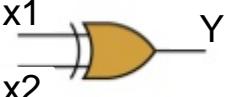


NN for XOR, Tensorboard for XOR NN

Lecture 08

NN for XOR

XOR data set

Boolean Expression	Logic Diagram Symbol	Truth Table															
$Y = x_1 \oplus x_2$		<table border="1"><thead><tr><th>x1</th><th>x2</th><th>Y</th></tr></thead><tbody><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></tbody></table>	x1	x2	Y	0	0	0	0	1	1	1	0	1	1	1	0
x1	x2	Y															
0	0	0															
0	1	1															
1	0	1															
1	1	0															

```
x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y_data = np.array([0, 1, 1, 0], dtype=np.float32)
```

XOR with logistic regression?

```

import tensorflow as tf
import numpy as np
x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y_data = np.array([[0], [1], [1], [0]], dtype=np.float32)

X = tf.placeholder(tf.float32, [None, 2])
Y = tf.placeholder(tf.float32, [None, 1])
W = tf.Variable(tf.random_normal([2, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Hypothesis using sigmoid
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)

# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

# Accuracy computation
# True if hypothesis>0.5 else False
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())

    for step in range(10001):
        sess.run(train, feed_dict={X: x_data, Y: y_data})
        if step % 100 == 0:
            print("step: ", step, "\tcost : ", sess.run(cost, feed_dict={X: x_data, Y: y_data}))
            print("W: \n", sess.run(W), "\nb: \n", sess.run(b), "\n", "*50")
    # Accuracy report
    h, c, a = sess.run([hypothesis, predicted, accuracy], feed_dict={X: x_data, Y: y_data})
    print("Hypothesis \n", h.reshape(-1,4), "\ny_data \n", y_data.reshape(-1,4), "\nCorrect \n", c.reshape(-1,4), "\nAccuracy \n", a)

```

```
step: 0          cost :  0.85884523
W:
 [[0.27288756]
 [2.2499688 ]]
b:
 [-0.87927616]
=====
step: 100         cost :  0.74584824
W:
 [[0.18489242]
 [1.2742372 ]]
b:
 [-0.8568866]
=====
step: 200         cost :  0.7148558
W:
 [[0.20223512]
 [0.79047865]]
b:
 [-0.59119153]
.
.
.
step: 9900        cost :  0.6931472
W:
 [[1.3336123e-07]
 [1.3270730e-07]]
b:
 [-1.7855265e-07]
=====
step: 10000        cost :  0.6931472
W:
 [[1.3336123e-07]
 [1.3270730e-07]]
b:
 [-1.7855265e-07]
=====
Hypothesis
 [[0.5 0.5 0.5 0.5]]
y_data
 [[0. 1. 1. 0.]]
Correct
 [[0. 0. 0. 0.]]
Accuracy
 0.5
```

XOR with logistic regression?



Neural Net (Single Layer)

```

import tensorflow as tf
import numpy as np
x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y_data = np.array([0, 1, 1, 0], dtype=np.float32)

X = tf.placeholder(tf.float32, [None, 2])
Y = tf.placeholder(tf.float32, [None, 1])
W = tf.Variable(tf.random_normal([2, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

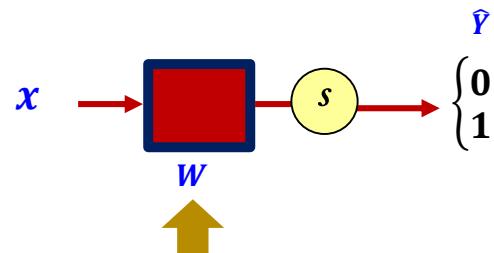
```

Hypothesis using sigmoid
`hypothesis = tf.sigmoid(tf.matmul(X, W) + b)`

row : 2
column : 1

$$\begin{matrix} x_1 & x_2 \end{matrix} \times \begin{matrix} W \\ W \end{matrix} = \begin{matrix} Y \\ Y \\ Y \\ Y \end{matrix}$$

0	0
0	1
1	0
1	1



Neural Net (Multi Layer)

```

import tensorflow as tf
import numpy as np
x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]],
                  dtype=np.float32)
y_data = np.array([[0], [1], [1], [0]], dtype=np.float32)

X = tf.placeholder(tf.float32, [None, 2])
Y = tf.placeholder(tf.float32, [None, 1])

```

```

W1 = tf.Variable(tf.random_normal([2, 2]), name='weight1')
b1 = tf.Variable(tf.random_normal([2]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

```

```

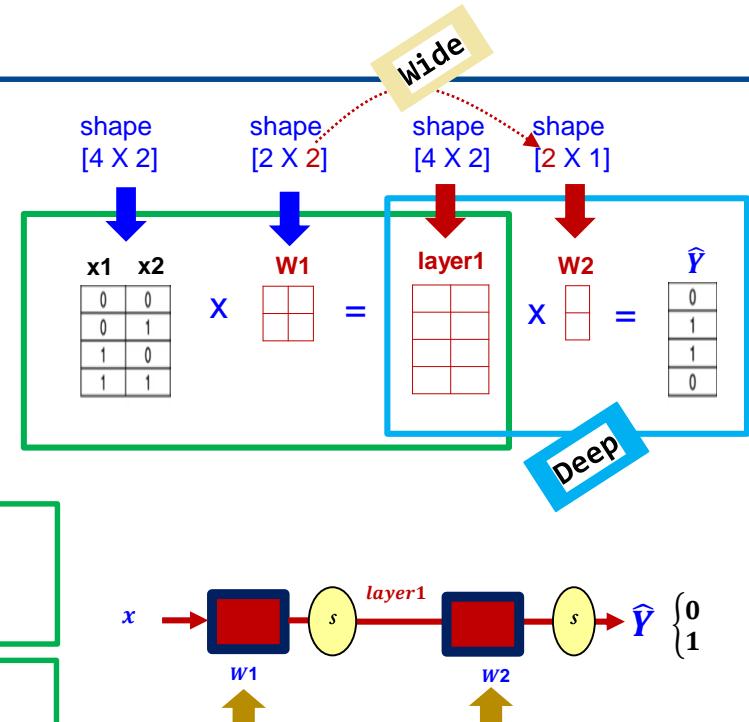
W2 = tf.Variable(tf.random_normal([2, 1]), name='weight2')
b2 = tf.Variable(tf.random_normal([1]), name='bias2')

