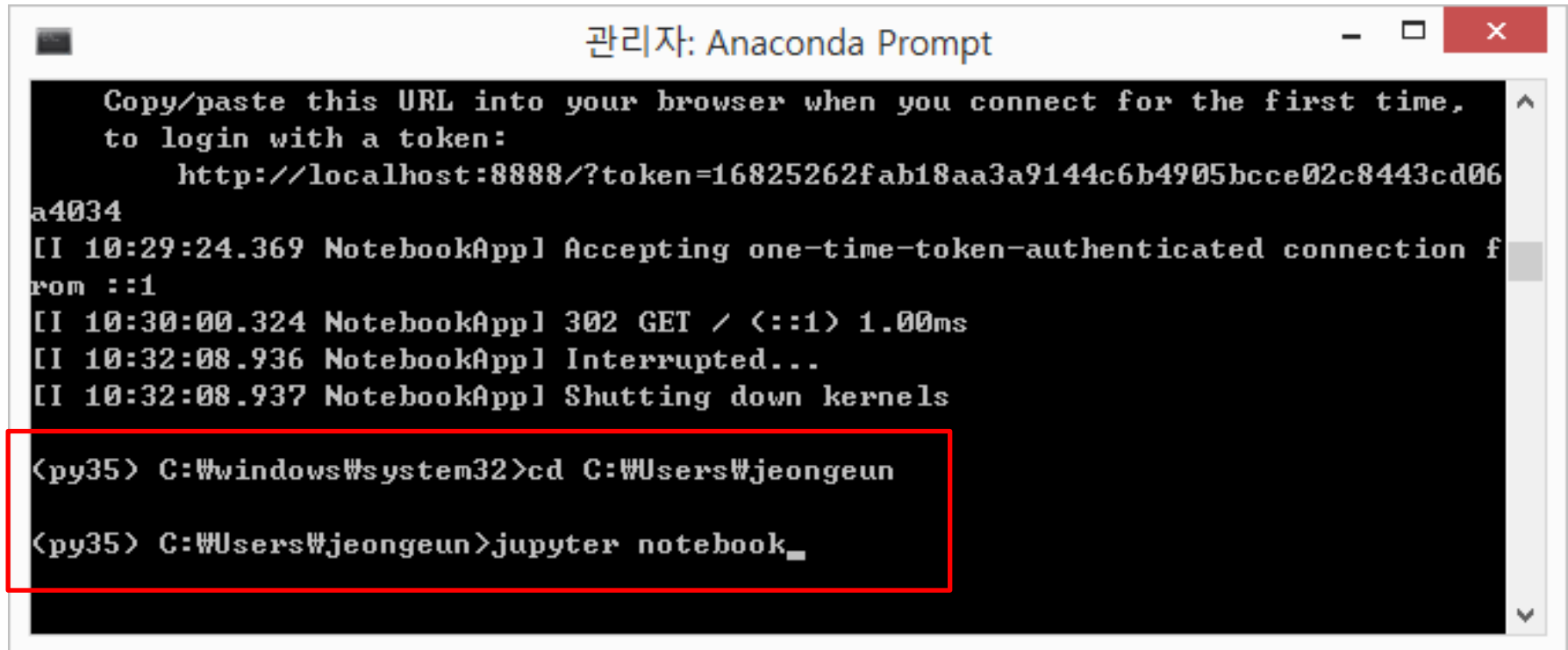
- CPU Only 버전으로 설치

⑤ 주피터 노트북이 실행되면 자동으로 기본 브라우저가 실행되어 주피터 노트북 서버에 접속합니다. 로컬 컴퓨터의 주피터 노트북 서버 주소는 <http://localhost:8888/> 입니다. 주피터 노트북을 실행한 현재 폴더를 기본 홈 페이지로 설정됩니다. 이 폴더 하위에 파이썬 주피터 노트북을 만들고 실행할 수 있습니다.



1. Tensorflow 설치

- CPU Only 버전으로 설치
- [참고] 홈 경로 변경하여 실행하기



```
관리자: Anaconda Prompt

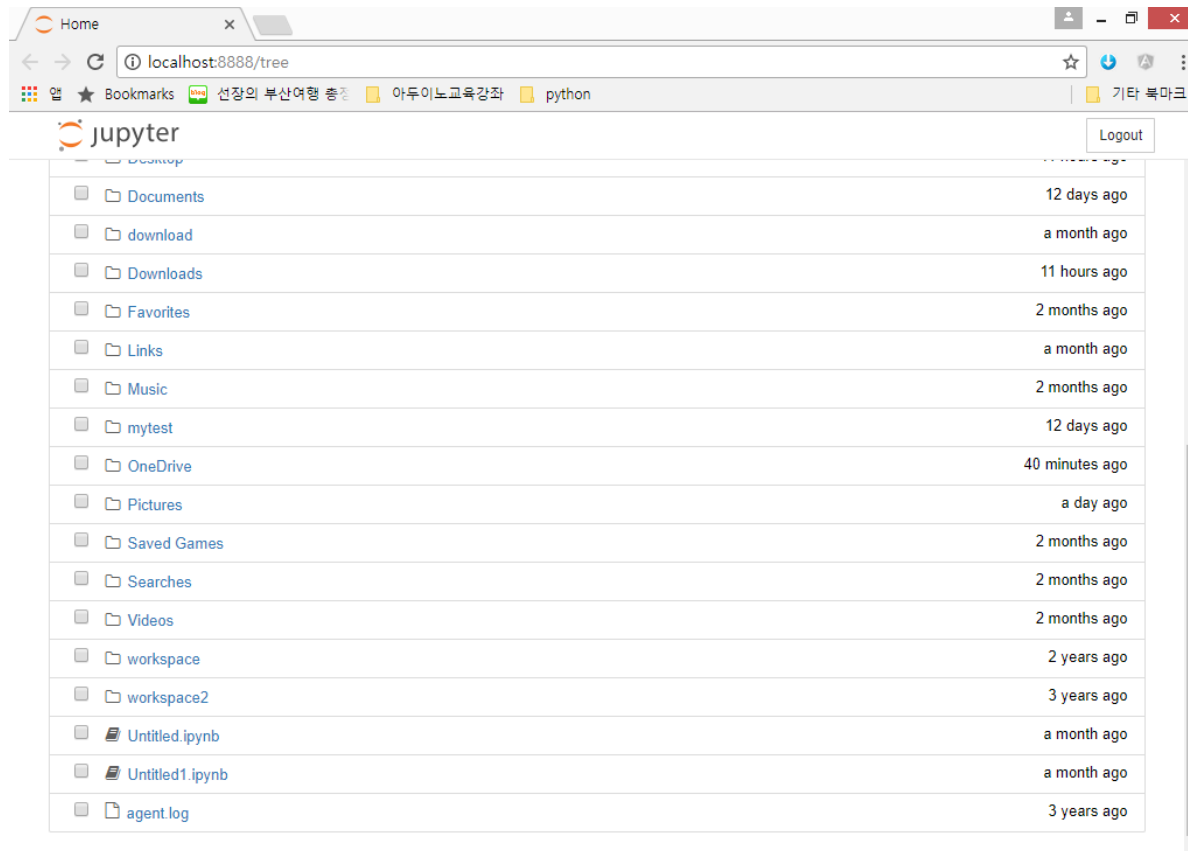
Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
    http://localhost:8888/?token=16825262fab18aa3a9144c6b4905bcce02c8443cd06
a4034
[I 10:29:24.369 NotebookApp] Accepting one-time-token-authenticated connection f
rom ::1
[I 10:30:00.324 NotebookApp] 302 GET / (:::1) 1.00ms
[I 10:32:08.936 NotebookApp] Interrupted...
[I 10:32:08.937 NotebookApp] Shutting down kernels

<py35> C:\windows\system32>cd C:\Users\jeongeun

<py35> C:\Users\jeongeun>jupyter notebook_
```

1. Tensorflow 설치

- CPU Only 버전으로 설치
[참고] 홈 경로 변경하여 실행하기

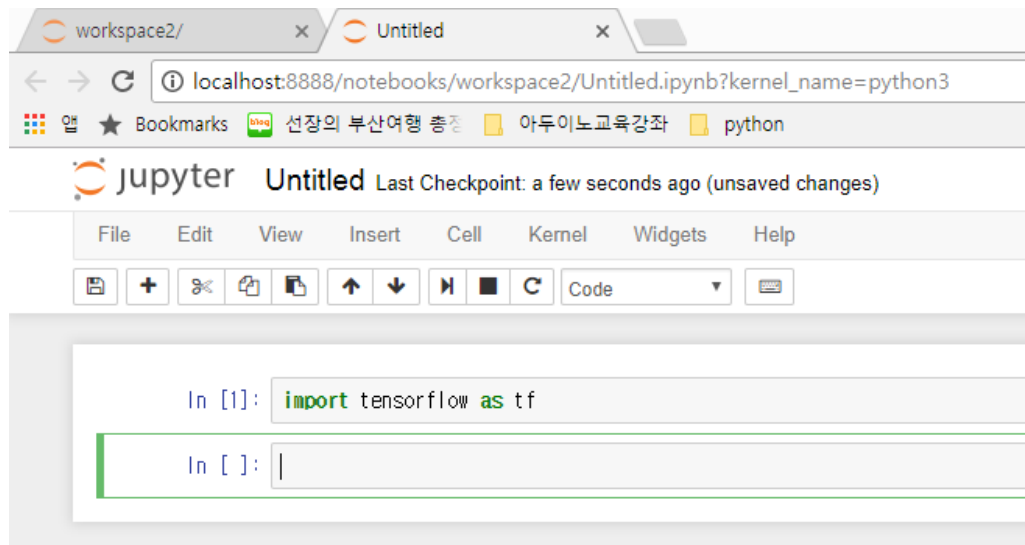


1. Tensorflow 설치

- CPU Only 버전으로 설치

⑥ 새로운 파이썬 노트북을 만들어 보겠습니다. 오른쪽 위에 있는 New 버튼을 누르면 새로운 파이썬 3 주피터 노트북을 생성할 수 있습니다.

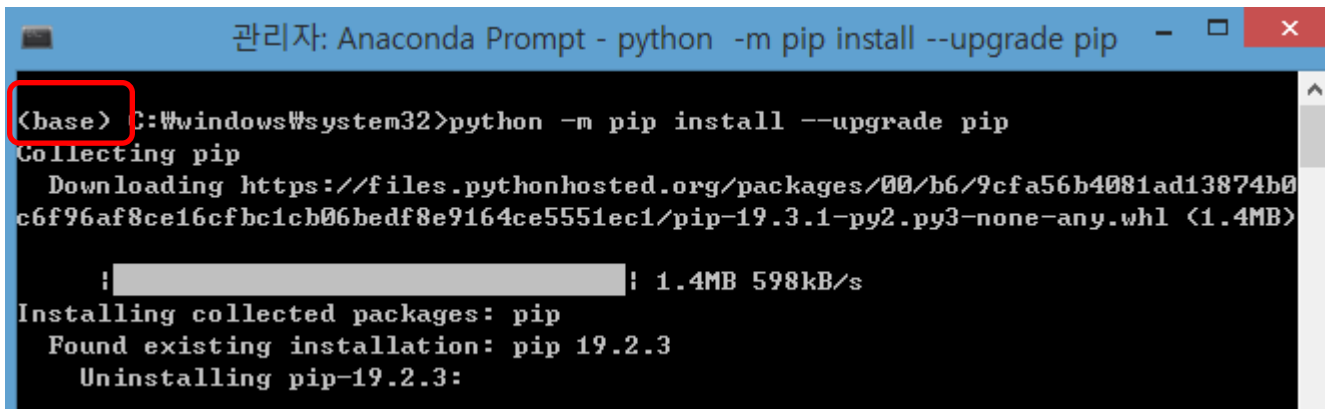
⑦ 새로운 브라우저 탭이 열리면서 Untitled 노트북이 생성됩니다. 첫번째 코드 셀(cell)에 IPython 쉘에서 했던 것처럼 **"import tensorflow as tf"**를 입력하고 *Shift+엔터* 키를 입력합니다. 아무런 메시지가 나오지 않으면 텐서플로를 주피터 노트북에서 사용할 수 있도록 설치에 성공한 것입니다.



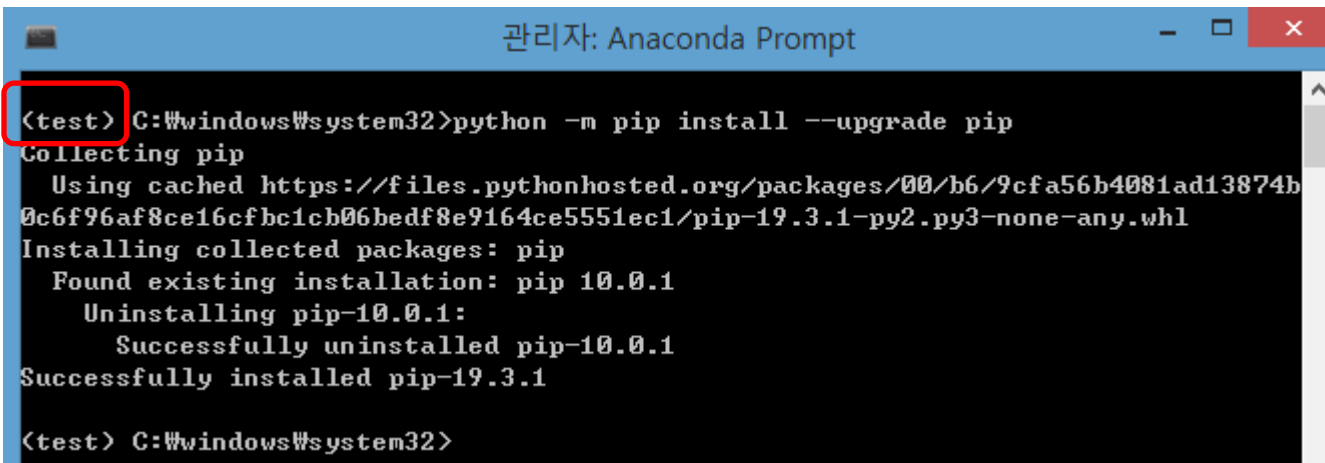
2. Conda의 가상환경 만들기

- 가상환경만들기

(test) > python -m pip install --upgrade pip



```
관리자: Anaconda Prompt - python -m pip install --upgrade pip
<base> C:\Windows\system32>python -m pip install --upgrade pip
Collecting pip
  Downloading https://files.pythonhosted.org/packages/00/b6/9cfa56b4081ad13874b0
c6f96af8ce16cfbc1cb06bedf8e9164ce5551ec1/pip-19.3.1-py2.py3-none-any.whl (1.4MB)
    |#####| 1.4MB 598kB/s
Installing collected packages: pip
  Found existing installation: pip 19.2.3
  Uninstalling pip-19.2.3:
```



```
관리자: Anaconda Prompt
<test> C:\Windows\system32>python -m pip install --upgrade pip
Collecting pip
  Using cached https://files.pythonhosted.org/packages/00/b6/9cfa56b4081ad13874b
0c6f96af8ce16cfbc1cb06bedf8e9164ce5551ec1/pip-19.3.1-py2.py3-none-any.whl
Installing collected packages: pip
  Found existing installation: pip 10.0.1
  Uninstalling pip-10.0.1:
    Successfully uninstalled pip-10.0.1
  Successfully installed pip-19.3.1
<test> C:\Windows\system32>
```

2. Conda의 가상환경 만들기

- 가상환경만들기

```
관리자: Anaconda Prompt - conda update conda

(base) c:\wp_data>conda --version
conda 4.7.12

(base) c:\wp_data>conda update conda
Collecting package metadata (current_repodata.json): done
Solving environment: /
```

```
1 #아나콘다 버전 확인
2 conda --version
3
4 #아나콘다 업데이트
5 conda update conda
```

```
관리자: Anaconda Prompt

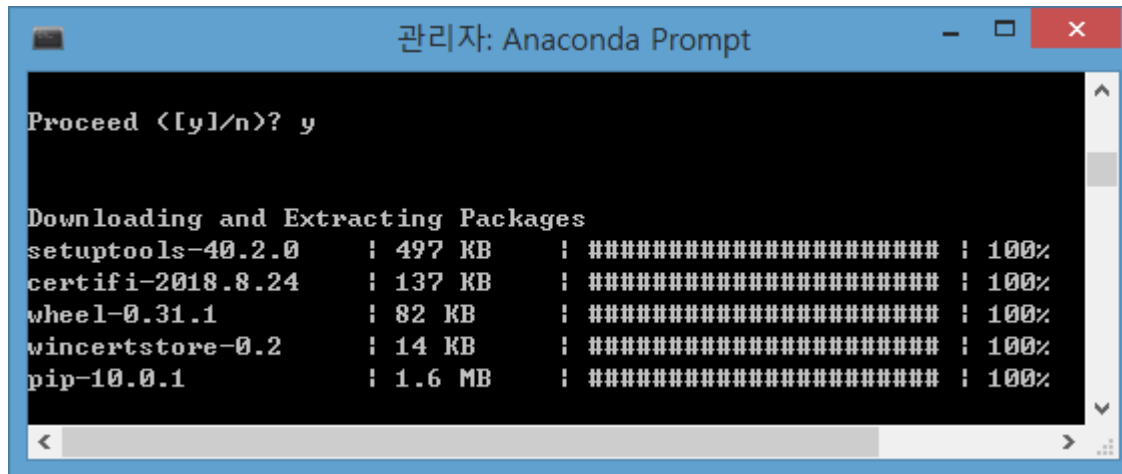
(base) c:\wp_data>conda --version
conda 4.7.12

(base) c:\wp_data>conda create --name test python=3.5
Collecting package metadata (current_repodata.json): done
Solving environment: failed with repodata from
with next repodata source.
Collecting package metadata (repodata.json): d
Solving environment: done
```

```
1 #아나콘다 가상환경 생성
2 conda create --name(-n) 가상환경명 설치할패키지
3
4 #예) 파이썬 3.5 버전 설치 & test 이름으로 가상환경 생성
5 conda create --name test python=3.5
6
7 #또는
8 conda create --n test python=3.5
```

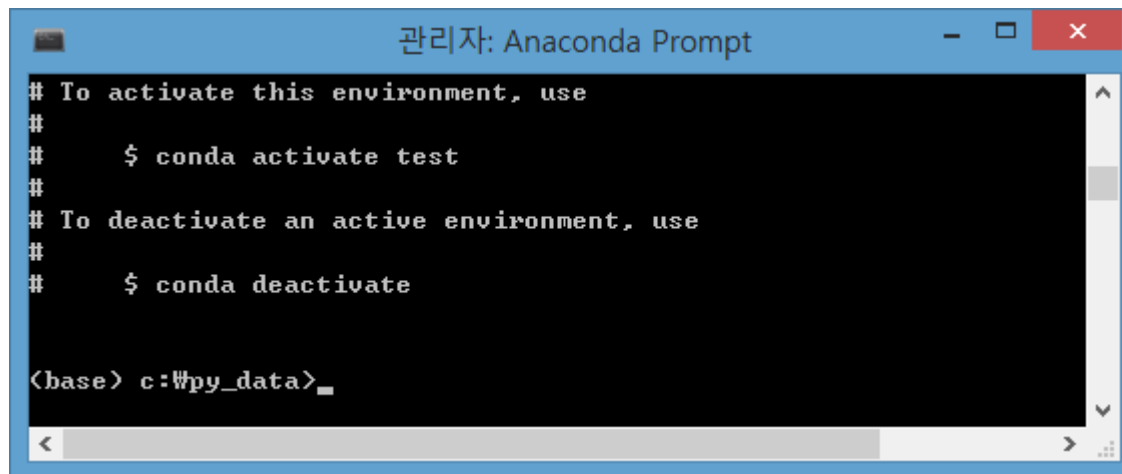

3. 가상환경 설정

- 가상환경만들기



A screenshot of an Anaconda Prompt window titled "관리자: Anaconda Prompt". The terminal shows the command "Proceed ([y]/n)? y" being entered. Below it, a section titled "Downloading and Extracting Packages" displays a progress table for several packages. Each package is shown with its name, size, a progress bar of hash symbols, and a 100% completion status.

Package Name	Size	Progress	Status
setuptools-40.2.0	497 KB	#####	100%
certifi-2018.8.24	137 KB	#####	100%
wheel-0.31.1	82 KB	#####	100%
wincertstore-0.2	14 KB	#####	100%
pip-10.0.1	1.6 MB	#####	100%



A screenshot of an Anaconda Prompt window titled "관리자: Anaconda Prompt". The terminal displays instructions for activating and deactivating a virtual environment. It shows the command "\$ conda activate test" for activation and "\$ conda deactivate" for deactivation. The prompt at the bottom is "(base) c:\wpy_data>".

```
# To activate this environment, use
#
#     $ conda activate test
#
# To deactivate an active environment, use
#
#     $ conda deactivate

(base) c:\wpy_data>
```

3. 가상환경 설정

- 가상환경만들기

```
관리자: Anaconda Prompt

(base) c:\py_data>conda info --envs
# conda environments:
#
base                * C:\Users\Wgugu9999\Anaconda3
test                C:\Users\Wgugu9999\Anaconda3\envs\test

(base) c:\py_data>
```

```
1 #설치 된 가상환경 리스트 확인
2 conda info --envs
3
4 #가상환경 활성화
5 #예) activate test
6 activate 가상환경명
7
8 #가상환경 비활성화
9 #예) deactivate test
10 deactivate 가상환경명
```

```
관리자: Anaconda Prompt

(base) c:\py_data>activate test

(test) c:\py_data>
```

3. 가상환경 설정

- 가상환경 삭제하기

```
1 #패키지 삭제
2 #예)conda remove --name test --all
3 conda remove --name 가상환경명 --all
4
5 #또는
6 conda remove -n 가상환경명 --all
```

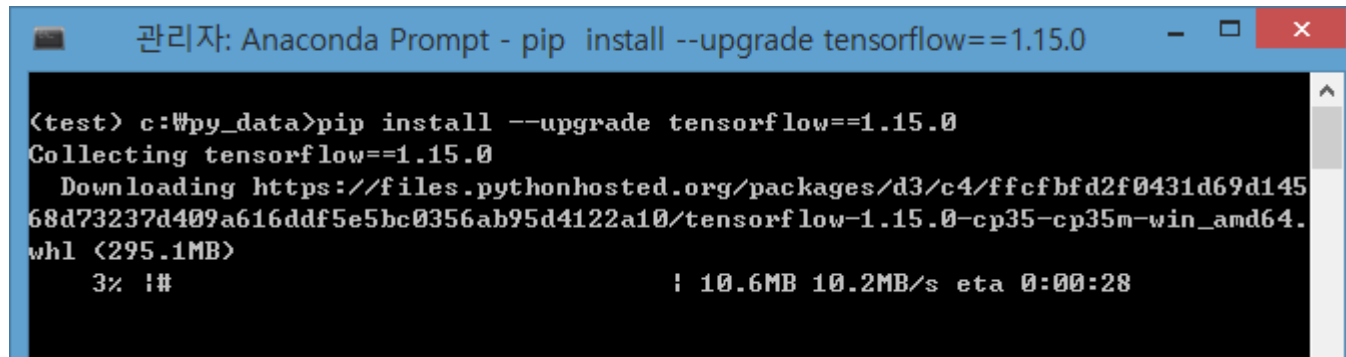
- 아나콘다는 clean명령어를 통해서 캐시를 삭제할 수 있다. 인덱스 캐시, 잠긴파일, 사용하지않는 패키지, 소스 캐시등을 삭제할 수 있다.(주의해서 사용해야한다!)

```
1 #아나콘다 클린
2 conda clean --all
3
4 #또는
5 conda clean -a
```