```

```

hypothesis = tf.sigmoid(tf.matmul(layer1, W2) + b2) ←  $\hat{Y}$ 

```



ex08_2.ipynb

NN for XOR (Multi Layer)

```

import tensorflow as tf
import numpy as np
x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y_data = np.array([[0], [1], [1], [0]], dtype=np.float32)
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)

W1 = tf.Variable(tf.random_normal([2, 2]), name='weight1')
b1 = tf.Variable(tf.random_normal([2]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([2, 1]), name='weight2')
b2 = tf.Variable(tf.random_normal([1]), name='bias2')
hypothesis = tf.sigmoid(tf.matmul(layer1, W2) + b2)

# cost/Loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
# Accuracy computation
# True if hypothesis>0.5 else False
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))
# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())
    for step in range(10001):
        sess.run(train, feed_dict={X: x_data, Y: y_data})
        if step % 100 == 0:
            print("step : ", step, "cost :", sess.run(cost, feed_dict={X: x_data, Y: y_data}))
            print("W1 : \n", sess.run(W1), "\nb1: \n", sess.run(b1))
            print("Layer 1: \n", sess.run(layer1, feed_dict={X: x_data, Y: y_data}))
            print("W2 : \n", sess.run(W2), "\nb2: \n", sess.run(b2))
            print("hypothesis : \n", sess.run(hypothesis, feed_dict={X:x_data}),"\n" , "="*50)
    # Accuracy report
    p, a = sess.run([predicted, accuracy],feed_dict={X: x_data, Y: y_data})
    print("• Correct: \n", p, "\n• Accuracy: \n", a)

```

```

step : 0 cost : 0.76641256
W1 :
[[ -0.11243332 -0.48388383]
 [ 0.5747957 -0.46335787]]
b1:
[ 0.48714975 -0.46306232]
Layer 1:
[[ 0.6194348  0.38625962]
 [ 0.74306214  0.28365153]
 [ 0.59259814  0.27949938]
 [ 0.72101706  0.19618613]]
W2 :
[[ 0.39562288]
 [ 0.32466972]]
b2:
[ 0.41144657]
hypothesis :
[[ 0.6860929 ]
 [ 0.68944204]
 [ 0.67626005]
 [ 0.6814391 ]]
=====
step : 100 cost : 0.6936093
W1 :
[[ -0.11757189 -0.4828046 ]
 [ 0.5614868 -0.4625066 ]]
b1:
[ 0.4612578 -0.48502386]
Layer 1:
[[ 0.61331254  0.38106653]
 [ 0.73550683  0.27938175]
 [ 0.5850856  0.27531356]
 [ 0.71201134  0.19304648]]
W2 :
[[ 0.06586391]
 [ 0.16061953]]
b2:
[-0.07541786]
hypothesis :
[[ 0.50654566]
 [ 0.5044748 ]
 [ 0.50183475]
 [ 0.50062126]]
=====
```

```

step : 200 cost : 0.6935026
W1 :
[[ -0.11463588 -0.47466004]
 [ 0.5607306 -0.45420668]]
b1:
[ 0.4608377 -0.4835354]
Layer 1:
[[ 0.6132129  0.38141763]
 [ 0.73527795  0.28135666]
 [ 0.5856962  0.27723965]
 [ 0.712372  0.19585545]]
W2 :
[[ 0.05790563]
 [ 0.13454905]]
b2:
[-0.07894189]
hypothesis :
[[ 0.5019715 ]
 [ 0.50037277]
 [ 0.4980689 ]
 [ 0.49716514]]
.
.
step : 9900 cost : 0.041418742
W1 :
[[ 3.9609594 -6.2198763]
 [ 3.967846 -6.2877 ]]
b1:
[ -6.1204443  2.2771738]
Layer 1:
[[ 2.19266117e-03 9.06968892e-01]
 [ 1.04088664e-01 1.78012215e-02]
 [ 1.03448220e-01 1.90266892e-02]
 [ 8.59163702e-01 3.60559716e-05]]
W2 :
[[ -8.281344]
 [ -8.41226 ]]
b2:
[ 4.095111]
hypothesis :
[[ 0.02784924]
 [ 0.95620143]
 [ 0.9559913 ]
 [ 0.04652059]]
=====
```

ex08_2.ipynb

Notes

step : 10000 cost : 0.039866842

```

W1 :
[[ 3.9923384 -6.23685 ]
 [ 3.9991689 -6.303531 ]]
b1:
[ -6.1786285  2.2928936]
Layer 1:
[[ 2.08556326e-03 9.08286810e-01]
 [ 1.02342851e-01 1.77992862e-02]
 [ 1.01717055e-01 1.90033056e-02]
 [ 8.60671580e-01 3.54452095e-05]]
W2 :
[[ -8.357225]
 [ -8.470572]]
b2:
[ 4.128153]
hypothesis :
[[ 0.02704105]
 [ 0.9577944 ]
 [ 0.957593 ]
 [ 0.0445754 ]]
=====
```

- Correct:

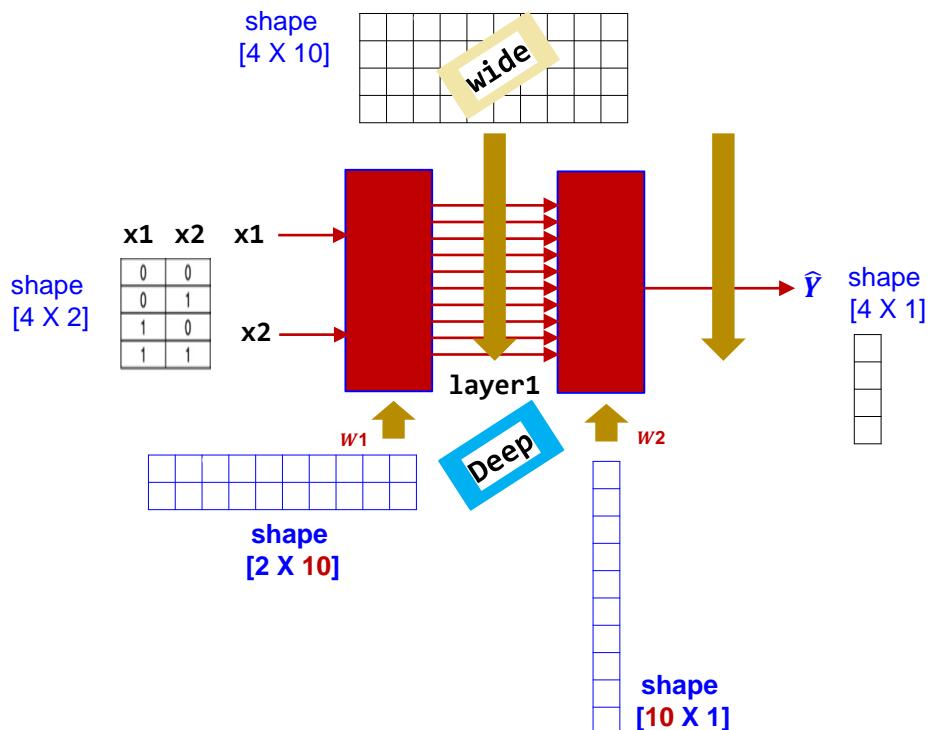
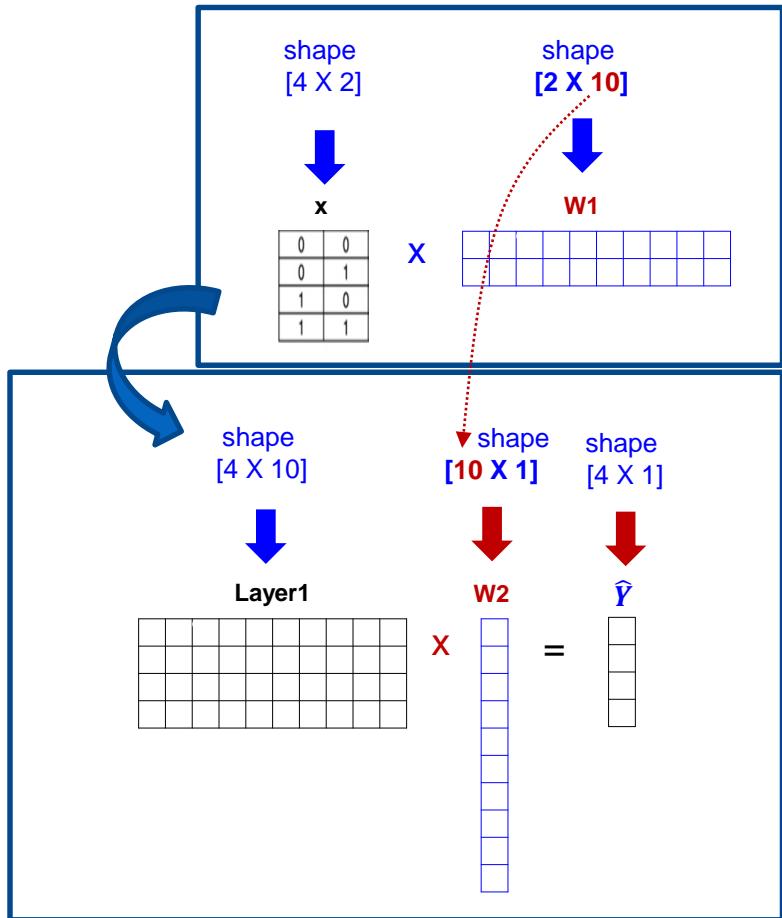
```

[[0.]
 [1.]
 [1.]
 [0.]]
=====
```

- Accuracy:

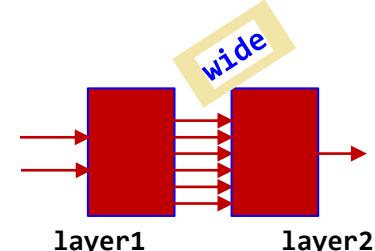
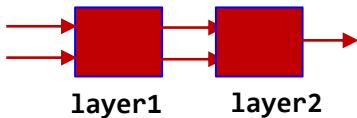
1.0

Neural Net (Multi Layer)



Wide NN for XOR

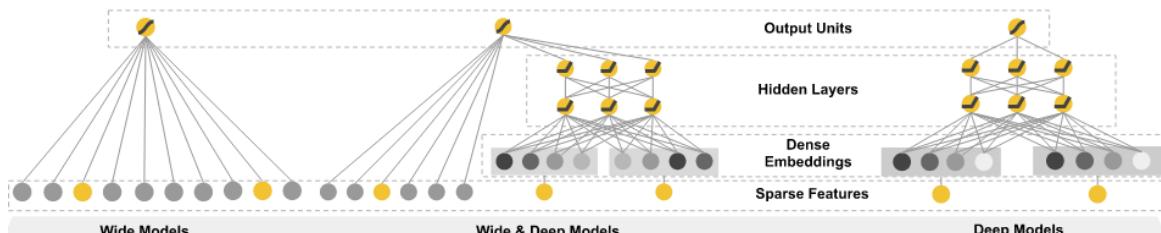
ex08_2(wide).ipynb



원하는 출력 형태로 정의

```
w1 = tf.Variable(tf.random_normal([2, 10]), name='weight1')
b1 = tf.Variable(tf.random_normal([10]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

w2 = tf.Variable(tf.random_normal([10, 1]), name='weight2')
b2 = tf.Variable(tf.random_normal([1]), name='bias2')
hypothesis = tf.sigmoid(tf.matmul(layer1, W2) + b2)
```



참고 : print(tf.shape(W1).eval(), tf.shape(W2).eval()) # W1, W2 shape 출력

Better learning

[2,10], [10,1]

Hypothesis:

[[0.00358802] [0.99366933] [0.99204296] [0.0095663]]

Correct:

[[0.] [1.] [1.] [0.]]

Accuracy: 1.0

[2,2], [2,1]

Hypothesis:

[[0.01338218] [0.98166394] [0.98809403] [0.01135799]]

Correct:

[[0.] [1.] [1.] [0.]]