3. 가상환경 설정

- 아나콘다 가상환경에 Tensorflow 설치하기

(test) > **pip install --upgrade tensorflow==1.15.0**



```
관리자: Anaconda Prompt - pip install --upgrade tensorflow==1.15.0

<test> c:\wpy_data>pip install --upgrade tensorflow==1.15.0
Collecting tensorflow==1.15.0
  Downloading https://files.pythonhosted.org/packages/d3/c4/ffcfbfd2f0431d69d145
68d73237d409a616ddf5e5bc0356ab95d4122a10/tensorflow-1.15.0-cp35-cp35m-win_amd64.
whl (295.1MB)
    3% |#                               | 10.6MB 10.2MB/s eta 0:00:28
```

for CPU and GPU

\$ pip install --upgrade tensorflow==1.15.0

only GPU

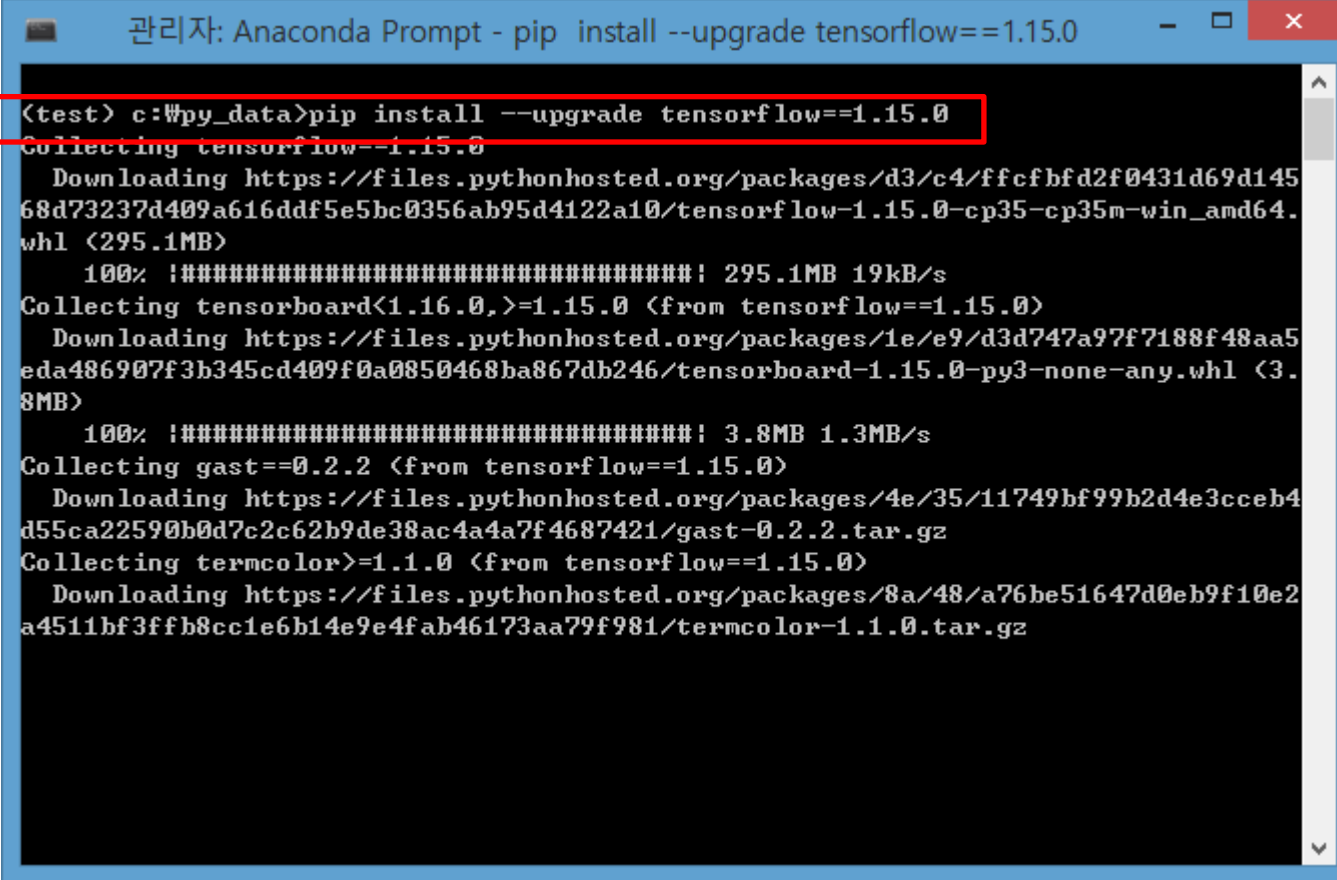
\$ pip install --upgrade tensorflow-gpu==1.15.0

only CPU

\$ pip install --upgrade tensorflow-cpu==1.15.0

3. 가상환경 설정

- tensorflow 1.15.0 버전 설치완료

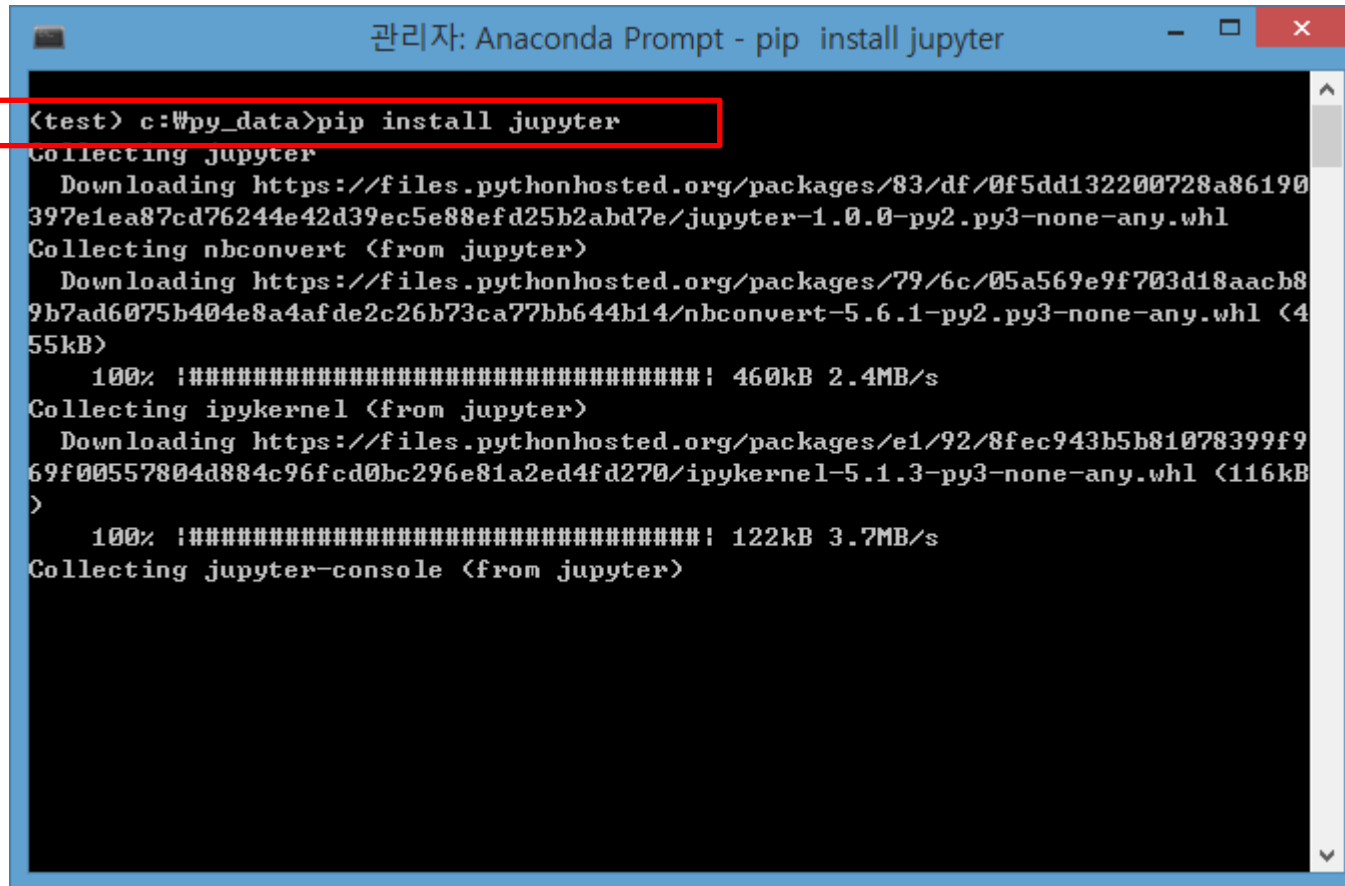


```
관리자: Anaconda Prompt - pip install --upgrade tensorflow==1.15.0

<test> c:\wpy_data>pip install --upgrade tensorflow==1.15.0
Collecting tensorflow==1.15.0
  Downloading https://files.pythonhosted.org/packages/d3/c4/ffcfbfd2f0431d69d14568d73237d409a616ddf5e5bc0356ab95d4122a10/tensorflow-1.15.0-cp35-cp35m-win_amd64.whl (295.1MB)
    100% |#####| 295.1MB 19kB/s
Collecting tensorboard<1.16.0,>=1.15.0 (from tensorflow==1.15.0)
  Downloading https://files.pythonhosted.org/packages/1e/e9/d3d747a97f7188f48aa5eda486907f3b345cd409f0a0850468ba867db246/tensorboard-1.15.0-py3-none-any.whl (3.8MB)
    100% |#####| 3.8MB 1.3MB/s
Collecting gast==0.2.2 (from tensorflow==1.15.0)
  Downloading https://files.pythonhosted.org/packages/4e/35/11749bf99b2d4e3cceb4d55ca22590b0d7c2c62b9de38ac4a4a7f4687421/gast-0.2.2.tar.gz
Collecting termcolor>=1.1.0 (from tensorflow==1.15.0)
  Downloading https://files.pythonhosted.org/packages/8a/48/a76be51647d0eb9f10e2a4511bf3ffb8cc1e6b14e9e4fab46173aa79f981/termcolor-1.1.0.tar.gz
```

3. 가상환경 설정

(test) > pip install jupyter

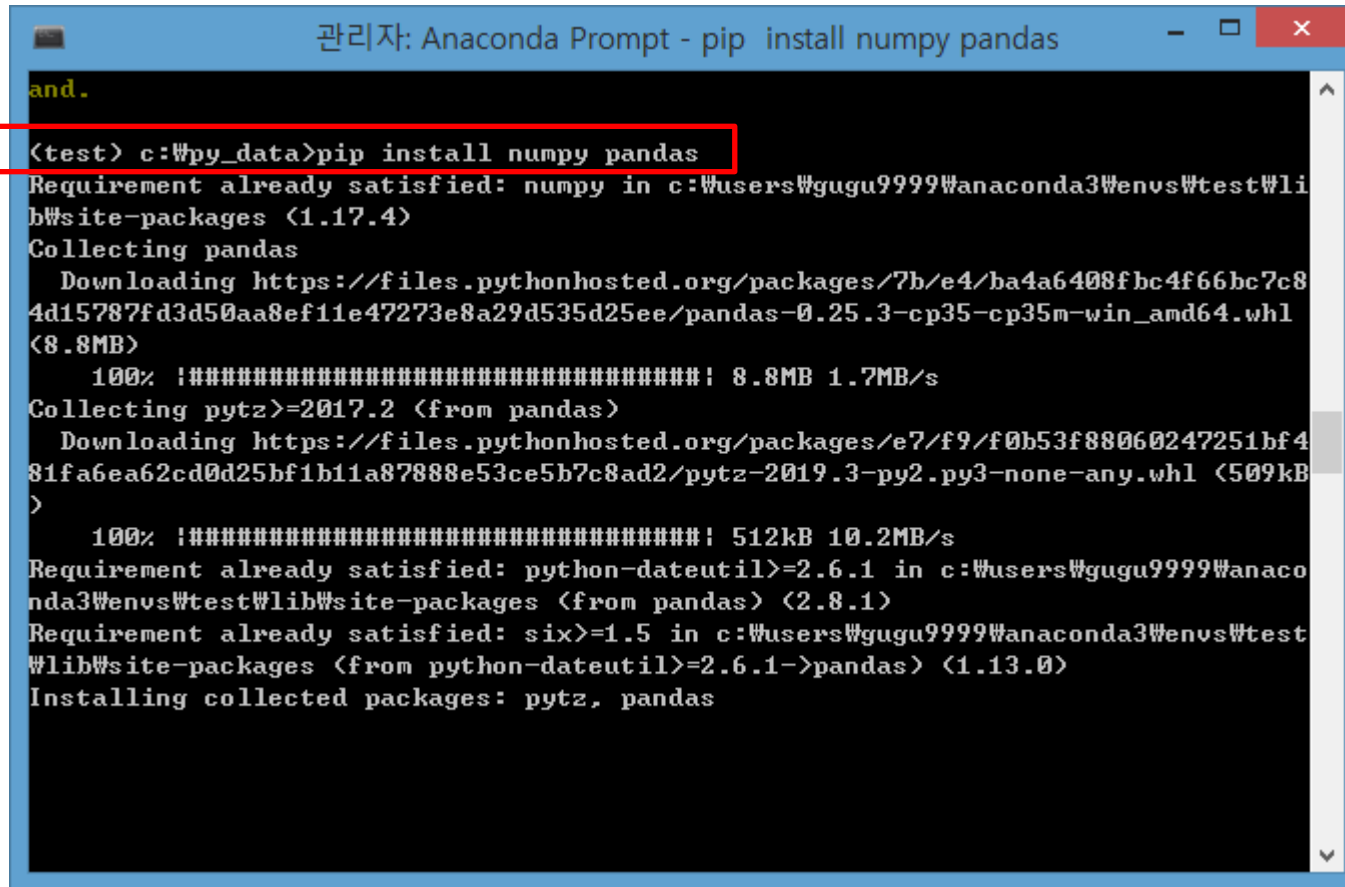


```
관리자: Anaconda Prompt - pip install jupyter

(test) c:\py_data>pip install jupyter
Collecting jupyter
  Downloading https://files.pythonhosted.org/packages/83/df/0f5dd132200728a86190397e1ea87cd76244e42d39ec5e88efd25b2abd7e/jupyter-1.0.0-py2.py3-none-any.whl
Collecting nbconvert (from jupyter)
  Downloading https://files.pythonhosted.org/packages/79/6c/05a569e9f703d18aacb89b7ad6075b404e8a4afde2c26b73ca77bb644b14/nbconvert-5.6.1-py2.py3-none-any.whl (455kB)
    100% |#####| 460kB 2.4MB/s
Collecting ipykernel (from jupyter)
  Downloading https://files.pythonhosted.org/packages/e1/92/8fec943b5b81078399f969f00557804d884c96fcd0bc296e81a2ed4fd270/ipykernel-5.1.3-py3-none-any.whl (116kB)
    100% |#####| 122kB 3.7MB/s
Collecting jupyter-console (from jupyter)
```

3. 가상환경 설정

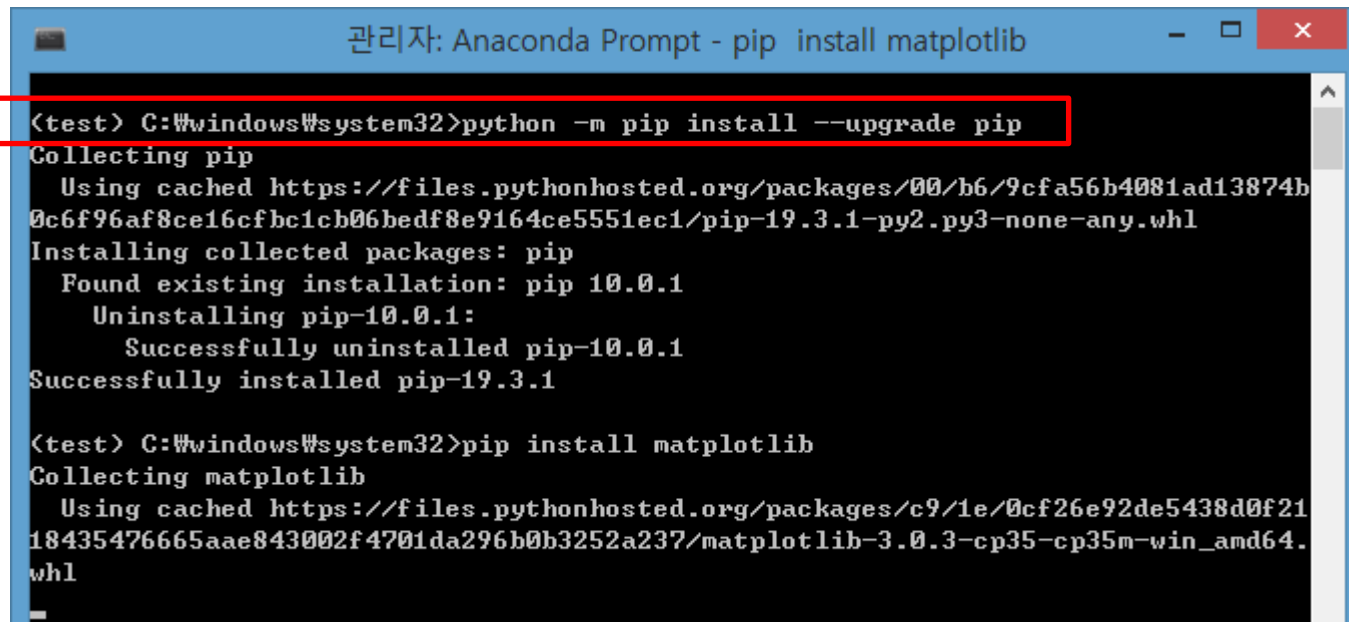
(test) > pip install numpy pandas



```
관리자: Anaconda Prompt - pip install numpy pandas
and.
<test> c:\wpy_data>pip install numpy pandas
Requirement already satisfied: numpy in c:\users\wgugu9999\anaconda3\envs\test\lib\site-packages (1.17.4)
Collecting pandas
  Downloading https://files.pythonhosted.org/packages/7b/e4/ba4a6408fbc4f66bc7c84d15787fd3d50aa8ef11e47273e8a29d535d25ee/pandas-0.25.3-cp35-cp35m-win_amd64.whl (8.8MB)
    100% |#####| 8.8MB 1.7MB/s
Collecting pytz>=2017.2 (from pandas)
  Downloading https://files.pythonhosted.org/packages/e7/f9/f0b53f88060247251bf481fa6ea62cd0d25bf1b11a87888e53ce5b7c8ad2/pytz-2019.3-py2.py3-none-any.whl (509kB)
    100% |#####| 512kB 10.2MB/s
Requirement already satisfied: python-dateutil>=2.6.1 in c:\users\wgugu9999\anaconda3\envs\test\lib\site-packages (from pandas) (2.8.1)
Requirement already satisfied: six>=1.5 in c:\users\wgugu9999\anaconda3\envs\test\lib\site-packages (from python-dateutil>=2.6.1->pandas) (1.13.0)
Installing collected packages: pytz, pandas
```

3. 가상환경 설정

(test) > pip install matplotlib



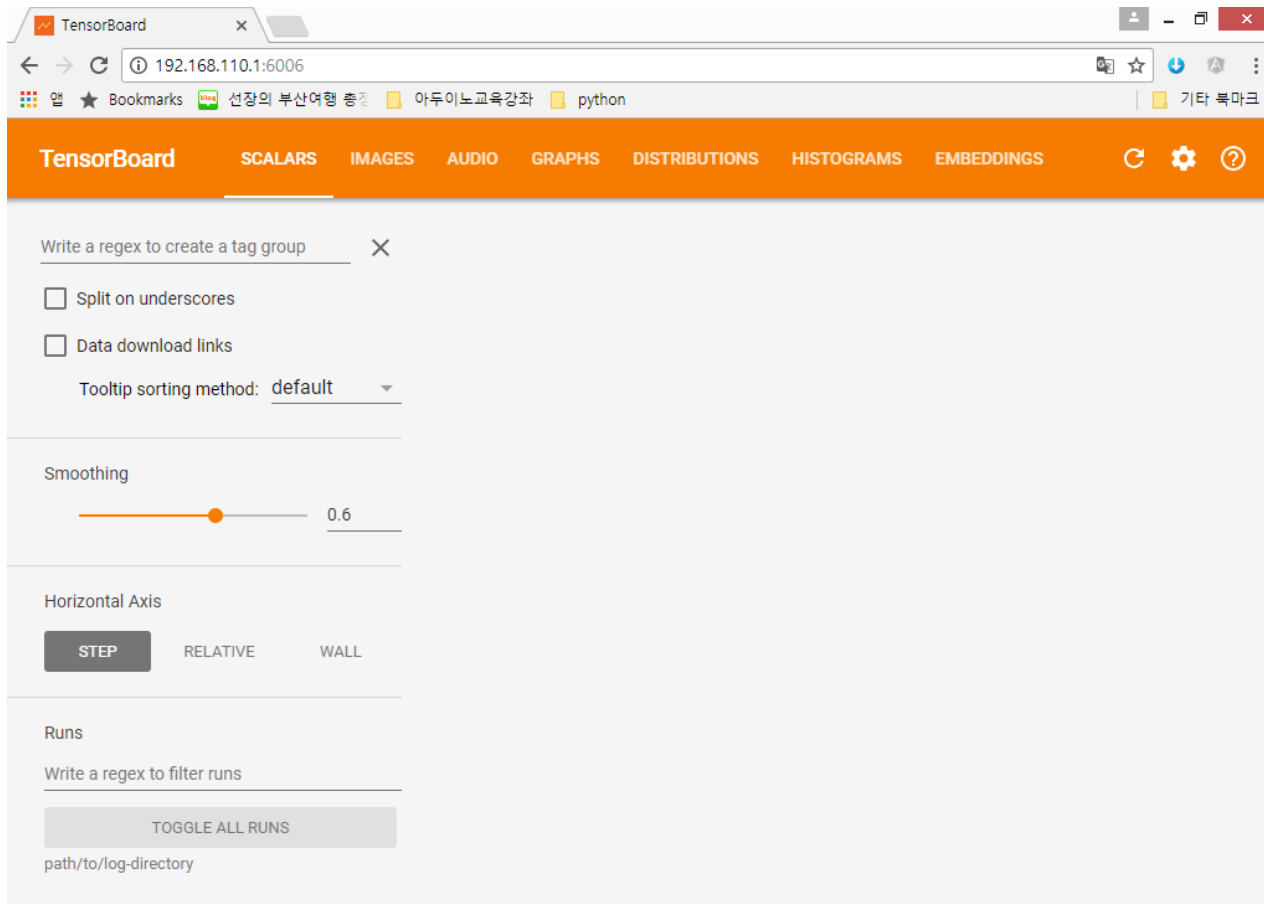
```
관리자: Anaconda Prompt - pip install matplotlib

<test> C:\Windows\system32>python -m pip install --upgrade pip
Collecting pip
  Using cached https://files.pythonhosted.org/packages/00/b6/9cfa56b4081ad13874b0c6f96af8ce16cfbc1cb06bedf8e9164ce5551ec1/pip-19.3.1-py2.py3-none-any.whl
Installing collected packages: pip
  Found existing installation: pip 10.0.1
  Uninstalling pip-10.0.1:
    Successfully uninstalled pip-10.0.1
  Successfully installed pip-19.3.1

<test> C:\Windows\system32>pip install matplotlib
Collecting matplotlib
  Using cached https://files.pythonhosted.org/packages/c9/1e/0cf26e92de5438d0f2118435476665aae843002f4701da296b0b3252a237/matplotlib-3.0.3-cp35-cp35m-win_amd64.whl
```


4. Tensorboard 설치 및 실행

- Tensorboard의 실행화면





TensorFlow

Tensorflow

- Tensorflow 개요

Tensorflow란

- Tensorflow

- 머신러닝 및 딥러닝을 위해 구글에서 배포한 오픈 소스 라이브러리
- 수치 계산을 위한 강력한 오픈소스 소프트웨어 라이브러리로 특히 대규모 머신러닝에 맞춰 튜닝되어 있다.

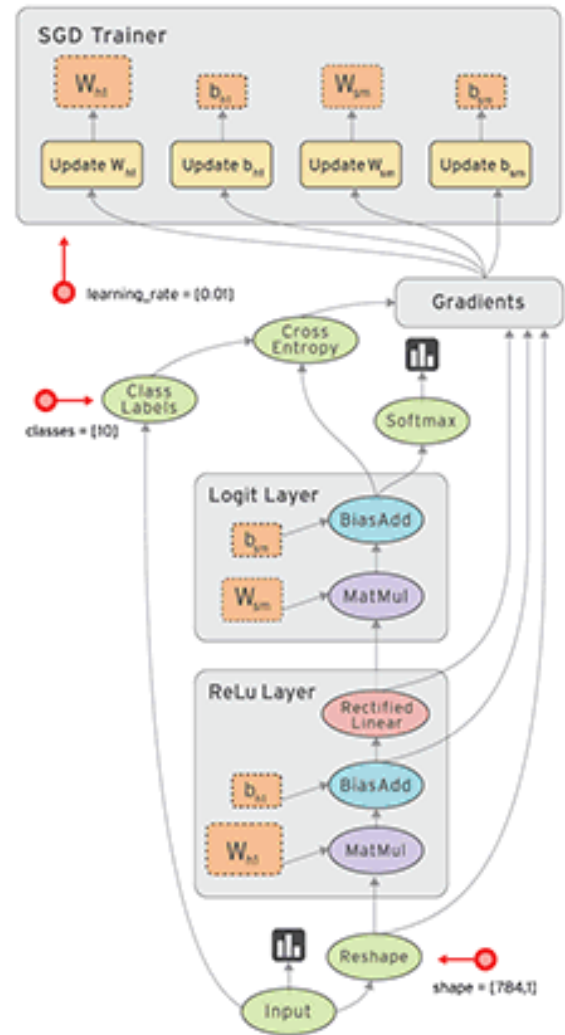
- Tensorflow의 기본원리

- 파이썬으로, 수행할 계산 그래프를 정의한 다음, 텐서플로가 최적화된 C++ 코드를 사용해 그래프를 효율적으로 실행시킨다.