Accuracy: 1.0

Deep NN for XOR

```

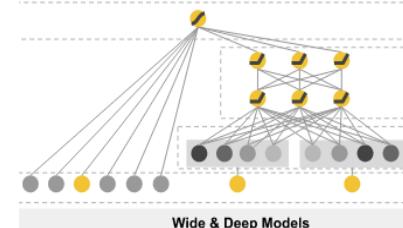
W1 = tf.Variable(tf.random_normal([2, 10]), name='weight1')
b1 = tf.Variable(tf.random_normal([10]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([10, 10]), name='weight2')
b2 = tf.Variable(tf.random_normal([10]), name='bias2')
layer2 = tf.sigmoid(tf.matmul(layer1, W2) + b2)

W3 = tf.Variable(tf.random_normal([10, 10]), name='weight3')
b3 = tf.Variable(tf.random_normal([10]), name='bias3')
layer3 = tf.sigmoid(tf.matmul(layer2, W3) + b3)

W4 = tf.Variable(tf.random_normal([10, 1]), name='weight4')
b4 = tf.Variable(tf.random_normal([1]), name='bias4')
hypothesis = tf.sigmoid(tf.matmul(layer3, W4) + b4)

```



Better learning

4 layers

2 layers

Hypothesis:

[[7.80e-04]
[9.99e-01]
[9.98e-01]
[1.55e-03]]

Correct:

[[0.]
[1.]
[1.]
[0.]]

Accuracy: 1.0

Hypothesis:

[[0.01338218]
[0.98166394]
[0.98809403]
[0.01135799]]

Correct:

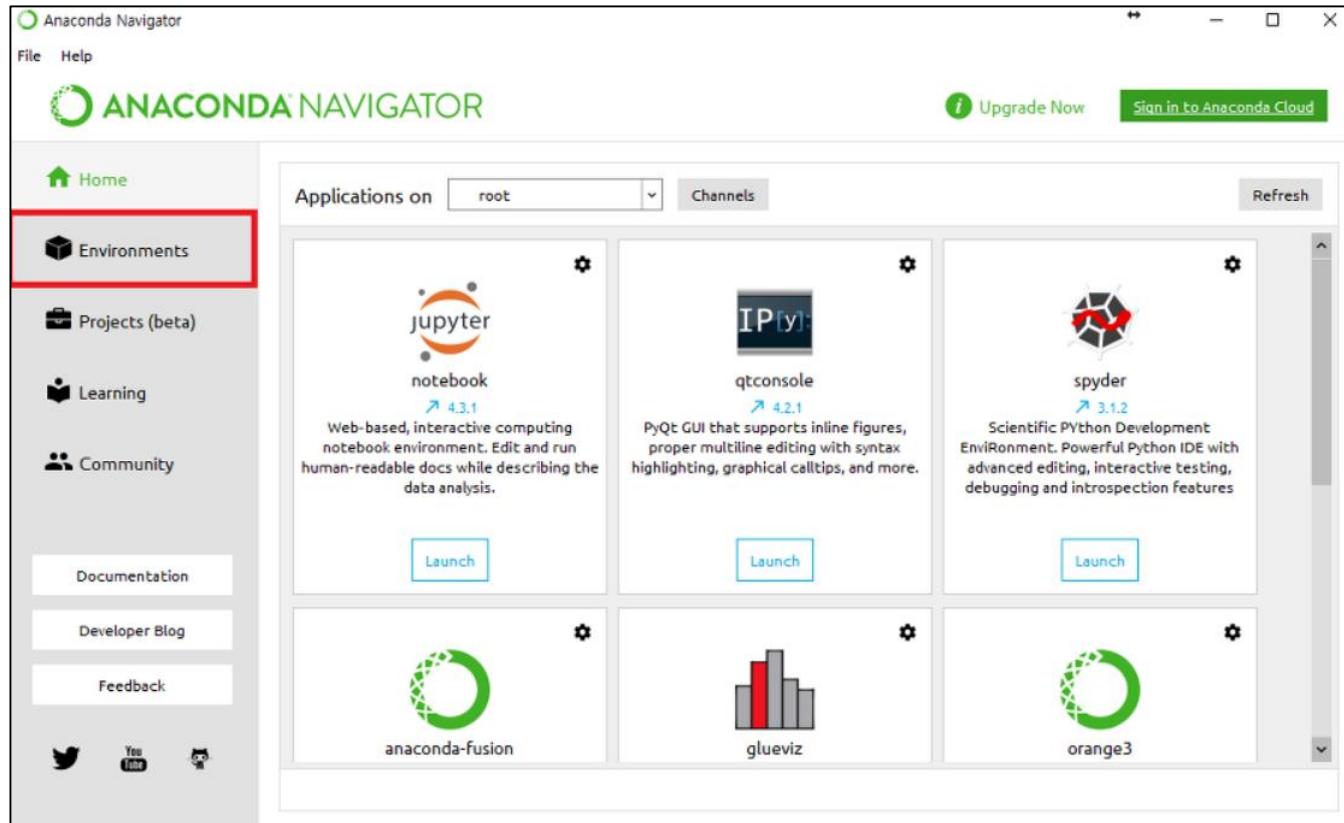
[[0.]
[1.]
[1.]
[0.]]

Accuracy: 1.0

Tensorboard for XOR NN

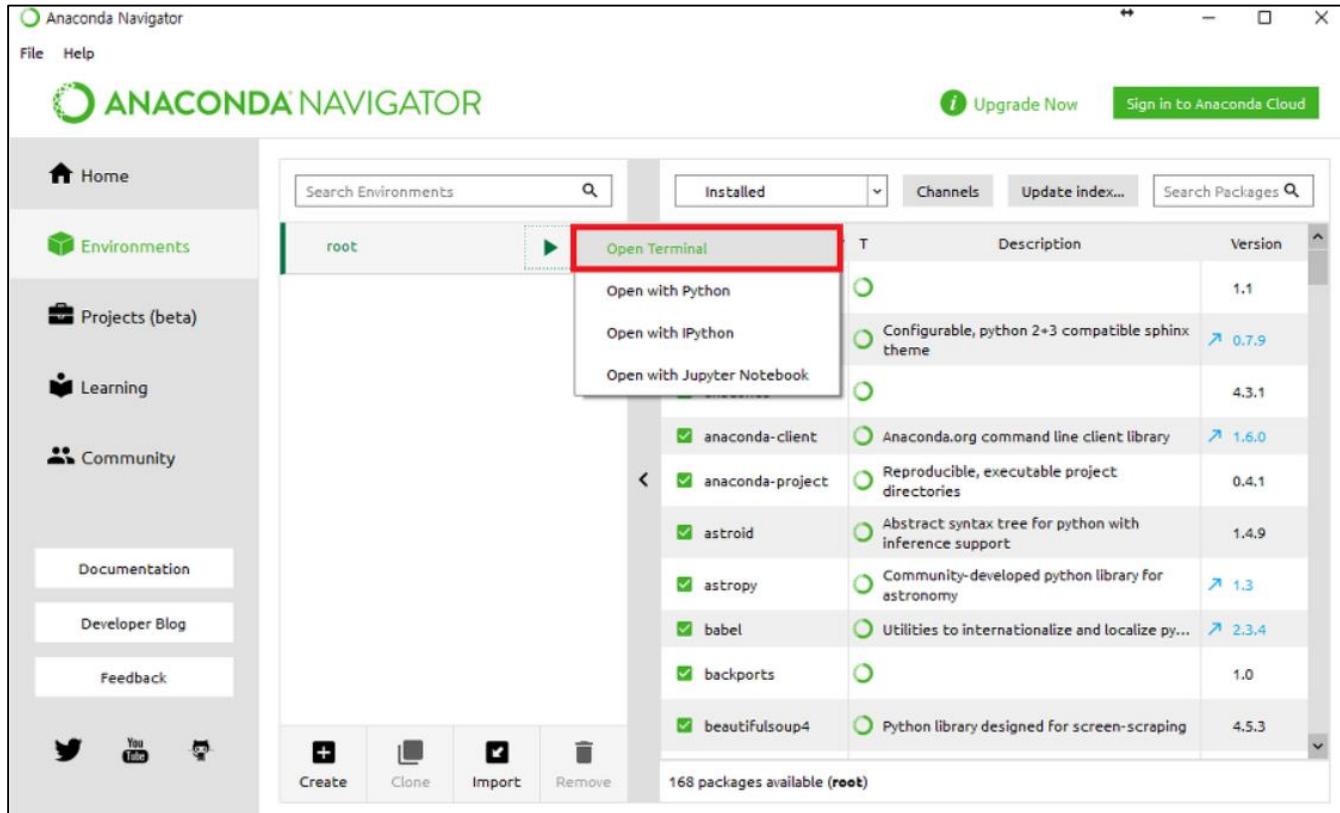
TensorBoard

왼쪽 사이드바에서 **Environments** 클릭



TensorBoard

root의 오른쪽 화살표 클릭 → Open Terminal 선택



TensorBoard

```
C:\Users\Park>conda create -n virtual_envs1 python=3.5
Solving environment: done
```

conda create -n virtual_envs1 python=3.5

current version: 4.4.10
latest version: 4.5.5

Please update conda by running

\$ conda update -n base conda

Package Plan

environment location: C:\Users\Park\Anaconda3\envs\virtual_envs1
added / updated specs:
- python=3.5

virtual_envs1부분은 원하는 이름으로
python은 무조건 3.5버전으로!!(17년 4월 17일 기준)
윈도우의 경우 tensorflow는
무조건 3.5버전만 지원하기 때문...

Proceed ([y]/n)? y

```
Preparing transaction: done
Verifying transaction: -
SafetyError: The package for setuptools located at C:\Users\Park appears to be corrupted. The path 'Scripts/easy_install.exe' has a sha256 mismatch.
    reported sha256: 993203a406e04936a07829b1f482fd27d739b640482e2
    actual sha256: e7d7af2a32ccb3fd29664403f39cebd32b2120dee5f5202
```

```
SafetyError: The package for wheel located at C:\Users\Park\Anaconda3\envs\virtual_envs1\Scripts\wheel.exe' has a sha256 mismatch.
    reported sha256: 993203a406e04936a07829b1f482fd27d739b640482e2
    actual sha256: fb88d82081d11c5878fe8d644d3d65b472754c9c0abbee0
```

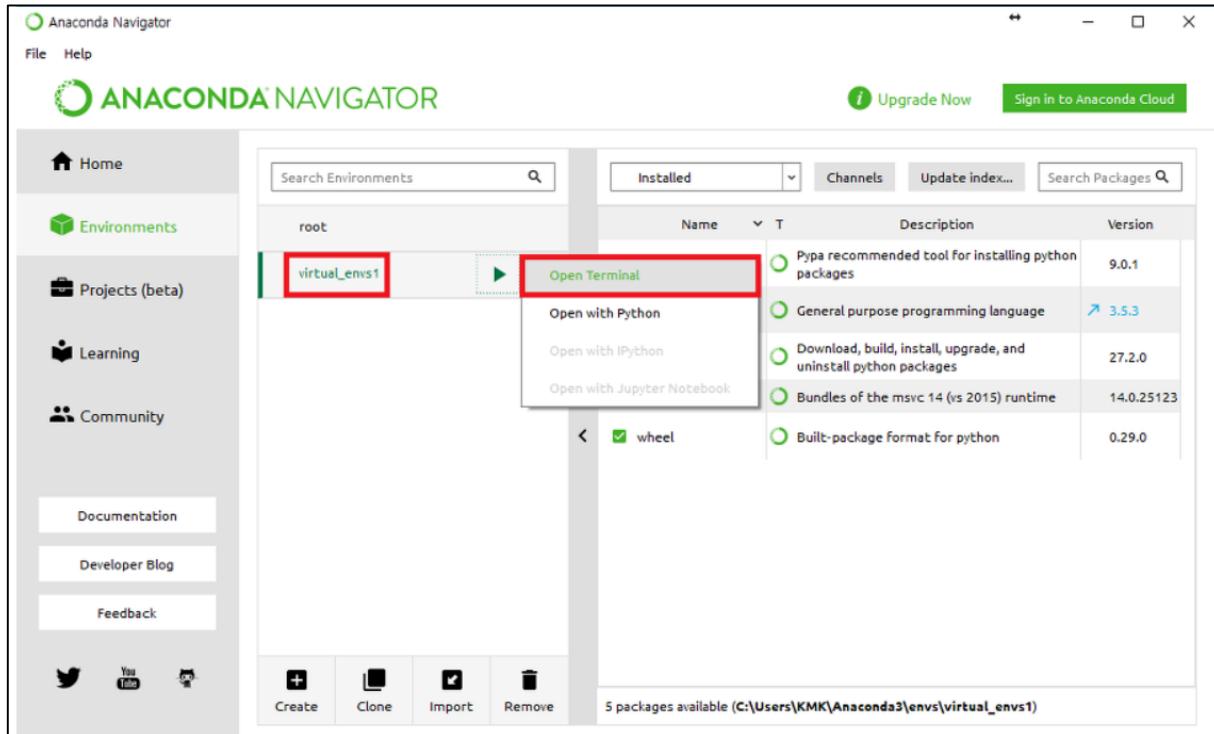
done
Executing transaction: done

```
#  
# To activate this environment, use:  
# > activate virtual_envs1  
#  
# To deactivate an active environment, use:  
# >  
# * 프롬프트가 나오면... 설치 완료된 것!!  
# (python 3.5버전을 가진 가상 환경을 구축함)
```