그래프생성



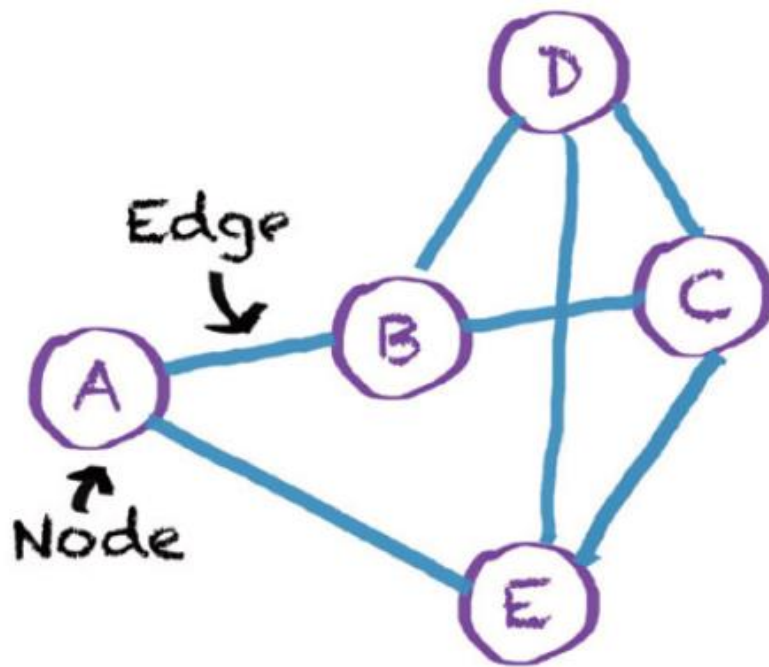
그래프실행



Tensorflow의 구성

- Tensorflow의 기본원리

- 파이썬으로 수행할 계산 그래프를 정의한 다음, 텐서플로가 최적화된 C++ 코드를 사용해 그래프를 효율적으로 실행시킨다.
- 컴퓨터공학에서 말하는 자료구조에서의 그래프 : Node + Edge로 구성



5개의 노드와 7개의 엣지로 이루어진 그래프 예시

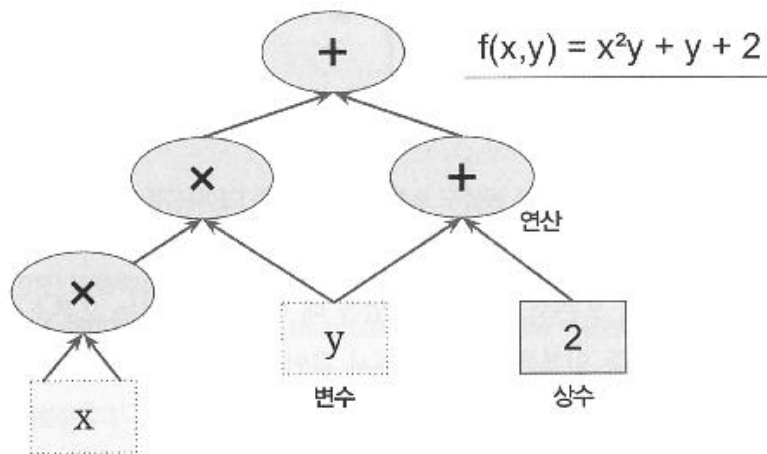
Tensorflow의 구성

• Tensorflow의 기본원리

- 텐서(Tensorflow의 데이터) + 연산으로 구성되는 그래프를 생성한 후, 실행
- 텐서플로는 **노드에 연산 operator, 변수 variable, 상수 Constant**등을 정의하고, 노드 간의 연결인 엣지를 통해 텐서를 주고받으면서 계산을 수행한다.

메모

텐서플로 라이브러리가 그래프 생성과 실행을 분리하는 방식을 취하는 이유는 생성과정에서 많은 그래프를 생성했더라도 실행과정에서는 전체 그래프 중 일부만 사용할 수도 있기 때문이다. 따라서 실행과정에서는 전체 그래프 대신 실제 연산에 필요한 그래프만 활용해서 최대한 효율적으로 연산을 수행하게 된다. 이런 기법을 전문적인 용어로 Lazy Evaluation이라고 한다.



그래프를 생성하는 방법

- 3 + 4 에 대한 연산을 Tensorflow를 활용해 프로그래밍 해보자.

① 먼저 텐서플로우 라이브러리를 import 한다.

```
1 import tensorflow as tf
```

② 상수 constant 값을 표현하는 tf.constant 노드를 2개 정의하고, 파이썬의 print 함수를 이용해서 노드의 값을 출력해보자.

```
3 # 그래프 노드를 정의하고 출력합니다.  
4 # 출력값 : Tensor("Const:0", shape=(), dtype=float32) Tensor("Const_1:0",  
5   shape=(), dtype=float32)  
6 node1 = tf.constant(3.0, dtype=tf.float32)  
7 node2 = tf.constant(4.0) # 암시적으로 tf.float32 타입으로 선언될 것입니다.  
8 print(node1, node2)
```

출력 결과

```
[3.0, 4.0]
```

그래프를 생성하는 방법

④ 더 복잡한 그래프 구조를 생성하면 더 복잡한 연산을 수행할 수 있다. 이제 2개의 노드의 값을 더하는 연산을 수행하는 `tf.add` 노드인 `node3`를 정의하고 세션을 실행해 값을 출력해보자.

```
14 # 2개의 노드의 값을 더하는 연산을 수행하는 node3을 정의합니다.
15 # 출력값:
16 # node3: Tensor("Add:0", shape=(), dtype=float32)
17 # sess.run(node3): 7.0
18 node3 = tf.add(node1, node2)
19 print("node3:", node3)
20 print("sess.run(node3):", sess.run(node3))
```

출력 결과

```
node3: Tensor("Add:0", shape=(), dtype=float32)
sess.run(node3): 7.0
```

그래프를 생성하는 방법

⑤ 마지막으로 세션을 종료한다.

```
22 sess.close()
```

⑥ 텐서플로는 텐서보드 Tensorboard라는 시각화툴을 활용해서 그래프 구조를 시각화해 볼 수 있다. node1, node2, node3로 만든 그래프를 텐서보드로 시각화하면 아래 그림과 같이 그래프 구조를 나타내는 그림을 얻을 수 있다.



TensorBoard를 이용한 그래프 시각화

그래프를 생성하는 방법

Tensorflow v1.x

```
# -*- coding: utf-8 -*-
```

```
import tensorflow as tf
```

```
# 그래프 노드를 정의하고 출력합니다.
```

```
# 출력값 : Tensor("Const:0", shape=(), dtype=float32) Tensor("Const_1:0", shape=(), dtype=float32)
```

```
node1 = tf.constant(3.0, dtype=tf.float32)
```

```
node2 = tf.constant(4.0) # 암시적으로 tf.float32 타입으로 선언될 것입니다.
```

```
print(node1, node2)
```

```
# 세션을 열고 그래프를 실행합니다.
```

```
# 출력값 : [3.0, 4.0]
```

```
sess = tf.Session()
```

```
print(sess.run([node1, node2]))
```

```
# 두개의 노드의 값을 더하는 연산을 수행하는 node3을 정의합니다.
```

```
# 출력값:
```

```
# node3: Tensor("Add:0", shape=(), dtype=float32)
```

```
# sess.run(node3): 7.0
```

```
node3 = tf.add(node1, node2)
```

```
print("node3:", node3)
```

```
print("sess.run(node3):", sess.run(node3))
```

```
sess.close()
```

그래프를 생성하는 방법

- 실행화면(Tensorflow v1.15.0)

```
In [1]: # -*- coding: utf-8 -*-

import tensorflow as tf

# 그래프 노드를 정의하고 출력합니다.
# 출력값 : Tensor("Const:0", shape=(), dtype=float32) Tensor("Const_1:0", shape=(), dtype=float32)
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # 일시적으로 tf.float32 타입으로 선언될 것입니다.
print(node1, node2)

# 세션을 열고 그래프를 실행합니다.
# 출력값 : [3.0, 4.0]
sess = tf.Session()
print(sess.run([node1, node2]))

# 두개의 노드의 값을 더하는 연산을 수행하는 node3을 정의합니다.
# 출력값:
# node3: Tensor("Add:0", shape=(), dtype=float32)
# sess.run(node3): 7.0
node3 = tf.add(node1, node2)
print("node3:", node3)
print("sess.run(node3):", sess.run(node3))

sess.close()

Tensor("Const:0", shape=(), dtype=float32) Tensor("Const_1:0", shape=(), dtype=float32)
[3.0, 4.0]
node3: Tensor("Add:0", shape=(), dtype=float32)
sess.run(node3): 7.0
```

그래프를 생성하는 방법

Tensorflow v2.0

```
# -*- coding: utf-8 -*-
```

```
import tensorflow as tf
```

```
# 그래프 노드를 정의하고 출력합니다.
```

```
# 출력값 : <tf.Tensor: id=0, shape=(), dtype=float32, numpy=3.0>, <tf.Tensor: id=1, shape=(), dtype=float32, numpy=4.0>
```

```
node1 = tf.constant(3.0, dtype=tf.float32)
```

```
node2 = tf.constant(4.0) # 암시적으로 tf.float32 타입으로 선언될 것입니다.
```

```
print(node1, node2)
```

```
# 그래프를 실행합니다.
```

```
# 출력값 : [3.0, 4.0]
```

```
print(node1.numpy(), node2.numpy())
```

```
# 두개의 노드의 값을 더하는 연산을 수행하는 node3을 정의합니다.
```

```
# 출력값:
```

```
# node3: <tf.Tensor: id=2, shape=(), dtype=float32, numpy=7.0>
```

```
# node3.numpy(): 7.0
```

```
node3 = tf.add(node1, node2)
```

```
print("node3:", node3)
```

```
print("node3.numpy():", node3.numpy())
```

그래프를 생성하는 방법

- 실행화면(Tensorflow v2.0)

```
# -*- coding: utf-8 -*-

import tensorflow as tf

# 그래프 노드를 정의하고 출력합니다.
# 출력값 : <tf.Tensor: id=0, shape=(), dtype=float32, numpy=3.0>, <tf.Tensor: id=1, shape=(), dtype=float32, numpy=4.0>
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # 암시적으로 tf.float32 타입으로 선언될 것입니다.
print(node1, node2)

# 그래프를 실행합니다.
# 출력값 : [3.0, 4.0]
print(node1.numpy(), node2.numpy())

# 두개의 노드의 값을 더하는 연산을 수행하는 node3을 정의합니다.
# 출력값:
# node3: <tf.Tensor: id=2, shape=(), dtype=float32, numpy=7.0>
# node3.numpy(): 7.0
node3 = tf.add(node1, node2)
print("node3:", node3)
print("node3.numpy():", node3.numpy())

tf.Tensor(3.0, shape=(), dtype=float32) tf.Tensor(4.0, shape=(), dtype=float32)
3.0 4.0
node3: tf.Tensor(7.0, shape=(), dtype=float32)
node3.numpy(): 7.0
```

그래프를 생성하는 방법

- Tensorflow v2에서 v1 코드 실행시 오류

```
In [2]: # -*- coding: utf-8 -*-

import tensorflow as tf

# 그래프 노드를 정의하고 출력합니다.
# 출력값 : Tensor('Const:0', shape=(), dtype=float32) Tensor('Const_1:0', shape=(), dtype=float32)
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # 일시적으로 tf.float32 타입으로 선언될 것입니다.
print(node1, node2)

# 세션을 열고 그래프를 실행합니다.
# 출력값 : [3.0, 4.0]
sess = tf.Session()
print(sess.run([node1, node2]))

# 두개의 노드의 값을 더하는 연산을 수행하는 node3을 정의합니다.
# 출력값:
# node3: Tensor('Add:0', shape=(), dtype=float32)
# sess.run(node3): 7.0
node3 = tf.add(node1, node2)
print("node3:", node3)
print("sess.run(node3):", sess.run(node3))

sess.close()
```

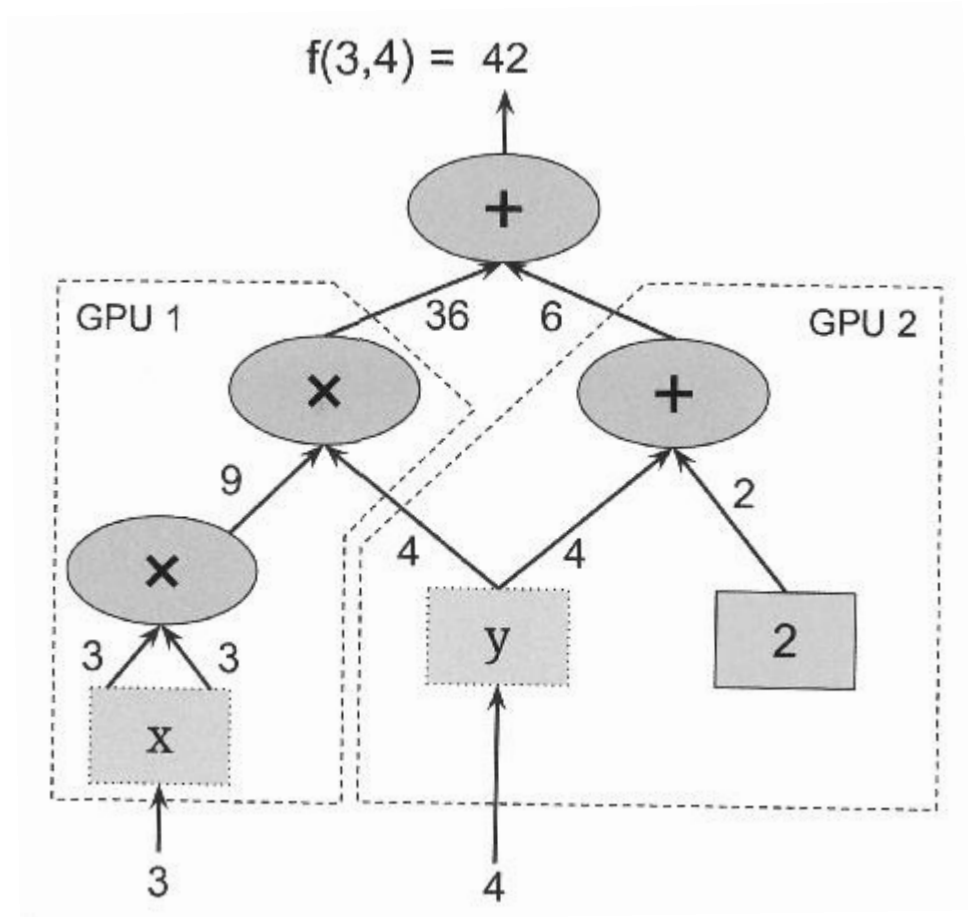
```
tf.Tensor(3.0, shape=(), dtype=float32) tf.Tensor(4.0, shape=(), dtype=float32)
```

```
AttributeError                                Traceback (most recent call last)
<ipython-input-2-7cd6027b25cf> in <module>
    11 # 세션을 열고 그래프를 실행합니다.
    12 # 출력값 : [3.0, 4.0]
----> 13 sess = tf.Session()
    14 print(sess.run([node1, node2]))
    15
```

```
AttributeError: module 'tensorflow' has no attribute 'Session'
```

Tensorflow의 특징

- Tensorflow
 - 여러 CPU, GPU, 서버에서 병렬 계산





TensorFlow

Tensorflow

- Tensorflow의 특징

1. Tensorflow의 기본 개념

- 용어

- ① 오퍼레이션(Operation)

그래프 상의 노드는 오퍼레이션(줄임말 *op*)으로 불린다. 오퍼레이션은 하나 이상의 *텐서*를 받을 수 있다. 오퍼레이션은 계산을 수행하고, 결과를 하나 이상의 텐서로 반환할 수 있습니다.

- ② 텐서(Tensor)

텐서플로우는 텐서(Tensors)라는 독특한 자료형을 가지며, 내부적으로 모든 데이터는 텐서를 통해 표현됩니다. 텐서는 일종의 다차원 배열인데, 그래프 내의 오퍼레이션 간에는 텐서만이 전달됩니다. (Caffe의 Blob과 유사합니다.)

- ③ 세션(Session)

그래프를 실행하기 위해서는 세션 객체가 필요합니다. 세션은 오퍼레이션의 실행 환경을 캡슐화한 것입니다.

- ④ 변수(Variables)

변수는 그래프의 실행시, 파라미터를 저장하고 갱신하는데 사용됩니다. 메모리 상에서 텐서를 저장하는 버퍼 역할을 합니다.

2. Tensor

- Tensorflow에서 표현되는 데이터의 형태

- 피쳐 벡터 feature vector : 피쳐를 순서대로 나열한 목록을 피쳐벡터라고 하는데, 우리가 텐서플로 코드에서 표현할 것이 바로 이 피쳐 벡터이다.
- 행렬은 벡터의 목록을 간결하게 표현하는데, 행렬의 각 열이 피쳐 벡터들이다.
- 텐서플로우에서는 행렬을 동일한 길이를 가지는 벡터들의 벡터로 표현한다.

컴퓨터가 행렬을 표현하는 방법

$[[1,2,3], [4,5,6]]$



사람이 행렬을 표현하는 방법

$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$

2. Tensor

- 텐서플로우는 뉴럴네트워크에 최적화되어 있는 개발 프레임워크이기 때문에, 그 자료형과, 실행 방식이 약간 일반적인 프로그래밍 방식과 상의하다.
- **텐서(Tensor)**
 - 텐서플로우는 텐서(Tensors)라는 독특한 자료형을 갖는다.
 - 일반적으로 텐서플로우는 텐서를 다차원 어레이의 일반화(the n-dimensional abstraction of matrices) 혹은 아무 차원이나 갖을 수 있는 값들의 집합(A tensor consists of a set of primitive values shaped into an array of any number of dimensions.) 으로 표현하기도 한다. 그렇기 때문에 텐서를 기존에 Numpy Array와 같게 생각해도 무방하다. 이러한 텐서는 Rank, Shape, Type 3가지 구성요소를 갖는다.

2. Tensor

- 텐서의 구성요소

(1) 텐서의 Rank

Numpy Array에서는 어레이의 구조를 차원(dimesion)으로 표현했지만, 텐서는 Rank라는 용어를 통해 텐서의 구조를 표현한다.

- ① 0 Rank 텐서 : 스칼라를 말한다.
- ② 1 rank tensor : 1 Rank 텐서란 1-D(1차원) 벡터와 같다.
- ③ 2 rank tensor : 2 Rank 텐서란 2-D(2차원) 벡터 즉 행렬을 말한다.
- ④ 3 rank tensor : 3 Rank 텐서란 3-D(3차원) 벡터를 의미이다.

```
3 # 0 rank tensor  
[1., 2., 3.] # 1 rank tensor  
[[1., 2., 3.], [4., 5., 6.]] # 2 rank tensor  
[[[1., 2., 3.], [7., 8., 9.]]] # 3 rank tensor
```

2. Tensor

- 텐서의 구성요소

(2) 텐서의 Shape

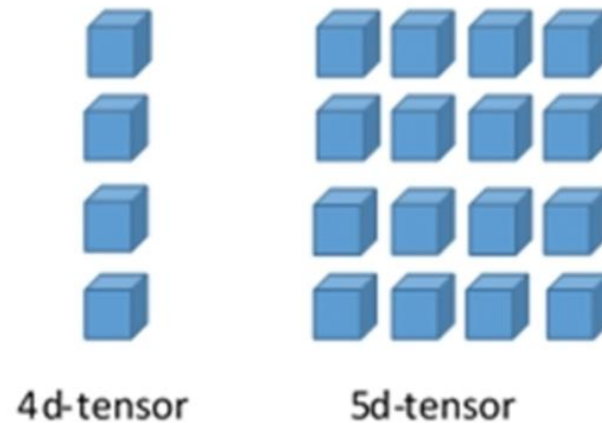
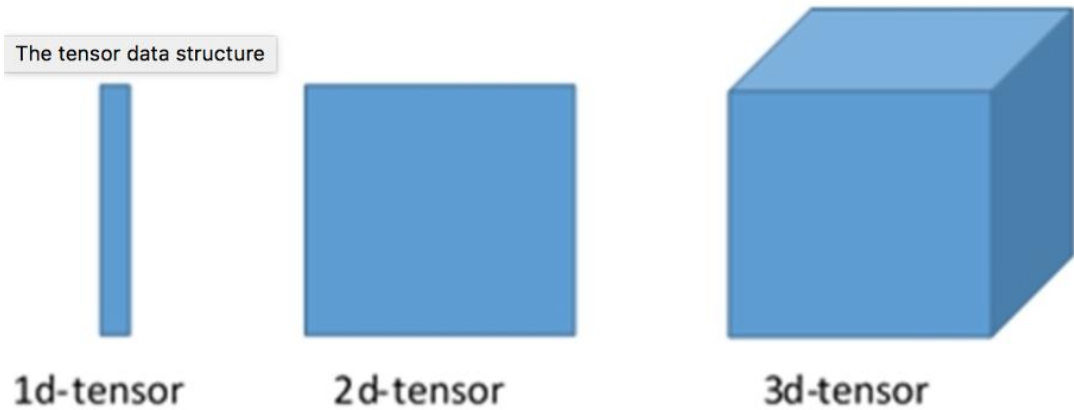
텐서의 Shape란 텐서가 몇 개의 행과 열을 갖는지 의미한다. 위 살펴봤던 Rank의 텐서의 Shape는 아래와 같다.

```
3 # 0 shape []  
[1., 2., 3.] # shape [3]  
[[1., 2., 3.], [4., 5., 6.]] # shape [2, 3]  
[[[1., 2., 3.], [7., 8., 9.]]] # shape [2, 1, 3]
```

2. Tensor

- 텐서의 구성요소

- (2) 텐서의 Shape



2. Tensor

- 텐서의 구성요소

(3) 텐서의 타입(Type)

텐서의 타입이란 텐서의 값이 어떤 타입인지를 의미하고 텐서는 아래와 같은 타입을 갖는다. https://www.tensorflow.org/api_docs/

- `tf.float16` : 16-bit half-precision floating-point.
- `tf.float32` : 32-bit single-precision floating-point.
- `tf.float64` : 64-bit double-precision floating-point.
- `tf.bfloat16` : 16-bit truncated floating-point.
- `tf.complex64` : 64-bit single-precision complex.
- `tf.complex128` : 128-bit double-precision complex.
- `tf.int8` : 8-bit signed integer.
- `tf.uint8` : 8-bit unsigned integer.
- `tf.uint16` : 16-bit unsigned integer.
- `tf.int16` : 16-bit signed integer.
- `tf.int32` : 32-bit signed integer.

3. Tensor의 유형

1) 상수형 (Constant) 텐서

상수형은 말 그대로 상수를 저장하는 데이터 형이다.

```
tf.constant(value, dtype=None, shape=None, name='Const',  
            verify_shape=False)
```

와 같은 형태로 정의 된다. 각 정의되는 내용을 보면

- value : 상수의 값이다.
- dtype : 상수의 데이터형이다. tf.float32와 같이 실수,정수등의 데이터 타입을 정의한다.
- shape : 행렬의 차원을 정의한다. shape=[3,3]으로 정의해주면, 이 상수는 3x3 행렬을 저장하게 된다.
- name : name은 이 상수의 이름을 정의한다. tensorboard에 표시되는 이름

3. Tensor의 유형

간단한 예제를 하나 보자.

- a, b, c 상수에, 각각 5, 10, 2 의 값을 넣은 후에, $d = a * b + c$ 를 계산해서 계산 결과 d 를 출력하려고 한다.

```
import tensorflow as tf

a = tf.constant([5], dtype=tf.float32)
b = tf.constant([10], dtype=tf.float32)
c = tf.constant([2], dtype=tf.float32)

d = a*b+c

print(d)
```

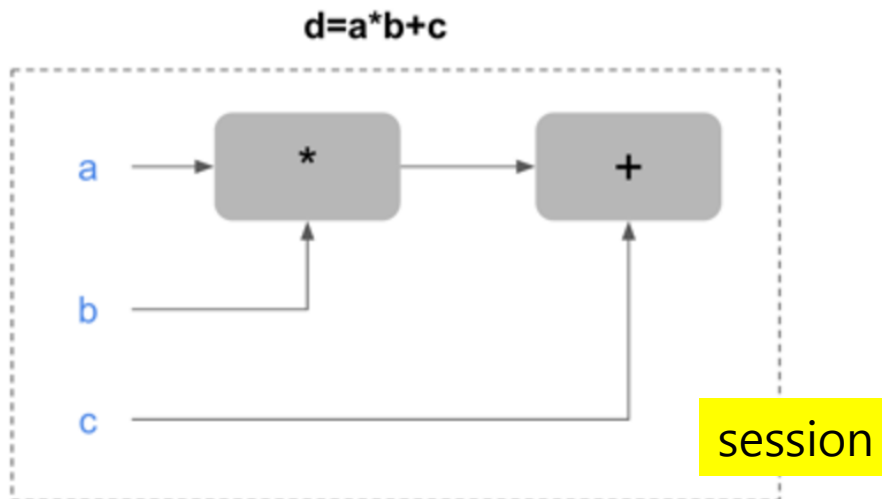
그런데, 막상 실행해보면, $a * b + c$ 의 값이 아니라 다음과 같이 Tensor... 라는 문자열이 출력된다.

[출력] Tensor("add_8:0", shape=(1,), dtype=float32)

3. Tensor의 유형

- 그래프와 세션의 개념

먼저 그래프와 세션이라는 개념을 이해해야 텐서플로우의 프로그래밍 모델을 이해할 수 있다. 아래의 $d=a*b+c$ 에서 d 역시 계산을 수행하는 것이 아니라 다음과 같이 $a*b+c$ 그래프를 정의하는 것이다.



- 실제로 값을 뽑아내려면, 이 정의된 그래프에 a,b,c 값을 넣어서 실행해야 하는데, 세션 (Session)을 생성하여, 그래프를 실행해야 한다. 세션은 그래프를 인자로 받아서 실행을 해주는 일종의 러너(Runner)라고 생각하면 된다.

3. Tensor의 유형

- 위의 코드를 수정해보자

```
import tensorflow as tf

a = tf.constant([5],dtype=tf.float32)
b = tf.constant([10],dtype=tf.float32)
c = tf.constant([2],dtype=tf.float32)

d = a*b+c

sess = tf.Session()
result = sess.run(d)
print(result)
```

- `tf.Session()`을 통하여 세션을 생성하고, 이 세션에 그래프 `d`를 실행하도록 `sess.run(d)`를 실행한다.

3. Tensor의 유형

- 이 그래프의 실행결과는 리턴값으로 result에 저장이 되고, 출력을 해보면 다음과 같이 정상적으로 52라는 값이 나오는 것을 볼 수 있다.

```
In [40]: import tensorflow as tf

a = tf.constant([5], dtype=tf.float32)
b = tf.constant([10], dtype=tf.float32)
c = tf.constant([2], dtype=tf.float32)

d = a*b+c

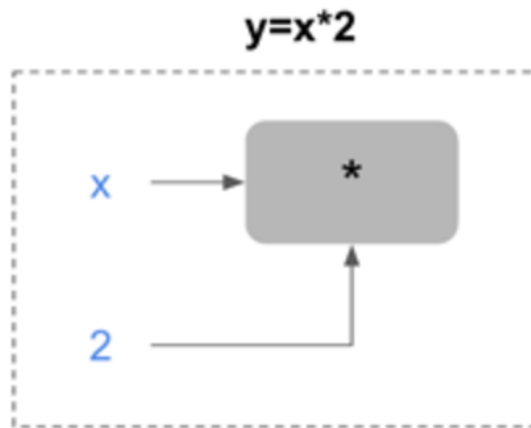
sess = tf.Session()
result = sess.run(d)
print result
```

```
[ 52.]
```

3. Tensor의 유형

2) 플레이스 홀더 (Placeholder)

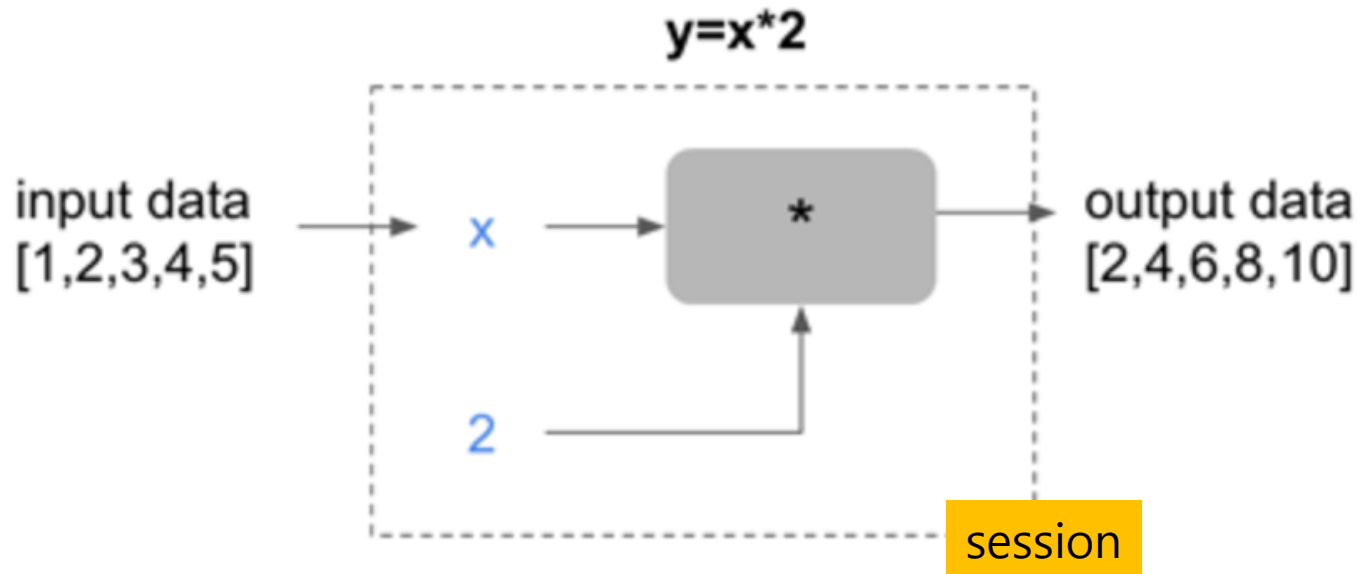
- Placeholder 란 값을 나중에 정의하는 특이한 자료형
- $y = x * 2$ 를 그래프를 통해서 실행한다고 하자. 입력 값으로는 1,2,3,4,5를 넣고, 출력은 2,4,6,8,10을 기대한다고 하자. 이렇게 여러 입력 값을 그래프에서 넣는 경우는 머신러닝에서 $y=W*x + b$ 와 같은 그래프가 있다고 할 때, x 는 학습을 위한 데이터가 된다. 즉, 플레이스 홀더라는 데이터 타입은 학습을 위한 학습용 데이터를 위한 데이터 타입이다.
- $y=x*2$ 를 정의하면 내부적으로 다음과 같은 그래프가 된다.



3. Tensor의 유형

2) 플레이스 홀더 (Placeholder)

그러면, x에는 값을 1,2,3,4,5를 넣어서 결과값을 그래프를 통해서 계산해 내야 한다. 개념적으로 보면 다음과 같다.



이렇게 학습용 데이터를 담는 그릇을 플레이스홀더(placeholder)라고 한다.

3. Tensor의 유형

2) 플레이스 홀더 (Placeholder)

플레이스홀더에 대해서 알아보면, 플레이스 홀더의 위의 그래프에서 x 즉 입력값을 저장하는 일종의 통(버킷)이다.

- **tf.placeholder(dtype,shape,name)** 으로 정의된다.

플레이스 홀더 정의에 사용되는 변수들을 보면

- dtype : 플레이스홀더에 저장되는 데이터형이다. tf.float32와 같이 실수,정수등의 데이터 타입을 정의한다.
- shape : 행렬의 차원을 정의한다. shapre=[3,3]으로 정의해주면, 이 플레이스홀더는 3x3 행렬을 저장하게 된다.
- name : name은 이 플레이스 홀더의 이름을 정의한다.

그러면 이 x에 학습용 데이터를 어떻게 넣을 것인가? 이를 피딩(feeding)이라고 한다.

3. Tensor의 유형

- 처음 `input_data=[1,2,3,4,5]`으로 정의하고,
`x=tf.placeholder(dtype=tf.float32)`를 이용하여, `x`를 float32 데이터형을 가지는 플레이스 홀더로 정의하자. `shape`은 편의상 생략하였다.
- 그리고 **`y=x * 2`로 그래프를 정의하였다.**

```
import tensorflow as tf

input_data = [1,2,3,4,5]
x = tf.placeholder(dtype=tf.float32)
y = x * 2

sess = tf.Session()
result = sess.run(y,feed_dict={x:input_data})

print(result)
```

3. Tensor의 유형

- 세션이 실행될때, x라는 통에 값을 하나씩 집어 넣는데, (앞에서도 말했듯이 이를 피딩이라고 한다.)
- sess.run(y, feed_dict={x:input data}) 와 같이 세션을 통해서 그래프를 실행할 때, feed dict 변수를 이용해서 플레이스홀더 x에, input data를 피드하면, 세션에 의해서 그래프가 실행되면서 x는 feed_dict에 의해서 정해진 피드 데이터 [1,2,3,4,5]를 하나씩 읽어서 실행한다.

```
import tensorflow as tf

input_data = [1,2,3,4,5]
x = tf.placeholder(dtype=tf.float32)
y = x * 2

sess = tf.Session()
result =
sess.run(y, feed_dict={x:input_data})

print result
```


3. Tensor의 유형

3) 변수형 (Variable)

- 마지막 데이터형은 변수형으로, $y=W*x+b$ 라는 학습용 가설이 있을때, x 가 입력데이터 였다면, W 와 b 는 학습을 통해서 구해야 하는 값이 된다. 이를 변수(Variable)이라고 하는데, 변수형은 Variable 형의 객체로 생성이 된다.

```
tf.Variable.__init__(initial_value=None, trainable=True,  
                     collections=None, validate_shape=True,  
                     caching_device=None, name=None, variable_def=None,  
                     dtype=None, expected_shape=None, import_scope=None)
```

- 변수형에 값을 넣는 것은 다음과 같이 한다.
- `var = tf.Variable([1,2,3,4,5], dtype=tf.float32)`

3. Tensor의 유형

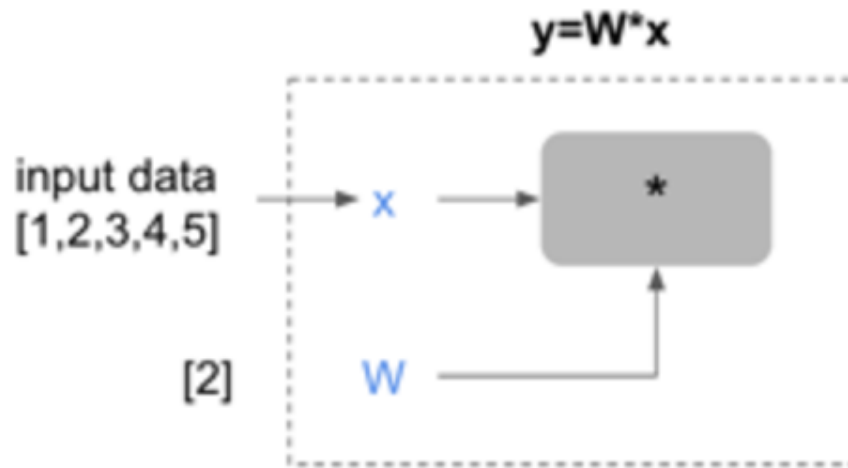
- 코드

```
import tensorflow as tf
input_data = [1,2,3,4,5]
x = tf.placeholder(dtype=tf.float32)
W = tf.Variable([2],dtype=tf.float32)
y = W*x
sess = tf.Session()
result = sess.run(y,feed_dict={x:input_data})
print(result)
```

3. Tensor의 유형

우리가 기대하는 결과는 다음과 같다.

- $y=W*x$ 와 같은 그래프를 가지고, x 는 $[1,2,3,4,5]$ 값을 피딩하면서, 변수 W 에 지정된 2를 곱해서 결과를 내기를 바란다.



- 그렇지만 코드를 실행해보면 다음과 같이 에러가 출력되는 것을 확인할 수 있다.

3. Tensor의 유형

- 오류발생

```
In [47]: import tensorflow as tf

input_data = [1,2,3,4,5]
x = tf.placeholder(dtype=tf.float32)
W = tf.Variable([2],dtype=tf.float32)
y = W*x

sess = tf.Session()
result = sess.run(y,feed_dict={x:input_data})

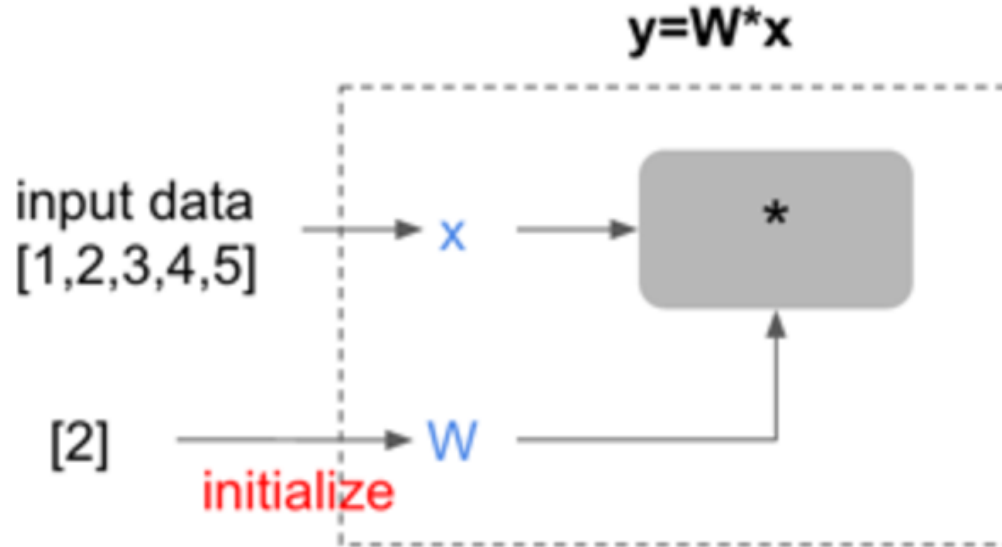
print result
```

```
-----
FailedPreconditionError                                Traceback (most recent call last)
<ipython-input-47-964facc7a53c> in <module>()
      7
      8 sess = tf.Session()
----> 9 result = sess.run(y,feed_dict={x:input_data})
     10
     11 print result
```

3. Tensor의 유형

3) 변수형 (Variable)

- 이유는 텐서플로우에서 변수형은 그래프를 실행하기 전에 초기화를 해줘야 그 값이 변수에 지정이 된다.



3. Tensor의 유형

3) 변수형 (Variable)

- 변수의 초기화

세션을 초기화 하는 순간 변수 W 에 그 값이 지정되는데, 초기화를 하는 방법은 다음과 같이 변수들을 `global_variables_initializer()` 를 이용해서 초기화 한 후, 초기화된 결과를 세션에 전달해 줘야 한다.

```
init = tf.global_variables_initializer()  
sess.run(init)
```

2. Tensorflow의 자료형

- 완성 코드

```
import tensorflow as tf

input_data = [1,2,3,4,5]
x = tf.placeholder(dtype=tf.float32)
W = tf.Variable([2],dtype=tf.float32)
y = W*x

sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
result = sess.run(y,feed_dict={x:input_data})

print(result)
```

2. Tensorflow의 자료형

- 결과화면

초기화를 수행한 후, 코드를 수행해보면 다음과 같이 우리가 기대했던 결과가 출력됨을 확인할 수 있다.

```
In [23]: 1 import tensorflow as tf
          2
          3 input_data = [1,2,3,4,5]
          4 x = tf.placeholder(dtype=tf.float32)
          5 W = tf.Variable([2], dtype=tf.float32)
          6 y = W*x
          7
          8 sess = tf.Session()
          9 init = tf.global_variables_initializer()
         10 sess.run(init)
         11 result = sess.run(y, feed_dict={x:input_data})
         12
         13 print(result)
```

```
[ 2.  4.  6.  8. 10.]
```


Tensorflow#2: 행렬과 텐서플로우

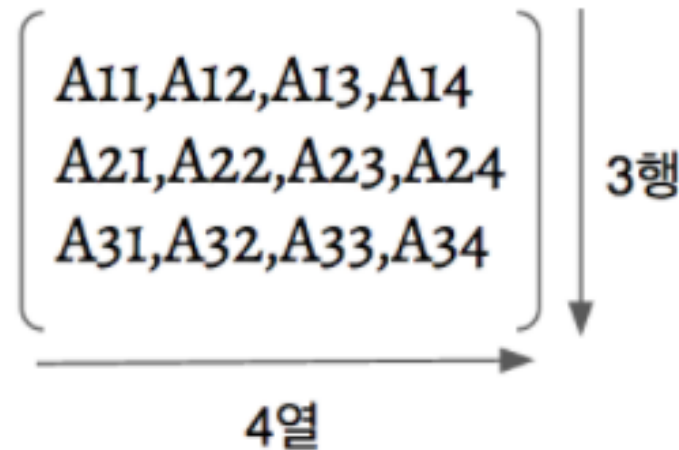
0. Tensorflow & Matrix

- 머신러닝은 거의 모든 연산을 행렬을 활용한다. 텐서플로우도 이 행렬을 기반으로 하고, 이 행렬의 차원을 shape 라는 개념으로 표현하는데, 행렬에 대한 기본적인 개념이 없으면 헷갈리기 좋다. 그래서 이 글에서는 간략하게 행렬의 기본 개념과 텐서플로우내에서 표현 방법에 대해서 알아보도록 한다

1. 행렬의 기본 개념

1-1. 행과 열

- 행렬의 가장 기본 개념은 행렬이다. $m \times n$ 행렬이 있을때, m 은 행, n 은 열을 나타내며, 행은 세로의 줄수, 열은 가로줄 수 를 나타낸다. 아래는 3×4 (3행 4열) 행렬이다.



1. 행렬의 기본 개념

1-2. 곱셈

- 곱셈은 앞의 행렬에서 행과, 뒤의 행렬의 열을 순차적으로 곱해준다.
- 아래 그림을 보면 쉽게 이해가 될 것이다.

$$\begin{array}{c} \xrightarrow{\text{red}} \\ \left[\begin{array}{c} A_{11}, A_{12}, A_{13} \\ A_{21}, A_{22}, A_{23} \\ A_{31}, A_{32}, A_{33} \end{array} \right] \times \left[\begin{array}{c} B_{11}, B_{12} \\ B_{21}, B_{22} \\ B_{31}, B_{32} \end{array} \right] = \left[\begin{array}{c} A_{11} * B_{11} + A_{12} * B_{21} + A_{13} * B_{31} \\ \\ \end{array} \right] \\ \downarrow \text{red} \end{array}$$

$$\left[\begin{array}{c} A_{11}, A_{12}, A_{13} \\ A_{21}, A_{22}, A_{23} \\ A_{31}, A_{32}, A_{33} \end{array} \right] \times \left[\begin{array}{c} B_{11}, B_{12} \\ B_{21}, B_{22} \\ B_{31}, B_{32} \end{array} \right] = \left[\begin{array}{c} A_{11} * B_{11} + A_{12} * B_{21} + A_{13} * B_{31}, A_{11} * B_{12} + A_{12} * B_{22} + A_{13} * B_{32} \\ \\ \end{array} \right]$$

$$\left[\begin{array}{c} A_{11}, A_{12}, A_{13} \\ A_{21}, A_{22}, A_{23} \\ A_{31}, A_{32}, A_{33} \end{array} \right] \times \left[\begin{array}{c} B_{11}, B_{12} \\ B_{21}, B_{22} \\ B_{31}, B_{32} \end{array} \right] = \left[\begin{array}{c} A_{11} * B_{11} + A_{12} * B_{21} + A_{13} * B_{31}, A_{11} * B_{12} + A_{12} * B_{22} + A_{13} * B_{32} \\ A_{21} * B_{11} + A_{22} * B_{21} + A_{23} * B_{31} \\ \end{array} \right]$$

$$\left[\begin{array}{c} A_{11}, A_{12}, A_{13} \\ A_{21}, A_{22}, A_{23} \\ A_{31}, A_{32}, A_{33} \end{array} \right] \times \left[\begin{array}{c} B_{11}, B_{12} \\ B_{21}, B_{22} \\ B_{31}, B_{32} \end{array} \right] = \left[\begin{array}{c} A_{11} * B_{11} + A_{12} * B_{21} + A_{13} * B_{31}, A_{11} * B_{12} + A_{12} * B_{22} + A_{13} * B_{32} \\ A_{21} * B_{11} + A_{22} * B_{21} + A_{23} * B_{31}, A_{21} * B_{12} + A_{22} * B_{22} + A_{23} * B_{32} \\ A_{31} * B_{11} + A_{32} * B_{21} + A_{33} * B_{31}, A_{31} * B_{12} + A_{32} * B_{22} + A_{33} * B_{32} \end{array} \right]$$

1. 행렬의 기본 개념

1-2. 곱셈

- 이렇게 앞 행렬의 행과 열을 곱해나가면 결과적으로 아래와 같은 결과가 나온다.

$$\begin{pmatrix} A_{11}, A_{12}, A_{13} \\ A_{21}, A_{22}, A_{23} \\ A_{31}, A_{32}, A_{33} \end{pmatrix} \times \begin{pmatrix} B_{11}, B_{12} \\ B_{21}, B_{22} \\ B_{31}, B_{32} \end{pmatrix} = \begin{pmatrix} A_{11} * B_{11} + A_{12} * B_{21} + A_{13} * B_{31}, A_{11} * B_{12} + A_{12} * B_{22} + A_{13} * B_{32} \\ A_{21} * B_{11} + A_{22} * B_{21} + A_{23} * B_{31}, A_{21} * B_{12} + A_{22} * B_{22} + A_{23} * B_{32} \\ A_{31} * B_{11} + A_{32} * B_{21} + A_{33} * B_{31}, A_{31} * B_{12} + A_{32} * B_{22} + A_{33} * B_{32} \end{pmatrix}$$

$\begin{matrix} 3 \times 3 \\ (m \times k) \end{matrix} \qquad \begin{matrix} 3 \times 2 \\ (k \times n) \end{matrix} \qquad \begin{matrix} 3 \times 2 \\ (m \times n) \end{matrix}$

- 이때 앞의 행렬의 열과, 뒤의 행렬의 행이 같아야 곱할 수 있다.
즉 $a \times b$ 행렬과 $m \times n$ 행렬이 있을때, 이 두 행렬을 곱하려면 b 와 m 이 같아야 한다. 그리고 이 두 행렬을 곱하면 $a \times n$ 사이즈의 행렬이 나온다.

2. 텐서 플로우에서 행렬의 표현

- 행렬에 대해서 간단하게 되짚어 봤으면, 그러면 텐서 플로우에서는 어떻게 행렬을 표현하는지 알아보자.

예제코드

```
import tensorflow as tf

x = tf.constant([ [1.0,2.0,3.0] ])
w = tf.constant([ [2.0],[2.0],[2.0] ])
y = tf.matmul(x,w)
print(x.get_shape())

sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
result = sess.run(y)

print(result)
```

$$(\begin{matrix} 1.0 & 2.0 & 3.0 \end{matrix}) \times \begin{pmatrix} 2.0 \\ 2.0 \\ 2.0 \end{pmatrix}$$

실행결과

(1, 3)
[[12.]]

2. 텐서 플로우에서 행렬의 표현

- 텐서플로우에서 행렬의 곱셈은 일반 * 를 사용하지 않고, **텐서플로우 함수 "tf.matmul" 을 사용한다.**
- 중간에, `x.get_shape()`를 통해서, 행렬 `x`의 `shape`를 출력했는데, `shape`는 행렬의 차원이라고 생각하면 된다. `x`는 1행3열인 1x3 행렬이기 때문에, 위의 결과와 같이 (1,3)이 출력된다.

constant 에 저장된 행렬에 대한 곱셈을 했는데, 당연히 Variable 형에서도 가능하다.