C:\Users\Park>

TensorBoard

cmd창을 끄고, 다시 Navigator로 이동
virtual_envs1 클릭 → 화살표 클릭 → Open Terminal 클릭



TensorBoard

공식 홈페이지

https://www.tensorflow.org/install/install_windows 참조

아래의 버전 중에서 자신의 환경에 적합한 하나를 선택해서 입력

CPU 버전

```
pip install --ignore-installed --upgrade https://storage.googleapis.com/tensorflow/windows/cpu/tensorflow-1.0.1-cp35-cp35m-win_amd64.whl
```

GPU 버전

```
pip install --ignore-installed --upgrade https://storage.googleapis.com/tensorflow/windows/gpu/tensorflow_gpu-1.0.1-cp35-cp35m-win_amd64.whl
```

```
(virtual_envs1) C:\Users\Park>pip install --ignore-installed --upgrade https://storage.googleapis.com/tensorflow/windows/cpu/tensorflow-1.0.1-cp35-cp35m-win_amd64.whl
Collecting tensorflow==1.0.1 from https://storage.googleapis.com/tensorflow/windows/cpu/tensorflow-1.0.1-cp35-cp35m-win_amd64.whl
  Using cached https://storage.googleapis.com/tensorflow/windows/cpu/tensorflow-1.0.1-cp35-cp35m-win_amd64.whl
  Get requirements from https://storage.googleapis.com/tensorflow/windows/cpu/tensorflow-1.0.1-cp35-cp35m-win_amd64.whl
    
```

pip install --ignore-installed --upgrade https://storage.googleapis.com/tensorflow/windows/cpu/tensorflow-1.0.1-cp35-cp35m-win_amd64.whl

```
Collecting numpy>=1.11.0 (from tensorflow==1.0.1)
  Using cached https://files.pythonhosted.org/packages/f3/71/94628784c3f07d4bc0dd38f8753e8f751d66cfda6823591179608c27f09/numpy-1.14.5-cp35-none-win_amd64.whl
Collecting protobuf>=3.1.0 (from tensorflow==1.0.1)
  Using cached https://files.pythonhosted.org/packages/f0/7d/1145805ef3ac475074f8d14d1c0512a79ef709ddf35ca89c5fa4fc94065/protobuf-3.6.0-cp35-cp35m-win_amd64.whl
Collecting six>=1.10.0 (from tensorflow==1.0.1)
  Using cached https://files.pythonhosted.org/packages/67/4b/141a581104b1f6397bfa78ac9d43d8ad29a7ca43ea90a2d863fe3056e86a/six-1.11.0-py2.py3-none-any.whl
Collecting setuptools (from protobuf>=3.1.0->tensorflow==1.0.1)
  Using cached https://files.pythonhosted.org/packages/7f/e1/820d941153923aac1d49d7fc37e17b6e73bfbd2904959fffbad77900cf92/setuptools-39.2.0-py2.py3-none-any.whl
Installing collected packages: wheel, numpy, six, setuptools, protobuf, tensorflow
Successfully installed numpy-1.14.5 protobuf-3.6.0 setuptools-39.2.0 six-1.11.0 tensorflow-1.0.1 wheel-0.31.1
```