실행결과
(1, 3)
[[12.]]

```
import tensorflow as tf

x = tf.Variable([ [1.,2.,3.] ], dtype=tf.float32)
w = tf.constant([ [2.],[2.],[2.]],
dtype=tf.float32)
y = tf.matmul(x,w)

sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
result = sess.run(y)

print(result)
```

2. 텐서 플로우에서 행렬의 표현

- 실행 결과

```
In [27]: import tensorflow as tf

x = tf.Variable([ [1.,2.,3.] ], dtype=tf.float32)
w = tf.constant([ [2.],[2.],[2.]], dtype=tf.float32)
y = tf.matmul(x,w)

sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
result = sess.run(y)

print(result)
```

```
[[ 12.]]
```

실행결과

(1, 3)

[[12.]]

2. 텐서 플로우에서 행렬의 표현

- Constant 및 Variable 뿐 아니라, Placeholder에도 행렬로 저장이 가능하다 다음은 Placeholder에 행렬 데이터를 feeding 해주는 예제이다.
- 입력 데이터 행렬 x 는 Placeholder 타입으로 3×3 행렬이고, 여기에 곱하는 값 w 는 1×3 행렬이다.

예제코드

```
import tensorflow as tf
```

```
input_data = [ [1.,2.,3.],[1.,2.,3.],[2.,3.,4.] ] #3x3 matrix
```

```
x = tf.placeholder(dtype=tf.float32,shape=[None,3])
```

```
w = tf.Variable([ [2.],[2.],[2.] ], dtype = tf.float32) #3x1 matrix
```

```
y = tf.matmul(x,w)
```

```
sess = tf.Session()
```

```
init = tf.global_variables_initializer()
```

```
sess.run(init)
```

```
result = sess.run(y,feed_dict={x:input_data})
```

```
print(result)
```

2. 텐서 플로우에서 행렬의 표현

- 실행결과

```
In [26]: import tensorflow as tf

input_data = [ [1.,2.,3.],[1.,2.,3.],[2.,3.,4.] ] #3x3 matrix
x = tf.placeholder(dtype=tf.float32, shape=[None,3])
w = tf.Variable([ [2.],[2.],[2.] ], dtype = tf.float32) #3x1 matrix
y = tf.matmul(x,w)

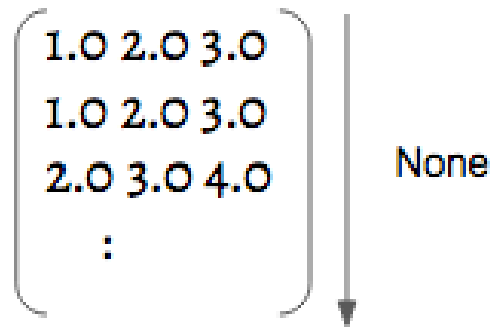
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
result = sess.run(y,feed_dict={x:input_data})

print(result)
```

```
[[ 12.]
 [ 12.]
 [ 18.]]
```

2. 텐서 플로우에서 행렬의 표현

- 이 예제에서 주의 깊게 봐야 할 부분은 placeholder x 를 정의하는 부분인데, $\text{shape}=[\text{None},3]$ 으로 정의했다 3x3 행렬이기 때문에, $\text{shape}=[3,3]$ 으로 지정해도 되지만 None 이란, 갯수를 알수 없음을 의미하는 것으로, 텐서플로우 머신러닝 학습에서 학습 데이터가 계속해서 들어오고 학습 때마다 데이터의 양이 다를 수 있기 때문에, 이를 지정하지 않고 None으로 해놓으면 들어오는 숫자 만큼에 맞춰서 저장을 한다.



2. 텐서 플로우에서 행렬의 표현

2-4. 브로드 캐스팅

- 텐서플로우 그리고 파이썬으로 행렬 프로그래밍을 하다보면 헷갈리는 개념이 브로드 캐스팅이라는 개념이 있다. 먼저 다음 코드를 보자

```
import tensorflow as tf

input_data = [ [1,1,1],[2,2,2] ]
x = tf.placeholder(dtype=tf.float32,shape=[2,3])
w = tf.Variable([[2],[2],[2]],dtype=tf.float32)
b = tf.Variable([4],dtype=tf.float32)
y = tf.matmul(x,w)+b

print (x.get_shape())
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
result = sess.run(y,feed_dict={x:input_data})

print(result)
```

2. 텐서플로에서 행렬의 표현

2-4. 브로드 캐스팅

```
In [29]: import tensorflow as tf

input_data = [
    [1, 1, 1], [2, 2, 2]
]
x = tf.placeholder(dtype=tf.float32, shape=[2, 3])
w = tf.Variable([[2], [2], [2]], dtype=tf.float32)
b = tf.Variable([4], dtype=tf.float32)
y = tf.matmul(x, w) + b

print (x.get_shape())
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
result = sess.run(y, feed_dict={x: input_data})

print(result)
```

(2, 3)
[[10.]
 [16.]

2. 텐서플로에서 행렬의 표현

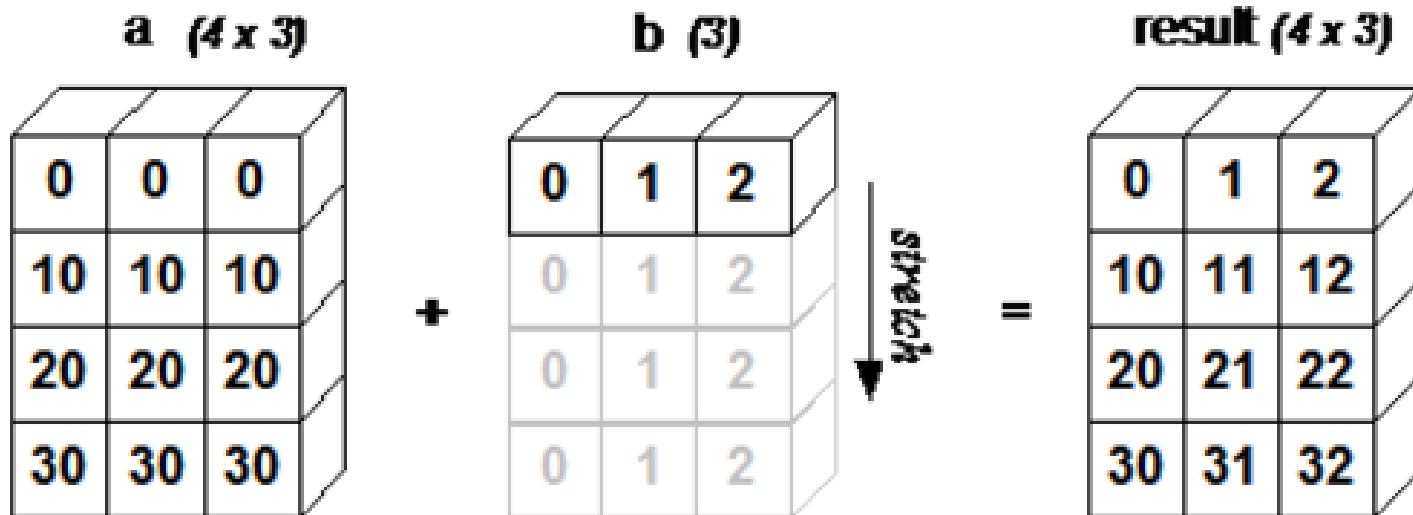
- 행렬 x 는 2×3 행렬이고 w 는 3×1 행렬이다. $x \cdot w$ 를 하면 2×1 행렬이 나온다.
- 문제는 $+b$ 인데, b 는 1×1 행렬이다. 행렬의 덧셈과 뺄셈은 차원이 맞아야 하는데, 이 경우 더하고자 하는 대상은 2×1 , 더하려는 b 는 1×1 로 행렬의 차원이 다르다. 그런데 어떻게 덧셈이 될까?
- 이 개념이 브로드 캐스팅이라는 개념인데, 위에서는 1×1 인 b 행렬을 더하는 대상에 맞게 2×1 행렬로 자동으로 늘려서 (stretch) 계산한다.
- 브로드 캐스팅은 행렬 연산 (덧셈, 뺄셈, 곱셈)에서 차원이 맞지 않을때, 행렬을 자동으로 늘려줘서(Stretch) 차원을 맞춰주는 개념으로 늘리는 것은 가능하지만 줄이는 것은 불가능하다.
- 브로드 캐스팅 개념은

<http://scipy.github.io/old-wiki/pages/EricksBroadcastingDoc>

에 잘 설명되어 있으니 참고하기 바란다. (아래 그림은 앞의 링크를 참조하였다.)

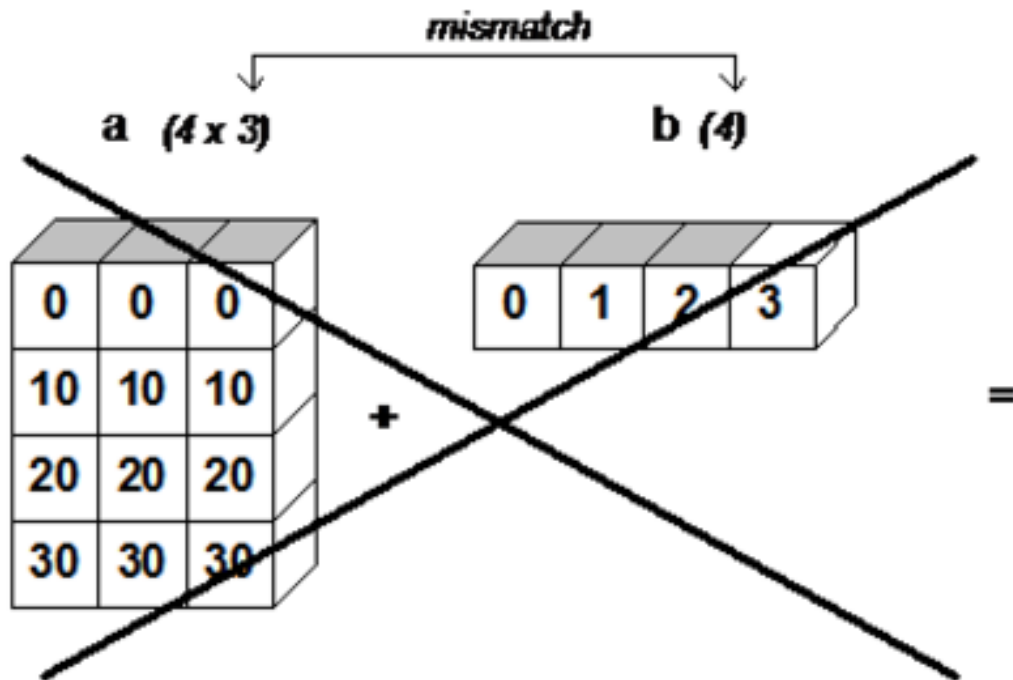
2. 텐서 플로우에서 행렬의 표현

- 아래는 4x3 행렬 a와 1x3 행렬 b를 더하는 연산인데, 차원이 맞지 않기 때문에, 행렬 b의 열을 늘려서 1x3 → 4x3 으로 맞춰서 연산한 예이다.



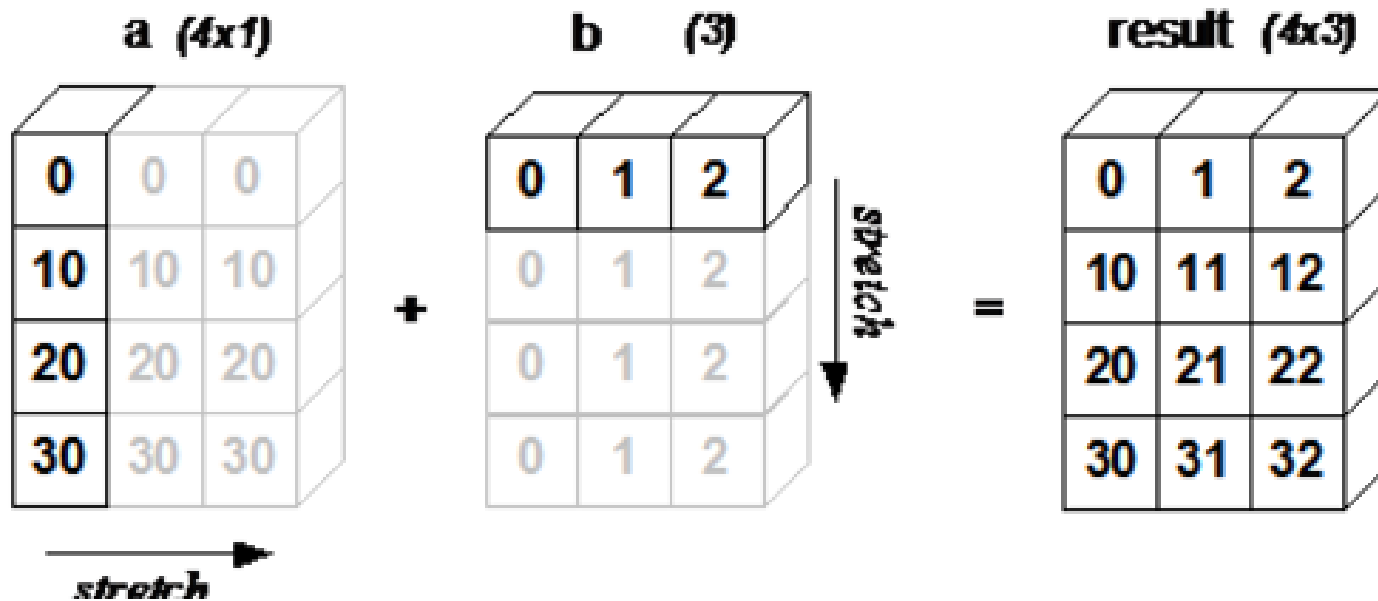
2. 텐서 플로우에서 행렬의 표현

- 만약에 행렬 b 가 아래 그림과 같이 1×4 일 경우에는 열을 $4 \rightarrow 3$ 으로 줄이고, 세로 행을 $1 \rightarrow 4$ 로 늘려야 하는데, 앞에서 언급한바와 같이, 브로드 캐스팅은 행이나 열을 줄이는 것은 불가능하다.



2. 텐서 플로우에서 행렬의 표현

- 다음은 양쪽 행렬을 둘다 늘린 케이스 이다.
- 4x1 행렬 a와 1x3 행렬 b를 더하면 양쪽을 다 수용할 수 있는 큰 차원인 4x3 행렬로 변환하여 덧셈을 수행한다.



3. 텐서플로우 행렬 차원 용어

- 텐서플로우에서는 행렬을 차원에 따라서 다음과 같이 호칭한다.
- 행렬이 아닌 숫자나 상수는 Scalar, 1차원 행렬을 Vector, 2차원 행렬을 Matrix, 3차원 행렬을 3-Tensor 또는 cube, 그리고 이 이상의 다차원 행렬을 N-Tensor라고 한다.

Rank	Math entity	Python example
0	Scalar (magnitude only)	<code>s = 483</code>
1	Vector (magnitude and direction)	<code>v = [1.1, 2.2, 3.3]</code>
2	Matrix (table of numbers)	<code>m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]</code>
3	3-Tensor (cube of numbers)	<code>t = [[[2], [4], [6]], [[8], [10], [12]], [[14], [16], [18]]]</code>
n	n-Tensor (you get the idea)	<code>....</code>

- 그리고 행렬의 차원을 Rank라고 부른다. scalar는 Rank가 0, Vector는 Rank가 1, Matrix는 Rank가 2가 된다.

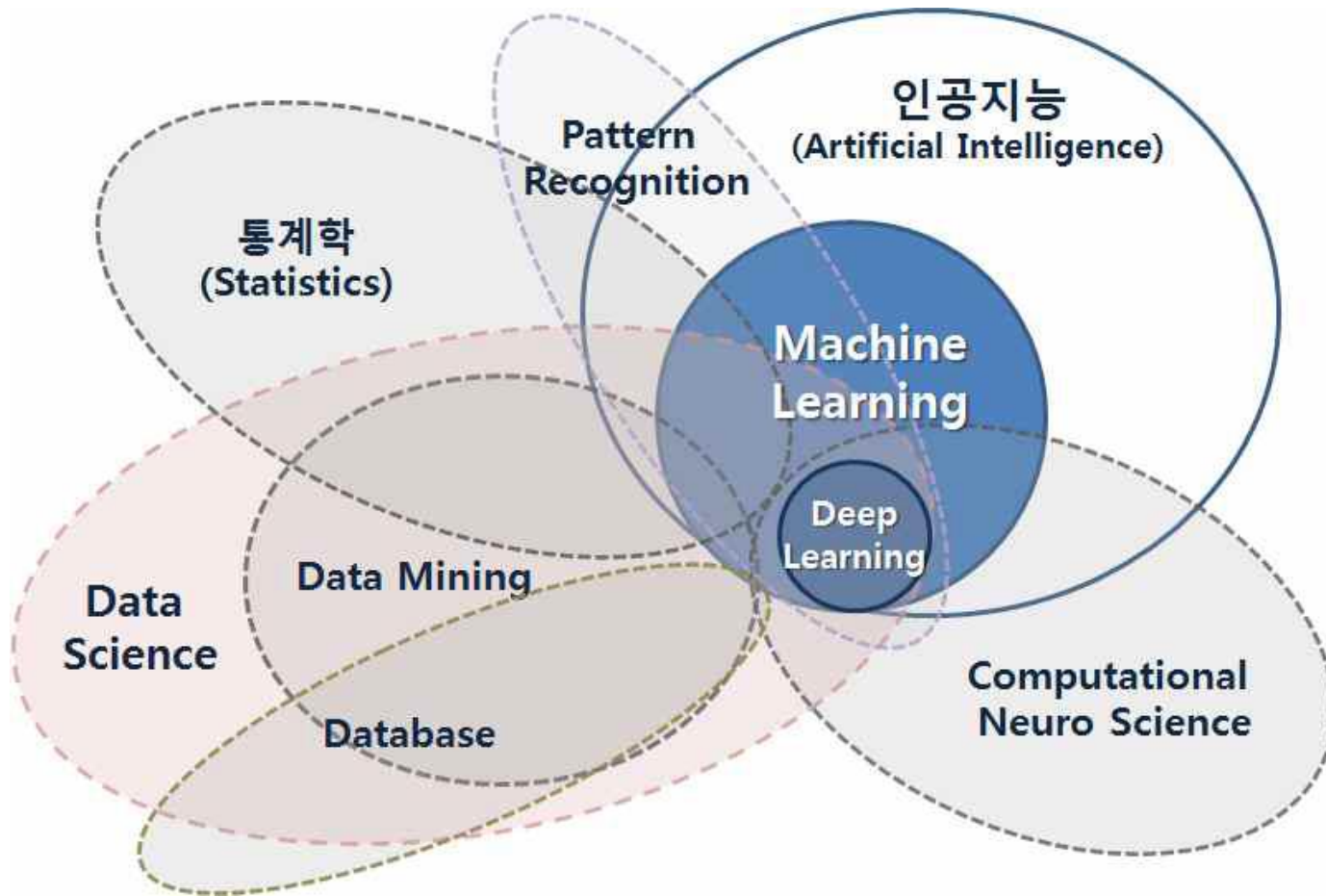
Tensorflow & Keras

1-2. 선형회귀분석

- Tensorflow 환경설정
- 선형회귀분석

머신러닝

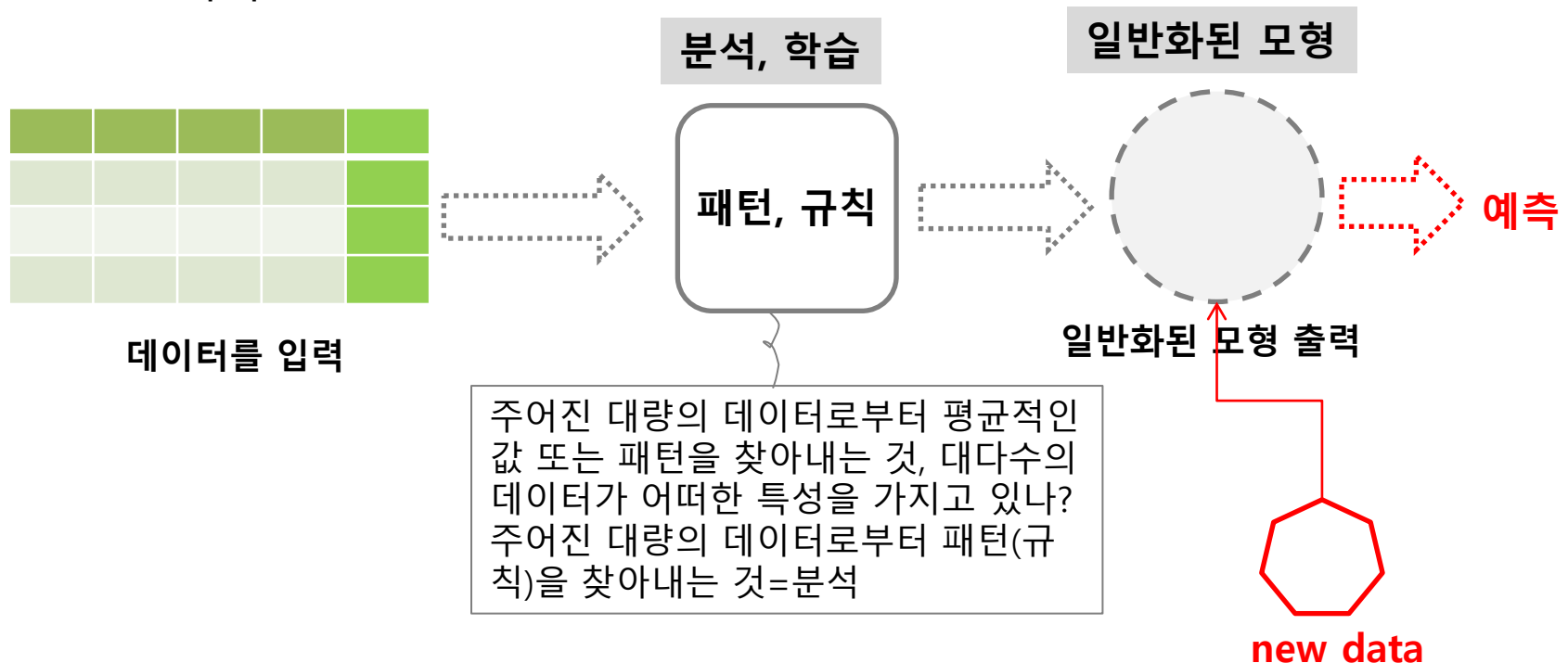
- 인공지능과 머신러닝



머신러닝

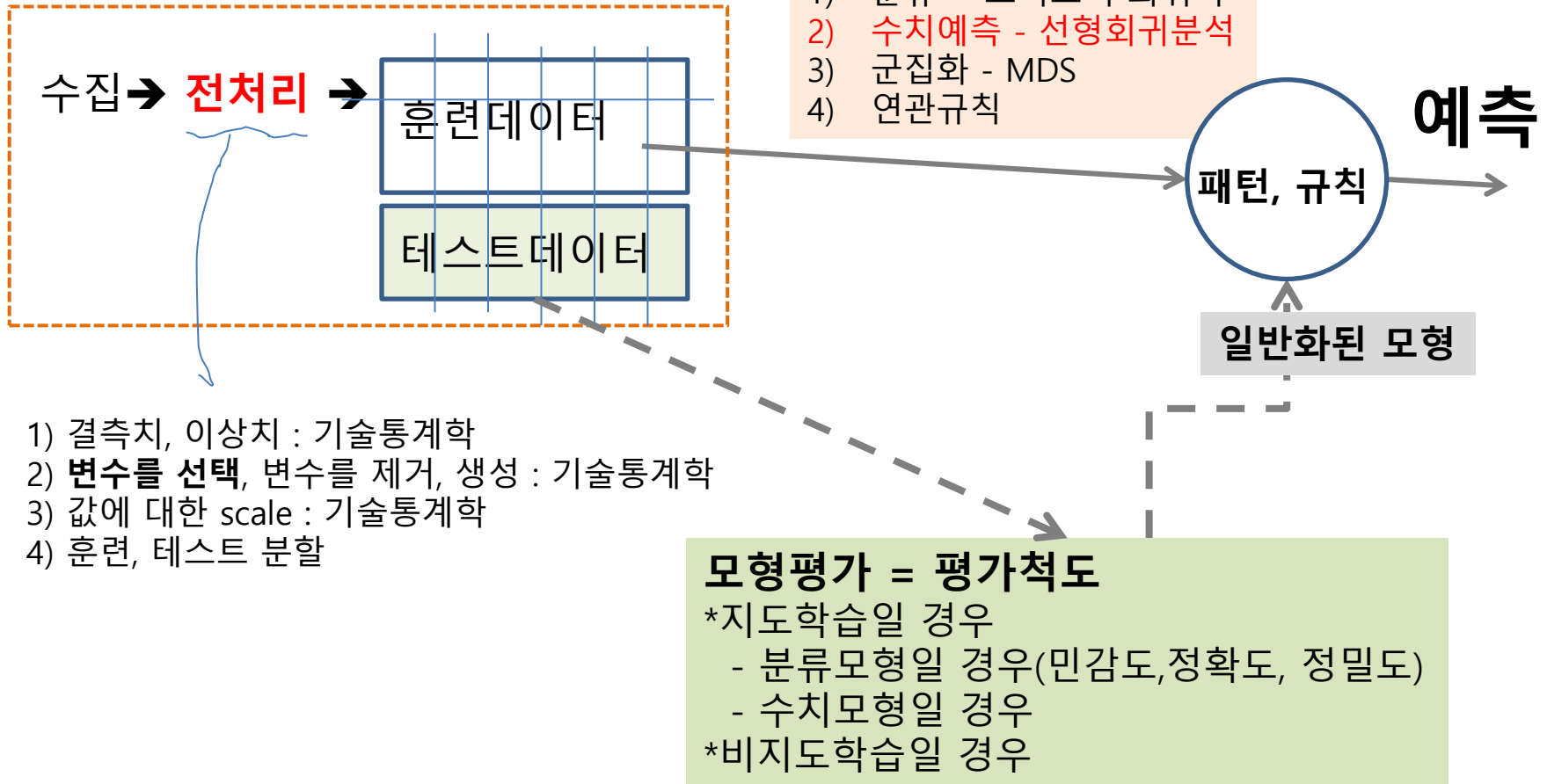
머신러닝의 개념

- 머신러닝(Machine Learning)은 컴퓨터 과학의 영역에 속하는 인공지능의 한 분야로서, 컴퓨터 프로그램이 어떤 것에 대한 학습을 통해 기존의 모델이나 결과물을 개선하거나 (자동으로) 예측 하거나 구축하는 과정을 의미



머신러닝 & 알고리즘

머신러닝



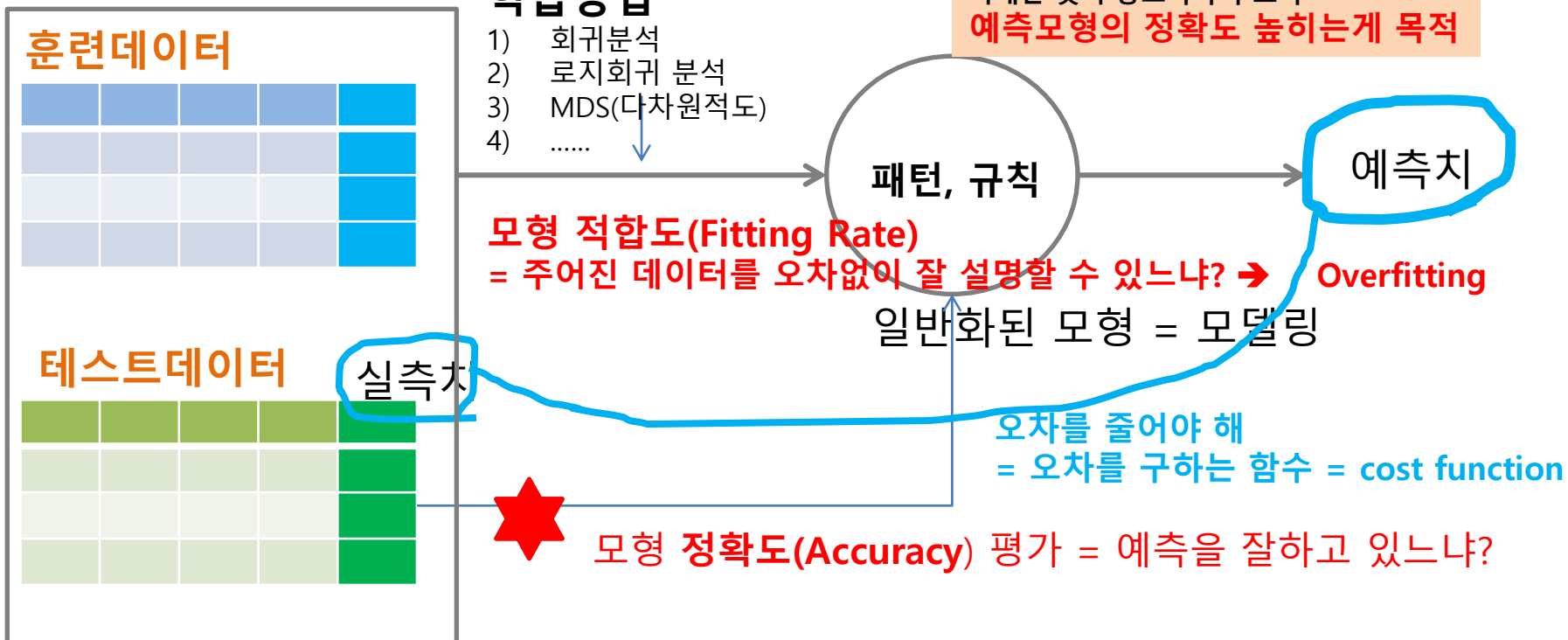
학습모형 평가

• 학습모형의 평가

대량의 주어진 데이터의 패턴, 규칙 찾아(fitting=학습)내어 → **예측**

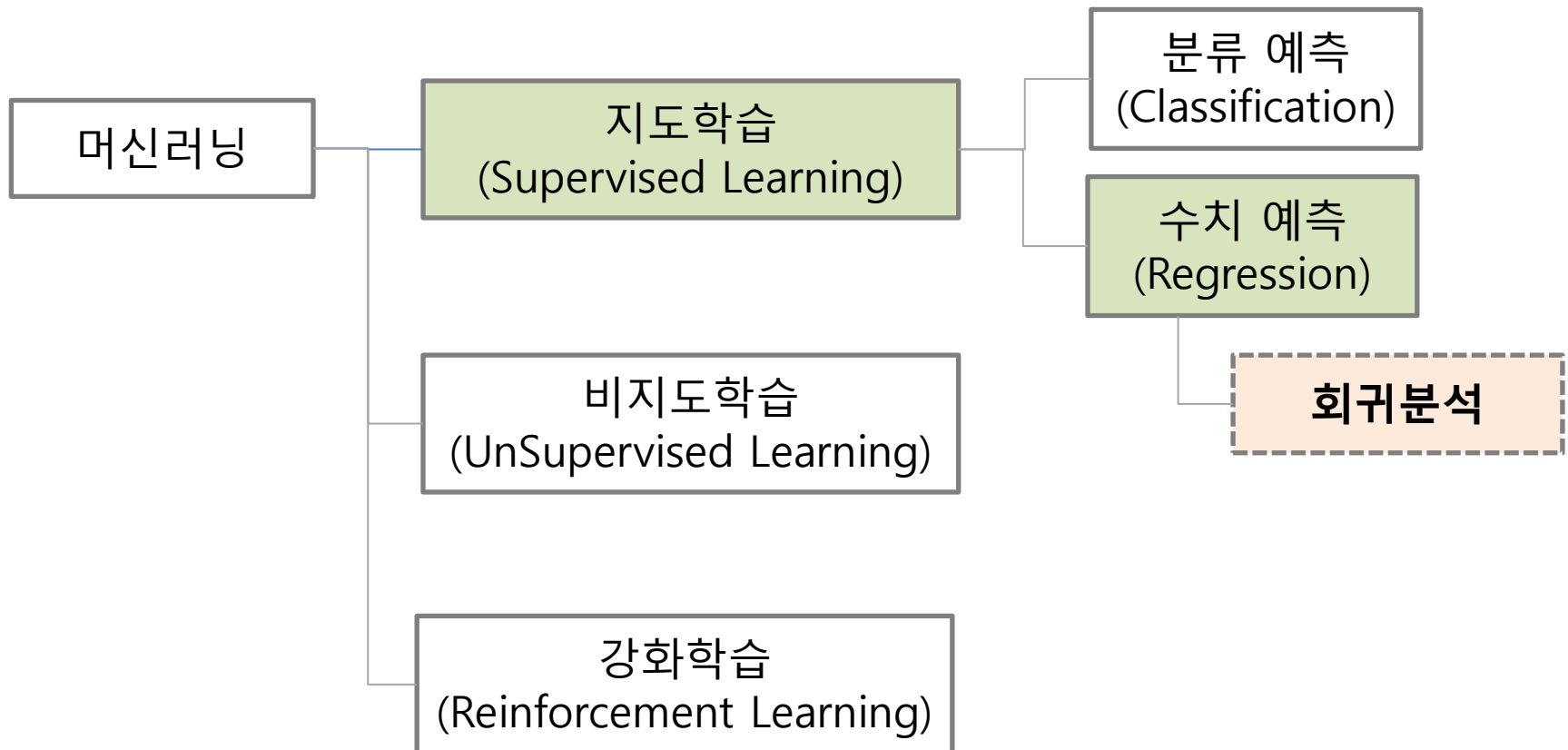
대략적으로 ~하다=평균 = **일반화**

• 지도 학습과정



머신러닝 학습방법의 종류

- 머신러닝 학습 방법



회귀(Regression) 분석

1. 회귀분석과 예측
2. 대수학(선형대수학)과 회귀분석
3. 회귀분석의 개념 및 용어 정리
4. 단순 선형 회귀 분석 및 회귀종류
5. R을 활용한 회귀분석 실습
6. 회귀분석을 위한 가정
7. 선형회귀 결과 추출
 1. 적합값 fitted value
 2. 잔차 residuals
 3. 회귀계수의 신뢰구간 confint
 4. 잔차 제곱합(최소 제곱법)
8. 예측과 신뢰구간
9. 모델평가
 1. 설명변수 평가
 2. 결정계수와 F통계량
10. 분산분석 및 모델 간의 비교
 - anova 분석
11. 모델 진단 그래프
 - Residuals vs Fitted
 - Normal Q-Q
 - Scale_Location
 - Residuals vs Leverage
 - Cook's distance
12. 시각화
13. 다중선형회귀
14. 회귀분석과 이상치판별
15. 머신러닝에서의 회귀분석
 - 경사하강법, cost function
 - 과소적합, 과대적합
 - Ridge, Lasso, Elastic Net
16. 데이터 공학
 - 회귀식을 위한 자료 변환

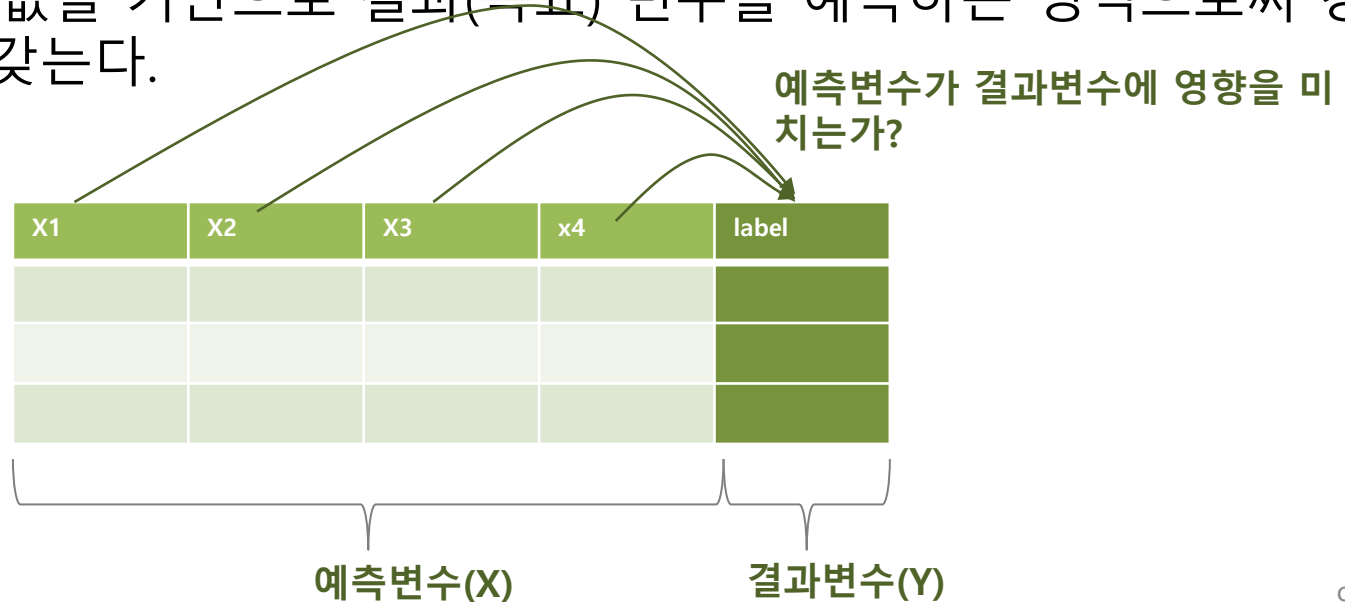
회귀(Regression) 와 예측

- 통계학의 일반적인 목표?

- 변수 X (혹은 $X_1, X_2, X_3, \dots, X_p$)가 변수 Y 와 관련이 있는가?
 - 이를 이용해 Y 를 예측할 수 있는가?
- 에 대해 답을 찾는 것을 목표로 한다.

- 통계학과 데이터 과학(머신러닝)이 서로 강하게 연결

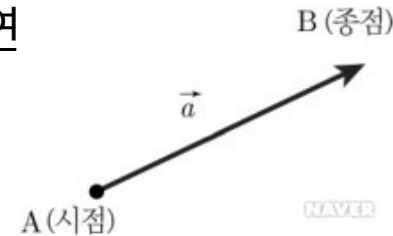
- '예측' 변수 값을 기반으로 결과(목표) 변수를 예측하는 영역으로써 강한 연결점을 갖는다.



회귀(Regression) 분석

- 대수학(代數學, algebra)

- 대수학이란 수학의 한 분야,
- 수 대신 문자를 쓰거나, 수학법칙을 간명하게 나타내는 것
- 기호와 방정식을 이용해 문제를 해결
- 대수학의 범위 : 선형대수학(벡터공간, 선형변환을 연구), 현대대수학(추상대수학 : 군, 환, 체 등의 대수적 구조를 연
- 벡터 : 크기와 방향을 가지는 양



- 선형 대수학

- 사회의 복잡한 현상을 선형화 과정을 거쳐 선형연립방정식이라는 단순한 형태의 수학문제로 바꿔 해결하려는데 결정적인 역할을 한다.

$$y = w * x + b$$

회귀(Regression) 분석

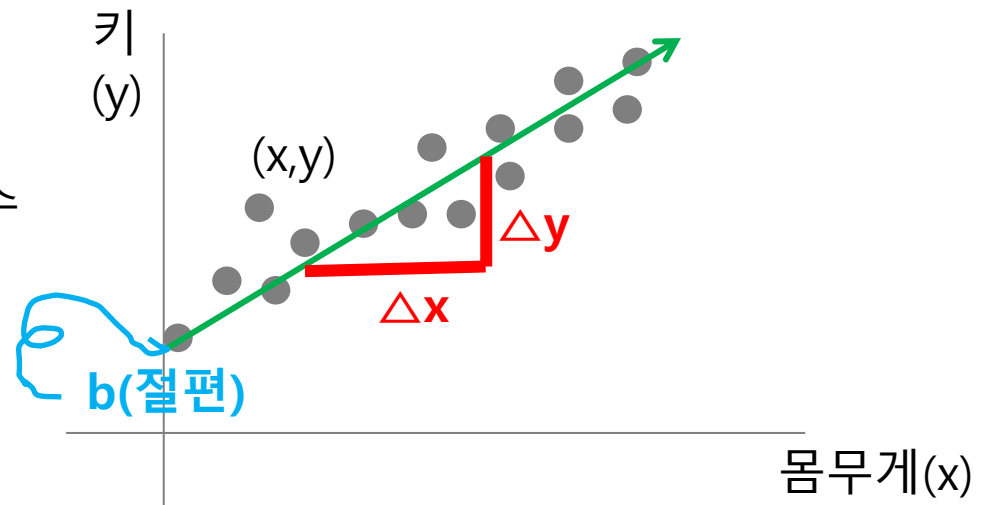
- 회귀분석이란

- 회귀분석은 분석 결과를 회귀식으로 표현해 요약 정리하는 것을 말한다.
- 이때 회귀식은 독립변수와 종속변수의 개수나 형태에 따라 다르지만, 가장 기본적인 회귀식은 **선형회귀식**을 사용한다.

$$\underline{y} = \underline{w} * \underline{x} + b$$

종속변수 영향을 주는 변수

예) 키 = 몸무게



$$w(\text{기울기}) = \frac{\Delta y}{\Delta x}$$

회귀(Regression) 분석

• 용어정리

- ① **응답변수(반응변수)** response variable : 예측하고자 하는 변수(유의어 : 종속변수, 변수 Y , 목표, 출력)
- ② **독립변수** independent variable : 응답치를 예측하기 위해 사용되는 변수(유의어:변수 X , 피처, 속성)
- ③ **레코드 record** : 한 특정 경우에 대한 입력과 출력을 담고 있는 벡터 (유의어:행, 사건, 예시 instance, 예제 example)
- ④ **절편 intercept** : 회귀직선의 절편. 즉, $X=0$ 일 때 예측값(유의어: b_0, β_0)
- ⑤ **회귀계수** regression coefficient : 회귀직선의 기울기 (유의어:기울기 slope, b_1, β_1 , 모수 추정값, 가중치)
- ⑥ **적합값** fitted value : 회귀식으로부터 얻은 추정치 \hat{Y}_i (유의어: 예측값)
- ⑦ **잔차** residual value : 관측값과 적합값의 차이(유의어:오차)
- ⑧ **최소제곱** least square : 잔차의 제곱합을 최소화하여 회귀를 피팅하는 방법(유의어 : 보통최소제곱)

회귀(Regression) 분석

- 회귀식의 다양한 형태

- ① 단순선형회귀식

$$y = w * x + b$$

기울기 절편(상수)

- ② 다중회귀식

$$y = w_1 * x_1 + w_2 * x_2 \dots + b$$

추가 종가 거래량.....

- ③ 다항회귀식 polynomial regression equation

$$y = w_1 * x_1 + w_2 * x_1^2 + b$$

회귀(Regression) 분석의 파라미터

- Cost Function

- 최적의 W 와 b 를 탐색하는데 드는 비용

- 우리가 세운 가설은 $H(x) = Wx + b$. (W, b) (Cost function), $H(x) - y$ 이다. 예측 값에서 실제 값을 빼주면 된다. 하지만 단순히 빼거나 더 할 경우 음수가 나올 수가 있어서 계산이 복잡해 질 수 있으므로 $(H(x) - y)^2$ 제곱을 해서 양수로 만들어 준다. 제곱을 하는 이유는 아래와 같다. 빨셈을 하면 데이터의 위치에 따라 음수와 양수가 섞인다. 계산하기가 피곤해진다.
- 절대 값을 취할 수도 있지만 제곱을 하는 방법도 있다. (음수 X 음수), (양수 X 양수) 모드 양수가 나오기 때문이다.
- 제곱을 할 경우 멀리 있는 값이 더 큰 값이 나온다. 따라서 멀리 있는 데이터에 벌점(Penalty)을 줄 수 있다.
- 다음 n 으로 나눈다. 데이터가 많아지면 데이터가 너무 커질 수 있는데, 값을 평균으로 만들면 계산하기가 편해진다.

2. 선형회귀(Linear Regression) 분석

- Cost Function

우리는 원본 데이터로부터 아래와 같은 식을 만들 수 있다.

$$\frac{(H(x^{(1)}) - y^{(1)})^2 + (H(x^{(2)}) - y^{(2)})^2 + (H(x^{(3)}) - y^{(3)})^2 + (H(x^{(4)}) - y^{(4)})^2 + (H(x^{(5)}) - y^{(5)})^2}{5}$$

이 식을 일반화 하면 $cost = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y(i))^2$ 이 된다.

이렇게 해서 W, b 에 대한 코스트 함수

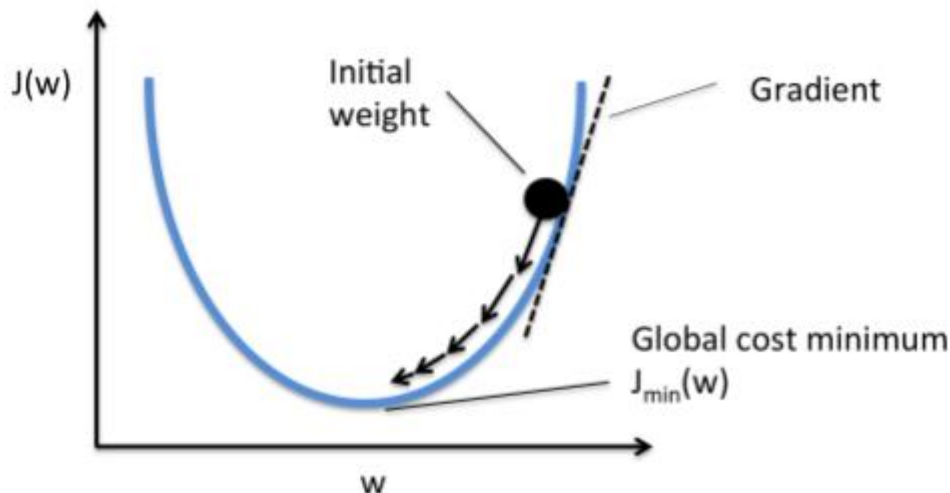
$$cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y(i))^2$$

가 만들어진다. 여기에서 거리를 가장 작게 만들어주는 W 와 b 를 구하면 된다.
이 과정이 선형회귀에서의 학습이다.

2. 선형회귀(Linear Regression) 분석

- Gradient descent

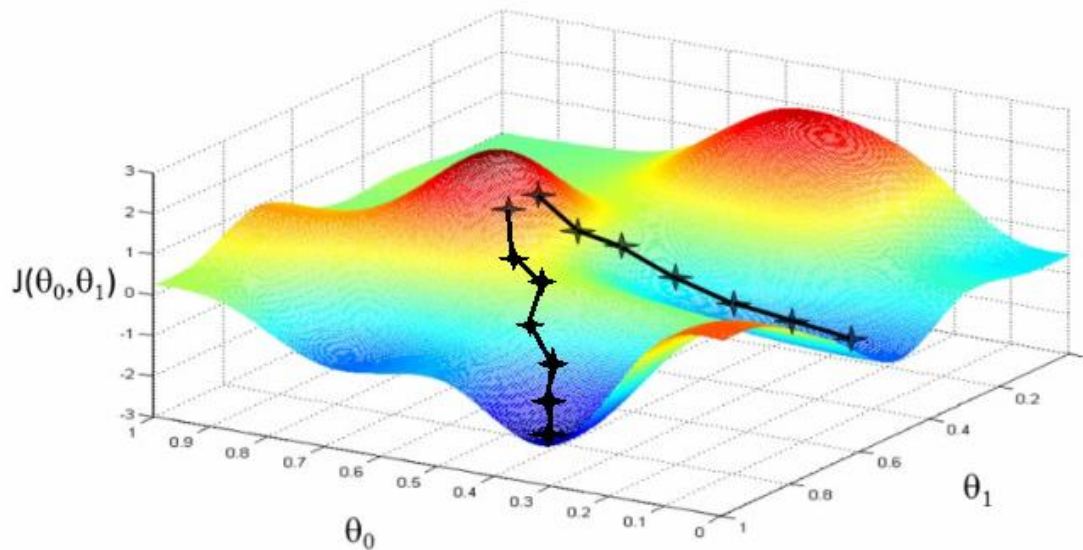
머신러닝은 데이터를 학습 하면서, 최적화된 값을 찾는 일련의 과정이다. 최적화된 값을 찾기 위해서는 오차를 계속 줄여가기 위한 어떤 방법을 개발해야 한다. 이것은 한치앞이 안보이는 울창한 밀림에서 계곡으로 가야 한다고 가정해 보자. 앞이 보이지 않기 때문에 계곡이 어디있는지 알 수 없지만 현재 위치에서 경사가 아래로 가파른쪽으로 내려가다 보면 결국 계곡에 다다르게 될 것이다. 이렇게 극소점을 찾기 위해 이동해 가는 방법을 경사하강법(Gradient descent) 라고 부른다. 이걸 수학적으로 풀어보자



2. 선형회귀(Linear Regression) 분석

- **Gradient descent**

우리는 검은색 공을 그래프의 밑바닥으로 옮겨야 한다. 공이 밑으로 가고 있는지는 공이 위치한 지점에서의 기울기(Gradient)가 이전 기울기 보다 낮아지는지를 검사하는 것으로 판단 할 수 있다. 이렇게 기울기가 줄어드는 방향으로 움직이다가 0이 되는 지점을 찾으면 여기가 그래프의 밑바닥이 된다. 아래 그림을 보자.