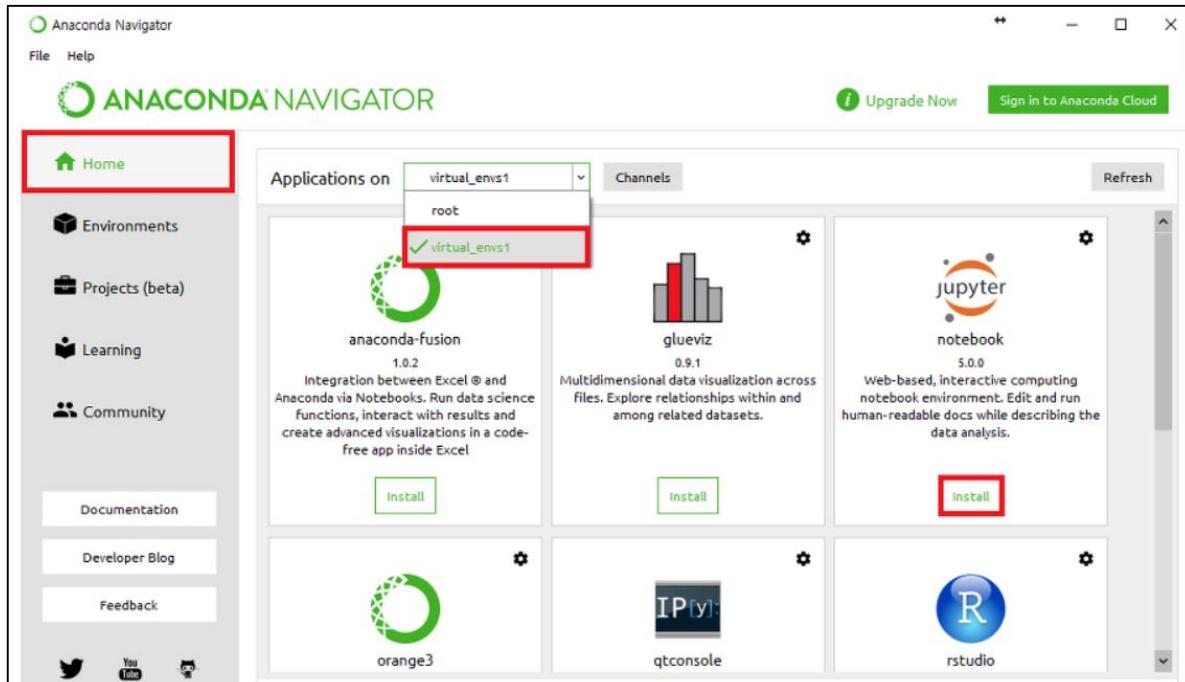
```
(virtual_envs1) C:\Users\Park>
```

TensorBoard

- 설치 완료 후 cmd창 끄고
Navigator로 이동

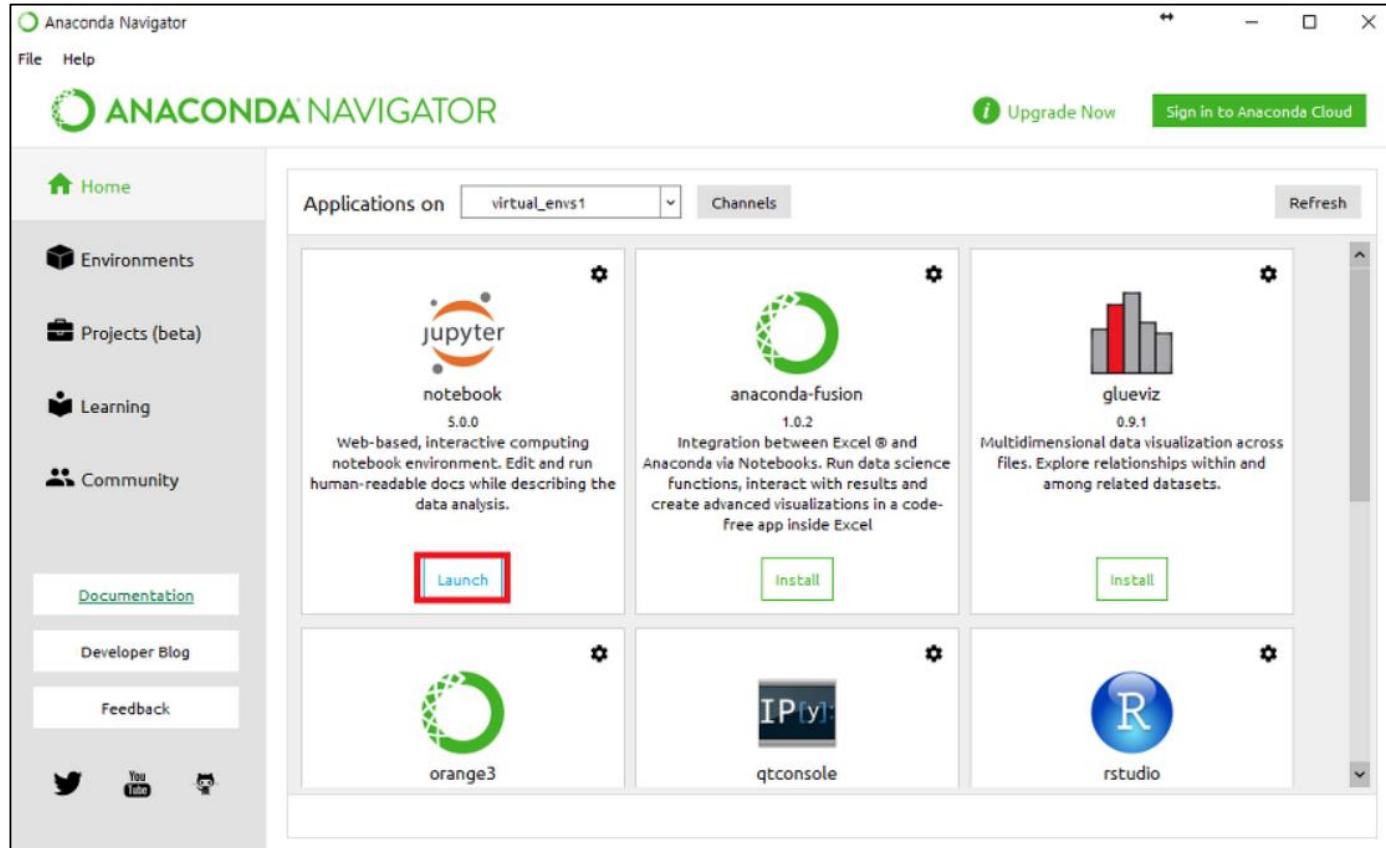


- 왼쪽 사이드바의 **Home** 클릭
→ Applications on [virtual_envs1] 클릭
→ 아래 프로그램들에서 **jupyter install** 클릭