2. 선형회귀(Linear Regression) 분석

- **Tensorflow 경사하강법 구현 Gradient descent**

경사하강법을 이용해서 어떻게 원하는 곳을 찾아가는지를 묘사하고 있다. 이제 경사하강법을 알고리즘으로 나타내보자. 변화율이므로 미분 방정식으로 표현된다.

$$\text{repeat until convergence} \{ \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \}$$

α 는 learning rate로 얼마나 빠르게 내려올지 결정하기 위해서 사용한다.은 편미분(partial derivative)하라는 의미다. 이 과정을 수렴할 때까지 반복 진행하면 된다. 미분이라고 해서 겁먹을 필요는 없다. Tensorflow 라이브러리에서 제공하는 함수를 호출하기만 하면 된다

```
a = tf.Variable(0.1)
optimizer = tf.train.GradientDescentOptimizer(a)
train = optimizer.minimize(cost)
```

Tensorflow를 활용한 선형회귀분석

- 먼저 필요한 파이썬 모듈을 import 한다.

```
import os
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import math

%matplotlib inline
```

TF 1.15

- tensorflow는 tensorflow Deep Learning Framework 모듈을 사용하기 위해 import 한다.
- numpy는 다차원 배열과 각종 산술 연산을 담당하는 Python 라이브러리이다.
- matplotlib는 그래프를 통해 시각화할 때 사용하는 라이브러리이며,

3. Python을 이용한 선형회귀분석

- [SAT and College GPA](#) 미국 105 : 대학 및 고등학교의 GPA(Grade Point Average - 평균학점)과 컴퓨터와 수학에 대한 SAT 점수를 포함하고 있다. 이 정보를 이용해서 고등학교에서의 GPA로 대학에서의 GPA를 예측해보자.
- <https://docs.google.com/spreadsheets/d/1WE0fqNndH3mHNiUpmQjyeEVecfUvv9OyCRYypYv4ujs/pubhtml?gid=1302996453&single=true>

3. Python을 이용

- scipy & 선형회귀

```
##텐서플로우와 회귀분석 - GPA문제적용
```

```
#!/usr/bin/python
```

```
from scipy import stats, polyval
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
# 파일을 줄 단위로 읽은 후 공백을 기준으로 split 한 후,  
# 고등학교와 대학교의 GPA를 배열에 넣는다.
```

```
f = open("sat.txt", "r") #헤더정보를제외
```

```
num = 0
```

```
h_GPA = []
```

```
u_GPA = []
```

```
while True:
```

```
    line = f.readline()
```

```
    if not line: break
```

```
    items = line.split()
```

```
    #h_GPA.append(items[0])
```

```
    #u_GPA.append(items[4])
```

```
    h_GPA.append(float(items[0]))#h_GPA(고등학교GPA)
```

```
    u_GPA.append(float(items[4])) #univ_GPA(대학교GPA)
```

```
f.close()
```

```
slope, intercept, r, p, std = stats.linregress(h_GPA,u_GPA)
```

```
ry = polyval([slope, intercept], h_GPA)
```

```
print(slope, intercept, r, p, std)
```

3. Python을 이용한 선형회귀분석

- **scipy & 선형회귀**

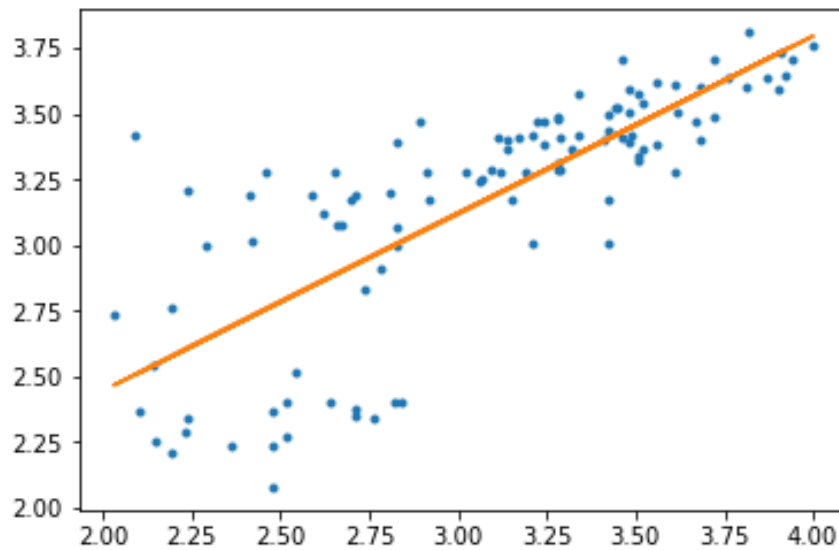
그래프를 그린다.

```
plt.plot(h_GPA, u_GPA, '.') #x축,y축:실측치그래프
```

```
plt.plot(h_GPA, np.array(h_GPA) * slope + intercept ) #모형그래프
```

```
plt.show()
```

0.674829903448 1.09682328179 0.779563120549 1.17561260288e-22 0.0534223818164



4. Tensorflow를 활용한 선형회귀분석

- 계산 그래프 관리
 - Tip) 기본 그래프를 초기화하기 `tf.reset_default_graph()`
 - 노드를 만들면 자동으로 기본 계산 그래프에 추가된다.

```
>>> x1 = tf.Variable(1)
>>> x1.graph is tf.get_default_graph()
True
```

기본 계산 그래프

```
>>> graph = tf.Graph()
>>> with graph.as_default():
...     x2 = tf.Variable(2)
...
>>> x2.graph is graph
True
>>> x2.graph is tf.get_default_graph()
False
```

새로운 Graph객체를 만들어
with 블록 안에서 임시로 이를
기본 계산 그래픽으로 사용할 수
있다.

4.Tensorflow를 활용한 선형회귀분석

- 회귀분석을 위한 데이터를 읽어온다. 다음의 텍스트파일은 (x,y) 좌표 총 100개의 행을 구성되어 있다.

```
simple_regression = np.loadtxt('simple_regression.txt')  
x_data = simple_regression[:,1]  
y_data = simple_regression[:,0]
```

- Dataset의 x와 y좌표를 읽어와 배열로 저장한다.
- numpy에서 제공하는 loadtxt 함수를 이용하면 해당 테스트 파일을 자동으로 파싱해서 배열을 생성해준다.
- 위의 예제는 (x,y)좌표 총 100개 데이터 셋이기 때문에 [100,2] 모양의 배열로 읽어오게 된다.
- 읽어온 배열을 x데이터와 y 데이터로 따로 나누어 준다.
- simple_regression[:,1]에서 :부분은 해당 축의 전체 데이터를 뜻하며, 1은 (x,y) 두 점 중에서 y만 나타내게 된다.
- 따라서 simple_regression[:,0], simple_regression[:,1] 이렇게 나눠주면 x좌표, y좌표를 각각 분리해 저장할 수 있다.

4.Tensorflow를 활용한 선형회귀분석

- 'simple_regression.txt' 파일

Simple linear regression

```
In [7]: simple_regression = np.loadtxt('simple_regression.txt')
x_data = simple_regression[:,1]
y_data = simple_regression[:,0]

simple_regression
```

```
Out [7]: array([[50.496375,  7.312312],
 [69.017336,  9.94901 ],
 [ 6.504676,  0.402668],
 [47.202416,  7.372892],
 [40.42739 ,  5.413481],
 [15.258608,  1.689084],
 [32.444642,  4.89831 ],
 [51.332003,  7.443845],
 [30.384774,  4.409411],
 [11.333958,  2.317807],
 [14.756302,  2.439615],
 [67.13818 ,  9.791725],
 [62.027474,  9.515288]
```

4.Tensorflow를 활용한 선형회귀분석

- Graph를 생성한다.

```
with tf.Graph().as_default() as simple_regression:
    X = tf.placeholder(tf.float32, [None], name='X')
    Y = tf.placeholder(tf.float32, [None], name='Y')
    lr = tf.constant(1e-3, tf.float32)
    W = tf.get_variable("W", dtype=tf.float32, initializer=tf.constant(1.,
tf.float32))
    b = tf.get_variable("b", dtype=tf.float32, initializer=tf.constant(1.,
tf.float32))

    h = W*X + b
    cost = tf.reduce_mean(tf.square(tf.subtract(h, Y)))
    train =
tf.train.GradientDescentOptimizer(learning_rate=lr).minimize(cost)
```

4.Tensorflow를 활용한 선형회귀분석

- Tensorflow 코드를 쓰면서 바로 실행하는 것이 아니라, 하고자 하는 연산을 그래프로 작성한 후, session 객체를 통해 실행시킨다(v1.X)

- with tf.Graph().as_default() as simple_regression:**

와 같이 simple_resgression이라는 이름을 가진 그래프를 만들어준다. 이제 아래에 쓰는 텐서 및 오퍼레이터는 모두 simple_regression이라는 이름의 그래프 안에 들어가게 된다.

X = tf.placeholder(tf.float32, [None], name='X')

Y = tf.placeholder(tf.float32, [None], name='Y')

- 먼저 그래프에 Placeholder2개를 추가한다. Placeholder는 외부 데이터를 받아와서 저장하는 변수로.tf.placeholder(자료형, 텐서모양, 이름) 형태로 구성되어 있으며, 입력데이터가 실수이므로 tf.float32, [None] 은 해당 데이터의 입력길이만큼 가변적으로 받아오는 모양(shape)이다. 지금 예제에서는 x_data가 총 100개이므로 [100] 이렇게 shape을 지정해주어도 상관없으나, 보통 batch size가 정해져 있지 않으므로 일반적으로 [None]이라고 지정한다. 마지막으로 name은 TensorBoard라는 시각화 툴에서 그래프를 나타낼때 표시될 이름이다.

4.Tensorflow를 활용한 선형회귀분석

```
lr = tf.constant(1e-3, tf.float32)
```

- 입력 데이터를 학습시킬 때 사용한 learning rate를 상수로 정의해준다.
- `tf.constant(값, 자료형)` 형태를 가지며, `1e-3`은 10^{-3} 이다. 마찬가지로 실수이므로 `tf.float32`자료형을 쓴다.

```
W = tf.get_variable("W", dtype=tf.float32,  
                    initializer=tf.constant(1., tf.float32))  
b = tf.get_variable("b", dtype=tf.float32,  
                    initializer=tf.constant(1., tf.float32))
```

- 선형회귀를 위해 만드는 가설함수(h)에 들어가는 파라미터들이다. 변수 1개 짜리 선형회귀이므로 가설 함수의 모양은 $h = W \cdot x + b$ 의 모양으로 가정한다.
- `tf.get_variable(이름, dtype=자료형, initializer=초기값)`이며, 마찬가지로 실수형이므로 `dtype = tf.float32`이며 `initializer`는 처음 학습을 시작하기 전에 변수를 초기화하는 과정에서 설정되는 값이다. Tensorflow에서는 여러가지 `initializer`를 제공하지만, 간단한 예제이므로 그냥 상수값 1로 초기화한다.

4.Tensorflow를 활용한 선형회귀분석

- 이제 선형회귀에 필요한 변수들은 모두 그래프에 추가되었다. 남은 것은 가설함수와 계산할 오차(Error) 그리고 최적화하는 방법이다.
- 먼저 가설함수부터 만들어보자.

$$\mathbf{h} = \mathbf{W} * \mathbf{x} + \mathbf{b}$$

- 입력데이터 X에 두 파라미터 W, b를 곱하고 더하여 가설함수를 계산한다.

$$\text{cost} = \text{tf.reduce_mean}(\text{tf.square}(\text{tf.subtract}(\mathbf{h}, \mathbf{Y})))$$

- 위의 cost가 계산할 오차이다. 가설함수의 h값과 실제 데이터의 Y값을 빼서 제곱한 값의 평균이다. 아래의 식을 텐서플로 코드로 표현한 것이다.

$$\frac{1}{N} \sum_i (h(x_i) - y_i)^2$$

$$\mathbf{h} = \mathbf{W} * \mathbf{X} + \mathbf{b}$$

$$\text{cost} = \text{tf.reduce_mean}(\text{tf.square}(\text{tf.subtract}(\mathbf{h}, \mathbf{Y})))$$

4.Tensorflow를 활용한 선형회귀분석

- 최적화

```
train = tf.train.GradientDescentOptimizer(learning_rate=lr).minimize(cost)
```

- tensorflow에서는 오차를 줄이는 알고리즘을 여러 개 제공하고 있다. (GradientDescent, ADAM, ADAGRAD, 등등..)
- `tf.train.GradientDescentOptimizer(learning_rate)` GradientDescent 알고리즘을 이용하여 최적화하는 함수이다. `learning_rate`를 파라미터로 받아서 이용하게 된다.
- Optimizer를 정의해주고 나면 무엇을 최적화 할 것인지를 알려줘야 한다. 뒤의 `minimize(cost)` 이 함수가 최적화 해주는 함수이다. 괄호 안에 줄여나갈 오차(loss)를 넣어주면 된다.

4. Tensorflow를 활용한 선형회귀분석

- 최적화(Optimization)

- 회귀분석의 목적은 손실함수의 값을 가능한 한 낮추는 매개변수를 찾는 것이다.
이는 곧 매개변수의 최적값을 찾는 문제이다.

- 최적화 방법

- ① 확률적 경사하강법(SGD)

- ② 모멘텀(Motmentum) : 물리식 활용

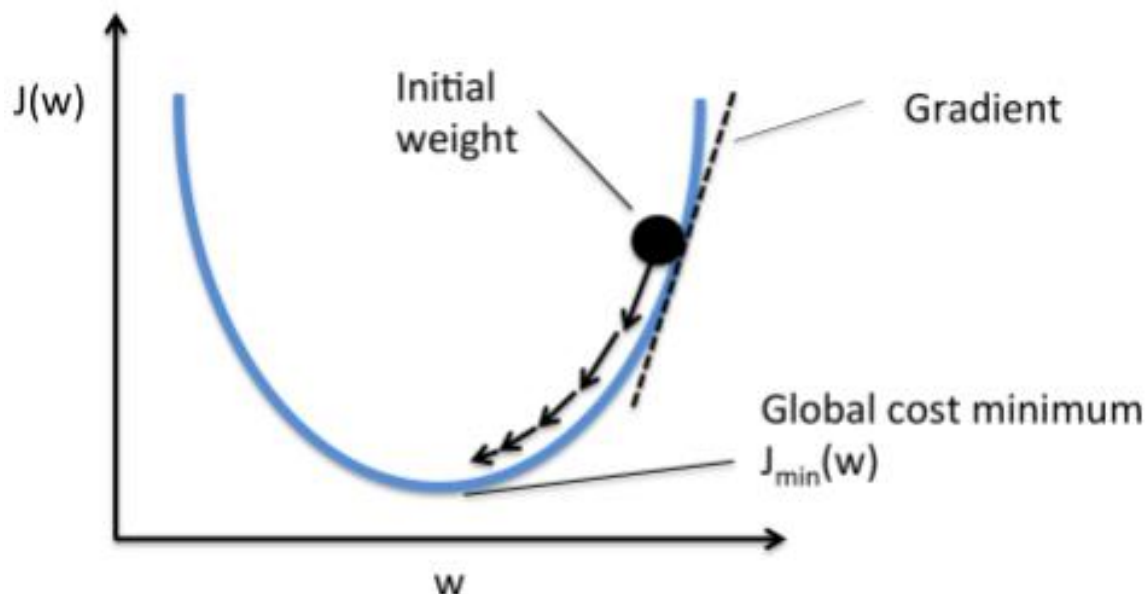
- ③ AdaGrad

- ④ Adam

4.Tensorflow를 활용한 선형회귀분석

- **확률적 경사하강법(SGD)**

머신러닝은 데이터를 학습 하면서, 최적화된 값을 찾는 일련의 과정이다. 최적화된 값을 찾기 위해서는 **오차를 계속 줄여가기 위한 어떤 방법**을 개발해야 한다. 이것은 한치앞이 안보이는 울창한 밀림에서 계곡으로 가야 한다고 가정해 보자. 앞이 보이지 않기 때문에 계곡이 어디있는지 알 수 없지만 현재 위치에서 경사가 아래로 가파른쪽으로 내려가다 보면 결국 계곡에 다다르게 될 것이다. 이렇게 극소점을 찾기 위해 이동해 가는 방법을 **경사하강법(Gradient descent)** 라고 부른다. 이걸 수학적으로 풀어보자



4.Tensorflow를 활용한 선형회귀분석

① 최적화 : 확률적 경사하강법(SGD)

-최적의 매개변수 값을 찾는 단서로 매개변수의 기울기(미분)을 이용했다. 매개변수의 기울기를 구해 기울어진 방향으로 매개변수 값을 갱신하는 일을 몇 번이고 반복해서 점점 최적의 값에 도달하도록 하는 기법을 사용했다. 이를 경사하강법이라한다.

• 확률적 경사하강법(SGD) 이해

색다른 모험가가 있다. 그는 전설에 나오는 세상에서 가장 깊고 낮은 골짜기를 찾아가려고 한다. 단, 지도를 볼 수 없고, 눈가리개를 해야 한다는 제약이 있다. 어떻게 '가장 깊은 골짜기'를 찾을 수 있을까?

- 이 어려운 상황에서 중요한 단서가 되는 것은 땅의 '기울기'이다. 그래서 지금 서있는 장소에서 가장 크게 기울어진 방향으로 가자고 하는 것이 SGD의 전략이다. 이 일을 반복하면 언젠가 깊은 곳에 도달할 수 있을지도 모른다는 전략이다.

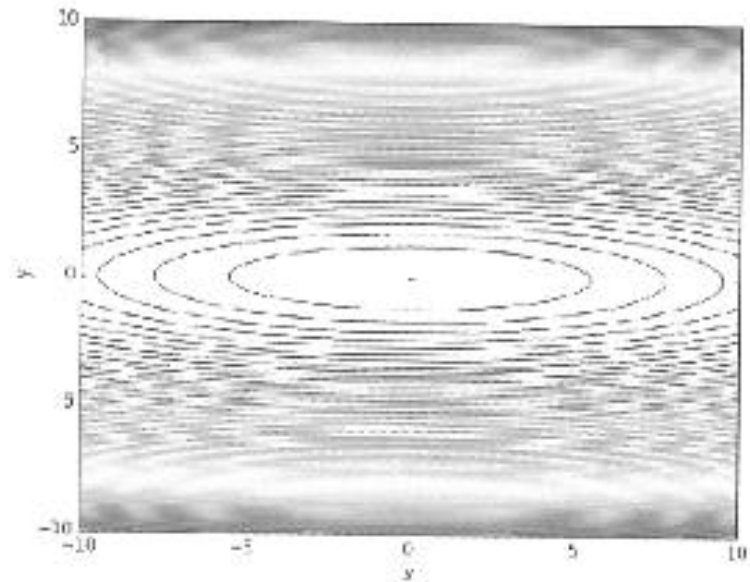
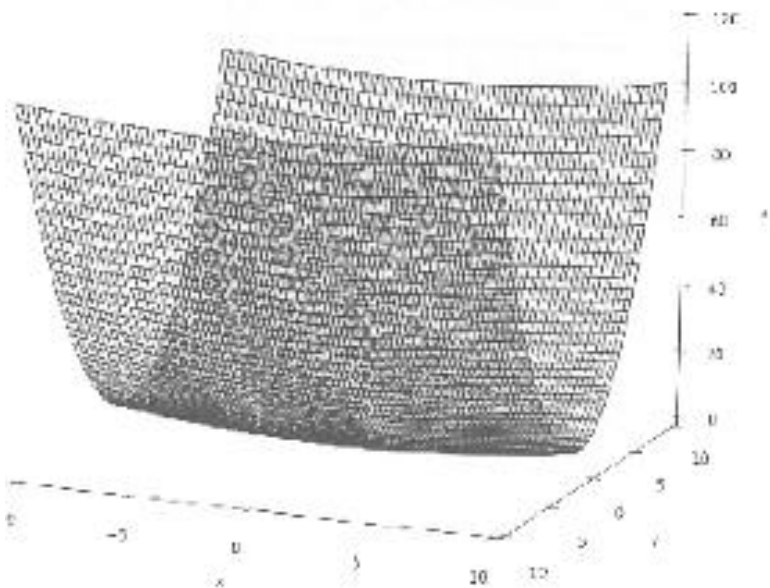
4. Tensorflow를 활용한 선형회귀분석

① 최적화 : 확률적 경사하강법(SGD)

- 확률적 경사하강법(SGD) 단점

- SGD는 단순하고 구현도 쉽지만, 문제에 따라서는 비효율적일 수 있다.

$f(x, y) = \frac{1}{20}x^2 + y^2$ 의 그래프(왼쪽)와 그 등고선(오른쪽)



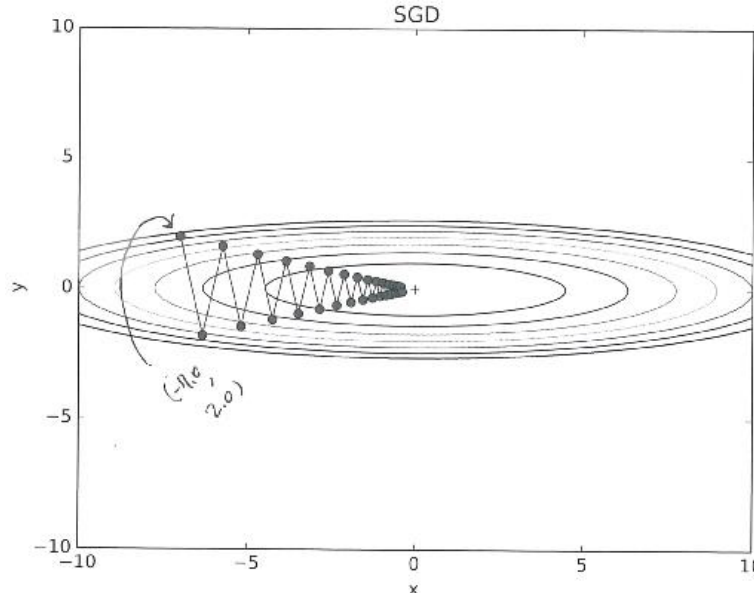
$f(x,y)$ 는 '밥그릇'을 x축 방향으로 늘인 듯한 모습

4. Tensorflow를 활용한 선형회귀분석

① 최적화 : 확률적 경사하강법(SGD)

•확률적 경사하강법(SGD) 단점

- SGD의 단점은 비등방성(anisotropy)함수(방향에 따라 성질, 즉, 여기서는 기울기가 달라지는 함수)에서는 탐색 경로가 비효율적이라는 것이다. 또한 SGD가 지그재그로 탐색하는 근본 원인은 기울어진 방향이 본래의 최솟값과 다른 방향을 가리켜서이다.



SGD에 의한 최적화 갱신 경로: 최솟값인(0,0)까지 지그재그로 이동하니 비효율적

4. Tensorflow를 활용한 선형회귀분석

② 최적화 : 모멘텀

- 모멘텀(Momentum)은 '운동량'이라는 의미로, 물리와 관련

$$\mathbf{v} \leftarrow \alpha \mathbf{v} - \eta \frac{\partial L}{\partial \mathbf{W}}$$
$$\mathbf{W} \leftarrow \mathbf{W} + \mathbf{v}$$

학습률

\mathbf{W} 에 대한 손실함수의 기울기

물리에서 속도와 같은 의미(velocity)
→ av 항은 물체가 아무런 힘을 받지 않을 때 서서히 하강시키는 역할을 함

\mathbf{W} 는 갱신할 가중치 매개변수

4. Tensorflow를 활용한 선형회귀분석

② 최적화 : 모멘텀

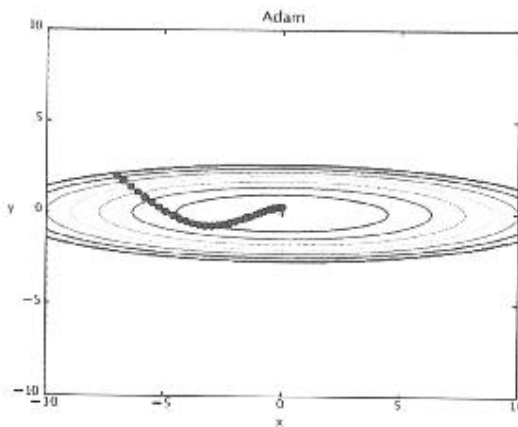
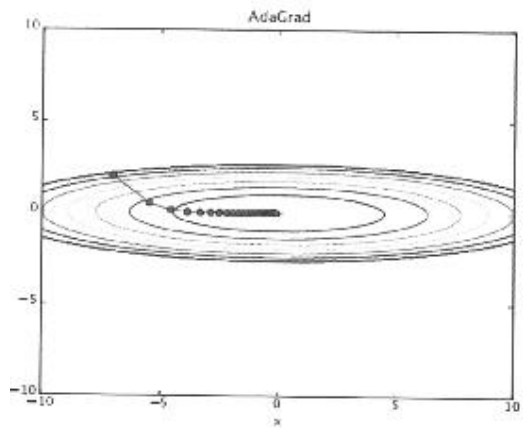
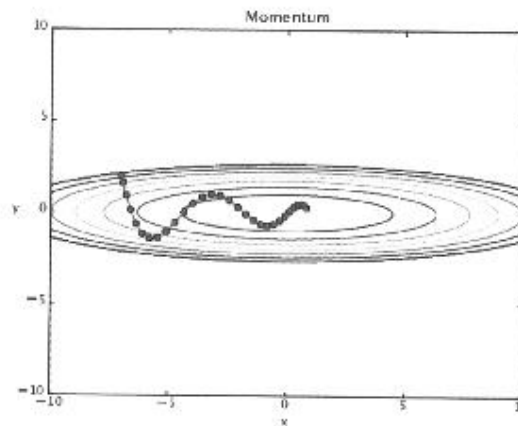
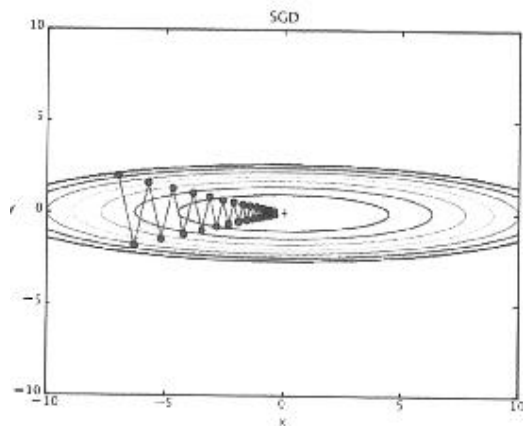
- 모멘텀(Momentum)은 '운동량'이라는 의미로, 물리와 관련
- 공이 그릇의 곡면(기울기)을 따라 구르듯 움직인다.



4. Tensorflow를 활용한 선형회귀분석

- 최적화모델

- SGD, 모멘텀, AdaGrad, Adam의 매개변수 갱신 방법



4. Tensorflow를 활용한 선형회귀분석

- Tensorflow 경사하강법 구현

-후진 모드 자동 미분(reverse-mode-autodiff) 적용하여 gradients 노드는 theta에 대한 MSE의 그래디언트 벡터를 계산한다.

```
gradients = tf.gradients(mse, [theta])[0]
```

오차제곱평균 Data list

- 주로 텐서플로에 내장된 옵티마이저(Optimizer)를 사용한다.

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(mse)
```

```
optimizer = tf.train.MomentumOptimizer(learning_rate=learning_rate,
                                         momentum=0.9)
```


4. Tensorflow를 활용한 선형회귀분석

- 이제 그래프를 그리는 것은 모두 끝났다. 남은 것은 Session으로 방금 만들 그래프를 실행시키기만 하면 완료된다.

```
with tf.Session(graph=simple_regression) as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(100):
        _, l = sess.run([train, cost], feed_dict={X:x_data, Y:y_data})
        print("loss ", l)

    W_, b_ = sess.run([W, b])
    print(W_, b_)
```

- with tf.Session(graph = 그래프이름) as sess:**

위의 명령어로 Tensorflow session을 불러온다. 아래에 적은 코드는 모두 session으로 실행가능하게 된다. sess.run() 을 이용하면 오퍼레이터 같은 경우는 해당 연산의 결과 값이 텐서의 경우는 텐서의 값이 리턴된다.

- 제일 먼저 해야할 것은 변수를 초기화해** 주는 일이다.

sess.run(tf.global_variables_initializer()) 은 그래프 내의 모든 변수를 초기화 해주는 명령어이다.

4. Tensorflow를 활용한 선형회귀분석

```
sess.run([ ], feed_dict={ })
```

- 위 명령에서 [] 안에는 실행시켜줄 텐서나 오퍼레이터를 여러 개 써줄 수 있다. 예를 들어서 [train, cost] 이런식으로 써주면 train 오퍼레이터의 결과값과 cost 텐서의 결과값이 동시에 리턴된다.
- 만약에 해당 텐서나 오퍼레이터를 실행시키는데 placeholder가 관여되어 있다면 반드시 값을 넣어주어야 한다.
- 예제에서는 전에 텍스트파일에서 읽어온 x좌표 값과 y좌표값이 된다.
- 학습할 때는 여러 번 반복하므로 for문을 사용한다.

```
sess.run( [W,b] )
```

- 위 명령은 학습을 다 마치고 학습된 값이 어떤것인지를 확인하기 위하여 맨 마지막에 텐서 두개를 실행시켜서 값을 받아온다.

4. Tensorflow를 활용한 선형회귀분석

- 노드 값의 생애주기

- 한 노드를 평가할 때 텐서플로는 이 노드가 의존하고 있는 다른 노드들을 자동으로 찾아 먼저 평가한다.

```
w = tf.constant(3)
x = w + 2
y = x + 5
z = x * 3
```

```
with tf.Session() as sess:
    print(y.eval()) # 10
    print(z.eval()) # 15
```



w와 x를 두 번 평가하지 않고, y와 z를 효율적으로 평가하려면 텐서플로가 한번의 그래프 실행에서 y와 z를 모두 평가하도록 해야 한다.

```
with tf.Session() as sess:
    y_val, z_val = sess.run([y, z])
    print(y_val) # 10
    print(z_val) # 15
```

4. Tensorflow를 활용한 선형회귀분석

- 결과확인

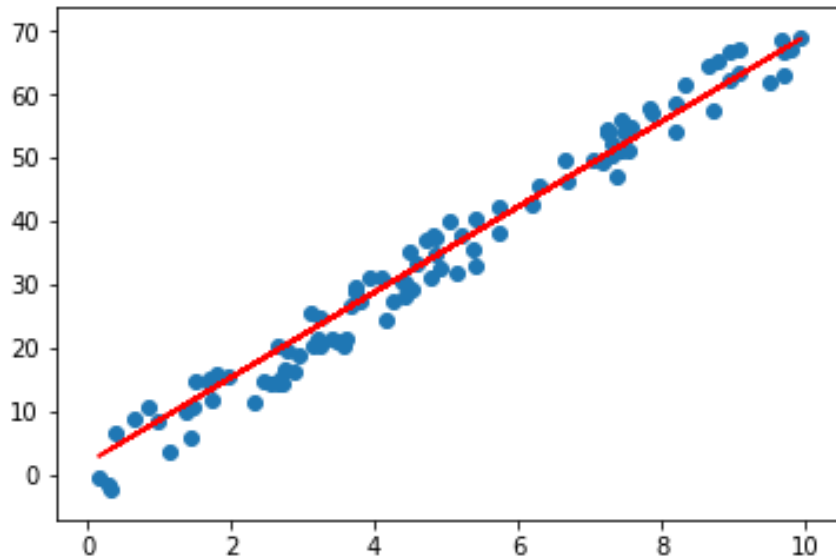
```
loss 1090.7887
loss 956.3057
loss 838.5583
loss 735.4638
loss 645.1989
loss 566.1668
loss 496.96964
loss 436.38367
loss 383.33716
loss 336.89197
loss 296.22644
loss 260.62134
loss 229.44707
loss 202.15208
loss 178.25368
loss 157.32918
loss 139.00842
loss 122.96748
loss 108.922585
```

- 학습을 진행하면서 loss 값이 점차 줄어드는 모습을 확인 할 수 있다.

4. Tensorflow를 활용한 선형회귀분석

- 학습결과를 그래프로 확인

```
plt.plot(x_data, x_data*W_ + b_, 'r')  
plt.scatter(x_data, y_data)  
plt.show()  
print(W_, b_)
```



6.737093 1.7930626

5. Tensorflow를 활용한 다중회귀분석(Multi Regression)

- 다중회귀 분석

```
multi_x, multi_y = [], []  
multi_raw = np.loadtxt('multi_regression.txt')  
multi_y = multi_raw[:,2]  
multi_x = multi_raw[:,1]
```

TF 1.15

- 기본적인 선형회귀와 크게 다른점은 없으나, 다만 x 변수가 여러 개 추가되는 점만 다르다. 즉, 가설함수는 다음과 같은 모양이 된다.

$$h = a_1 * x_1 + a_2 * x_2 + .. + a_n * x_n + b$$

- 다중 선형회귀분석을 위해 필요로 하는 'multi_regression.txt' 파일을 읽어온다.

5. Tensorflow를 활용한 다중회귀분석(Multi Regression)

- 다중회귀 분석

```
multi_x, multi_y = [], []  
multi_raw = np.loadtxt('multi_regression.txt')  
multi_y = multi_raw[:,2]  
multi_x = multi_raw[:,1:2]
```

- 일반 선형회귀문제와 다른점은 x데이터 부분이다, multi_raw[:,2] 앞의 :는 첫째행의 전체이며 뒤에 :2 는 둘째행의 0~1까지 데이터를 의미한다. 따라서 multi_y는 y점 5개 multi_x는 (x1,x2) 점 5개가 된다.

5. Tensorflow를 활용한 다중회귀분석(Multi Regression)

- 다중회귀 분석

```
with tf.Graph().as_default() as multi_regression:
    num_x = 2
    X = tf.placeholder(tf.float32, [None, num_x], name='X')
    Y = tf.placeholder(tf.float32, [None], name='Y')
    lr = tf.constant(1e-3, tf.float32)
    W = tf.get_variable("W", [1, num_x], tf.float32)
    b = tf.get_variable("b", dtype=tf.float32, initializer=tf.constant(1., tf.float32))

    h = tf.matmul(W, X, transpose_b=True) + b
    cost = tf.reduce_mean(tf.square(tf.subtract(h, Y)))
    train = tf.train.GradientDescentOptimizer(lr).minimize(cost)
```

- 또 달라진 점은 W의 모양이다. W를 굳이 저와같이 쓰지 않고도 손으로 일일이 $h=a_1*x_1 + a_2*x_2 + \dots$ 이런식으로 쓸 수 있지만, 만약 x변수가 100개가 된다고 생각하면 손으로 직접치는 것은 무리한 작업이 될 것이다.
- W모양을 [1,x변수 개수]의 형태로 만들어주고, 행렬을 곱하게 되면 이런 수고를 덜 수 있다.

5. Tensorflow를 활용한 다중회귀분석(Multi Regression)

- 다중회귀 분석의 가설함수

$$h = \text{tf.matmul}(W, X, \text{transpose_b=True}) + b$$

- 가설 함수는 위와 같다.
- matmul은 행렬 두개를 곱해주는 연산이다. 파라미터 W가 (1X2) 행렬이고, 입력데이터X가(데이터갯수 X 2) 행렬이므로 뒤의 X를 transpose해주어서 (2 X 데이터갯수) 모양으로 바꾸어 준 뒤 행렬 곱셈을 한다.
- 따라서 h는 (1 X 데이터개수), 즉 데이터 개수만큼의 가설함수 결과값이 배열로 나타나는 것이다.
- 나머지는 앞의 선형회귀와 같다. 오차 cost를 정의해주고, GradientDescentOptimizer를 통하여 오차를 줄여나간다.

5.Tensorflow를 활용한 다중회귀분석(Multi Regression)

- 다중회귀분석 결과 확인

```
with tf.Session(graph=multi_regression) as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(1000):
        _, l = sess.run([train, cost], feed_dict={X:multi_x, Y:multi_y})
        print("loss ", l)
    W_multi, b_multi = sess.run([W, b])
```

- tf.Session(graph=그래프이름), 세션을 불러올 때 실행시킬 그래프 이름을 정확히 넣어주어야 한다.
- 1000번동안 train이라는 이름의 Optimizer와 에러값 cost를 실행시키면서 오차를 계속 줄여 나아간다.

5. Tensorflow를 활용한 다중회귀분석(Multi Regression)

- 실행결과

```
with tf.Session(graph=multi_regression) as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(1000):
        _, l = sess.run([train, cost], feed_dict={X:multi_x, Y:multi_y})
        print("loss ", l)
    W_multi, b_multi = sess.run([W, b])
```

```
loss 0.19029444
loss 0.1897181
loss 0.1891512
loss 0.1885936
loss 0.18804511
loss 0.18750551
loss 0.18697464
loss 0.18645225
loss 0.18593831
loss 0.18543252
loss 0.18493469
loss 0.18444481
loss 0.18396257
loss 0.18348779
loss 0.18302044
loss 0.18256035
loss 0.18210734
loss 0.18166188
```

5. Tensorflow를 활용한 다중회귀분석(Multi Regression)

- 다중회귀분석 결과 확인

```
for j in range(len(multi_x)):
    sum = 0
    for i in range(2):
        sum += W_multi[0][i] * multi_x[j,i]
    print(multi_y[j], sum+b_multi)
print(W_multi, b_multi)
```

- 학습하여 얻어진 W라미터와 b값을 W_multi, b_multi로 받아온 후, 직접 가설함수 $h = W_1 \cdot x_1 + W_2 \cdot x_2 + b$ 를 계산하여 실제 데이터값과 학습을 얻어진 데이터 값을 비교하는 코드이다.

5. Tensorflow를 활용한 다중회귀분석(Multi Regression)

- 실행결과

```
for j in range(len(multi_x)):
    sum = 0
    for i in range(2):
        sum += W_multi[0][i] * multi_x[j,i]
    print(multi_y[j], sum+b_multi)
print(W_multi, b_multi)
```

1.0 1.5345438122749329

2.0 2.272863268852234

3.0 3.1532837748527527

4.0 3.820552706718445

5.0 4.7720237374305725

[[0.80937 0.7738447]] 0.72517383 **coefficient**

- 실행하면 위와 같은 비슷한 결과가 출력된다. 왼쪽 값이 실제 데이터, 오른쪽 값이 학습한 모델로 구한 값이다.(실측치와 예측치의 오차를 확인해볼 수 있다)
- 밑단에 `[[0.80937 0.7738447]]` 은 학습하여 얻어진 W, 0.72517383은 b값이 된다.

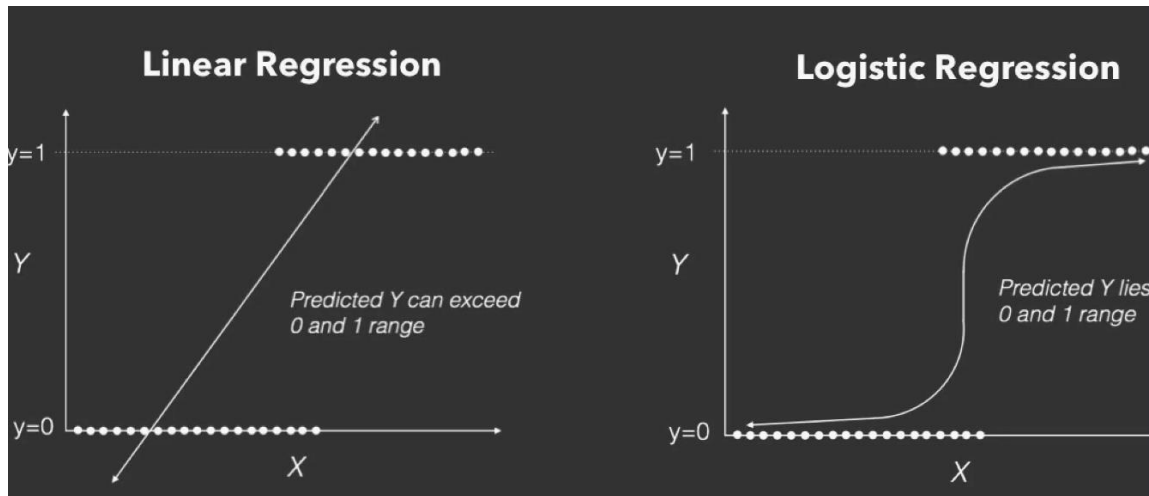
5. Tensorflow를 활용한 Logistic Regression

- 로짓함수

$$\log \frac{p}{1-p} = a + b[1] * x[i,1] + + b[n]x[i,n]$$

선형방정식 = Z

$$\mathbf{z} = \log \frac{p}{1-p} \quad (\text{로짓함수라고 한다})$$

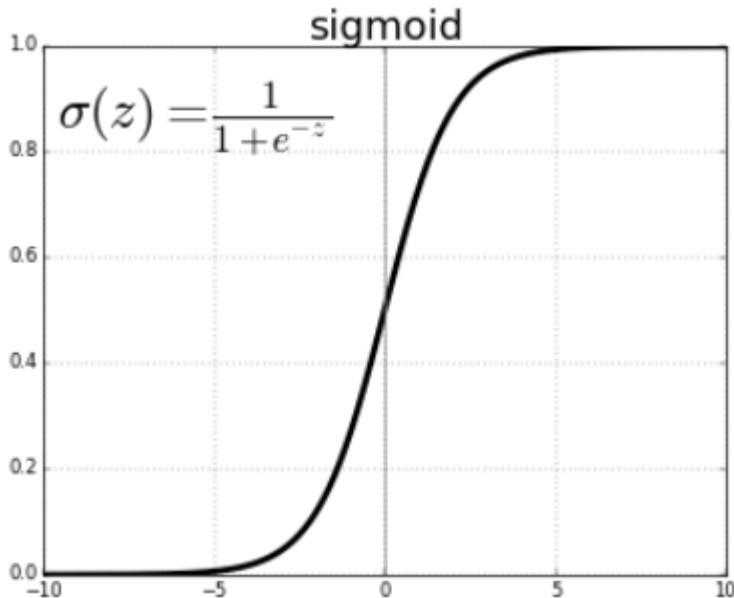


5. Tensorflow를 활용한 Logistic Regression

- 로지스틱 회귀분석(Logistic Regression)

- 분류모형

- 선형회귀와 거의 비슷하지만, 출력에 대해 값을 구하는 것이 아니라, x_data 에 대하여 0인지 1인지 두 가지의 상태로 분류하는 회귀분석이다.



- sigmoid는 위와 같은 함수이며, x 가 어떤 값이 들어오든 (수식에선 z) 항상 0과 1사이의 값만 나오게 된다.
- 0,1 두가지 상태를 분류한다고 하면 1일때에는 1에 가까운 값이 나오게, 0일때에는 0에 가까운 값이 나오게 하면 된다.
- sigmoid 함수 값이 꼭 0,1로 나오지 않아도된다. 일반적으로는 0.5보다 크면 1, 0.5보다 작으면 0으로 분류한다.

5. Tensorflow를 활용한 Logistic Regression

- Logistic Regression 데이터 준비

```
logistic = np.loadtxt('logistic_regression.txt')  
logistic_x = logistic[:, :-1]  
logistic_y = logistic[:, -1]
```

TF 1.15

- 먼저 데이터를 읽어와서 배열에 저장한다.
- `[:, :-1]` 에서 `-1`이 의미하는 것은 인덱스에서 제일 마지막 숫자이다. 예제에서는 크기 4짜리 배열이므로 0,1,2,3 인덱스 중 마지막 인덱스 3을 의미한다.

5. Tensorflow를 활용한 Logistic Regression

- Logistic Regression 그래프 만들기

TF 1.15

```
with tf.Graph().as_default() as logistic_regression:
```

```
    num_x = len(logistic_x[0])
```

```
    X = tf.placeholder(tf.float32, [None, num_x], name='X')
```

```
    Y = tf.placeholder(tf.float32, [None], name='Y')
```

```
    lr = tf.constant(1e-1, tf.float32)
```

```
    W = tf.get_variable("W", [1, num_x], tf.float32)
```

```
    b = tf.get_variable("b", dtype=tf.float32, initializer=tf.constant(1., tf.float32))
```

```
    h = tf.matmul(W, X, transpose_b=True) + b
```

```
    hypothesis = tf.sigmoid(h)
```

```
    cost = -tf.reduce_mean((1-Y) * tf.log(hypothesis) + Y * tf.log(1-  
hypothesis))
```

```
    train = tf.train.GradientDescentOptimizer(lr).minimize(cost)
```

5. Tensorflow를 활용한 Logistic Regression

- **Logistic Regression 그래프 만들기**

- X 변수 개수가 3개이므로 X, placeholder의 모양이 [None, 3], W파라미터 모양은 [1,3]이 된다.

- 다중 선형회귀 분석과 마찬가지로 가설함수는

$$\mathbf{h} = \mathbf{tf.matmul(W, X, transpose_b=True)} + \mathbf{b}$$

W배열과 X배열을 곱한 후 b파라미터를 더해준다.

- 이제 가설 함수 값을 sigmoid 함수에 넣어서 0과 1사이의 값이 되게 만들어 준다.

- 다중 선형회귀와 cost 함수가 다르다.

$$\mathbf{cost} = -\mathbf{tf.reduce_mean}((1-Y) * \mathbf{tf.log(hypothesis)} + \\ Y * \mathbf{tf.log(1-hypothesis)})$$

- 이 오차함수의 뜻은 Y가 0일때 $-\log(\text{sigmoid함수값})$ 이 최소가 되도록, 즉 log안의 sigmoid 값이 0에 가까워지도록 Y가 1일때에는 $-\log(1-\text{sigmoid함수값})$ 이 되도록, 즉 log 안의 sigmoid이 값이 1에 가까워지도록 계산하는 오차 함수이다.

5. Tensorflow를 활용한 Logistic Regression

- Logistic Regression

TF 1.15

```
with tf.Session(graph=logistic_regression) as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(10000):
        _, l = sess.run([train, cost], feed_dict={X:logistic_x, Y:logistic_y})
        print("loss ", l)
    W_logistic, b_logistic = sess.run([W, b])
```

```
loss 3.8476257
loss 2.6574295
loss 1.5468754
loss 0.75774974
loss 0.555451
loss 0.54326975
loss 0.5396911
loss 0.5366704
loss 0.5337413
loss 0.5308661
loss 0.5280404
loss 0.52526236
loss 0.5225304
loss 0.5198433
loss 0.5171996
loss 0.5145982
loss 0.5120378
loss 0.50951713
loss 0.50703543
loss 0.50455114
```

- 만번 정도 실행시킨 후, 파라미터 값을 `W_logistic`과 `b_logistic`에 저장해 둔다.

5. Tensorflow를 활용한 Logistic Regression

- Logistic Regression

TF 1.15

```
for j in range(len(logistic_x)):
    sum_ = 0
    for i in range(3):
        sum_ += W_logistic[0][i] * logistic_x[j,i]
    print('label', logistic_y[j], 'probability', 1/(1+math.exp(sum_+b_)))
print(W_logistic, b_logistic)
```

```
for j in range(len(logistic_x)):
    sum_ = 0
    for i in range(3):
        sum_ += W_logistic[0][i] * logistic_x[j,i]
    print('label', logistic_y[j], 'probability', 1/(1+math.exp(sum_+b_)))
print(W_logistic, b_logistic)
```

```
label 0.0 probability 1.2787762392179444e-07
label 0.0 probability 1.9225515194186828e-05
label 0.0 probability 0.21004270497375938
label 1.0 probability 0.9808086433296774
label 1.0 probability 0.988169932169127
label 1.0 probability 0.9607203967175564
[[10.306391 -0.24564645 -4.7672925 ]] 11.32249
```

- 학습을 통해 구한 파라미터 W, b를 이용하여 가설함수 h를 계산하고, sigmoid 함수에 넣어서 0~1사이의 값으로 만들어준다.
- 0 일 때에는 1.27×10^{-7} 로 거의 0에 가까운 값이, 1 일 때에는 0.98로 거의 1에 가까운 값이 나오는 것을 확인할 수 있다.