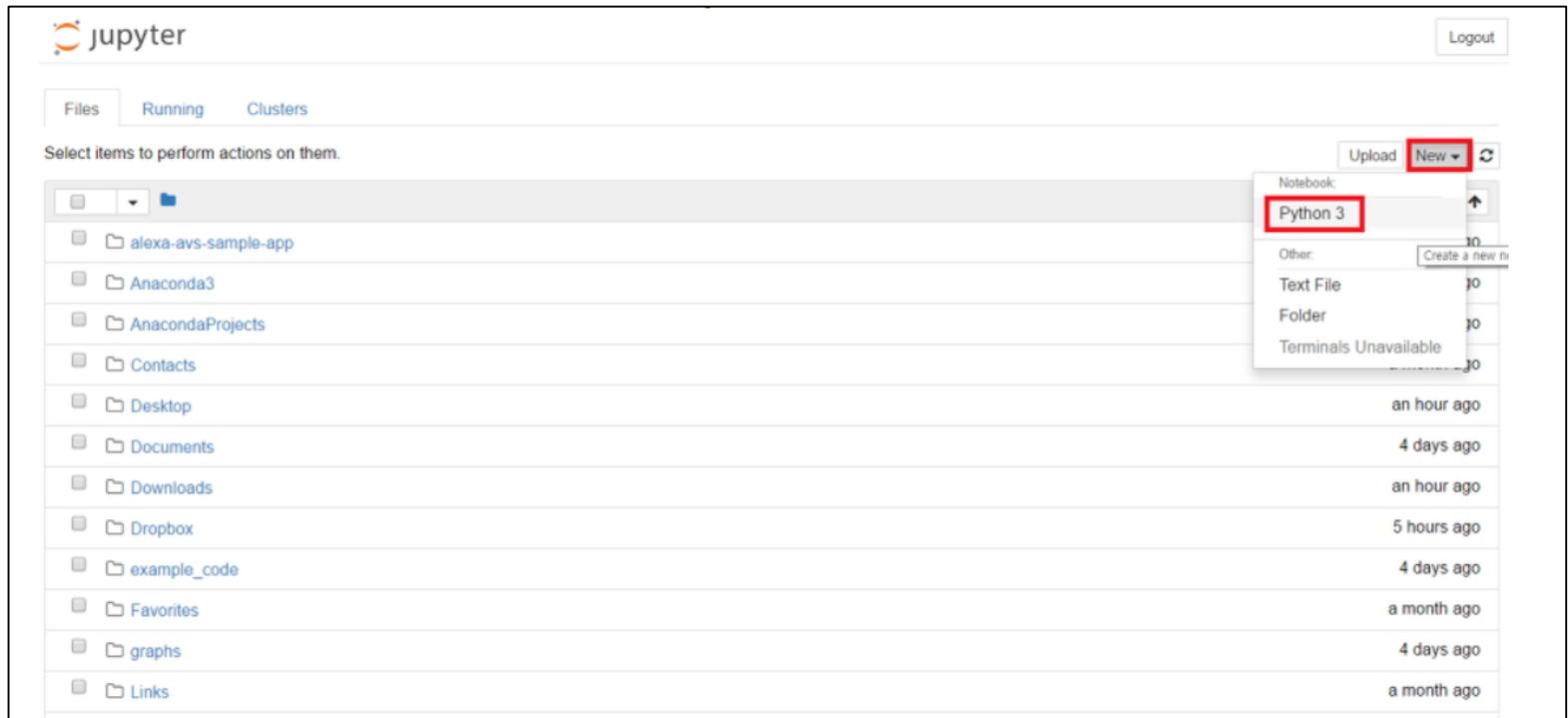
TensorBoard

install 후 **Launch** 클릭



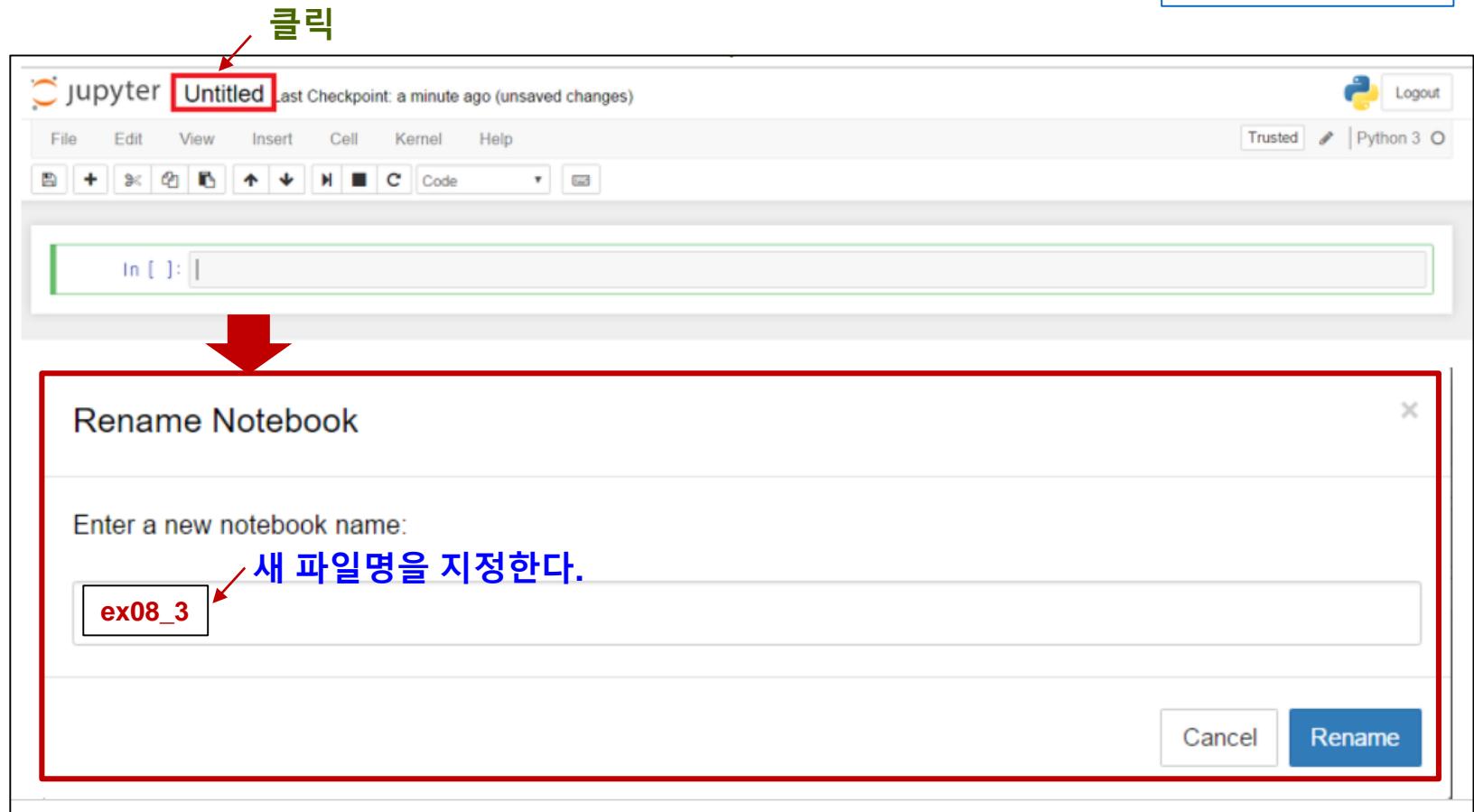
TensorBoard

jupyter가 켜진 후
오른쪽 상단에서 **New** 클릭,
Python 3 클릭



TensorBoard

ex08_3.ipynb



TensorBoard

ex08_3.ipynb

코드 입력력

```
import tensorflow as tf

a = tf.constant(5, name='a')
b = tf.constant(7, name='b')

absum = tf.add(a,b, name='absum')
a_absum_sum = tf.add(a,absum, name='a_absum_sum')

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    print(sess.run(absum))
    print(sess.run(a_absum_sum))
    writer = tf.summary.FileWriter('./graphs/ex08_3')
    writer.add_graph(sess.graph)
```

그래프를 저장할 디렉토리명 (임의의 이름으로 설정)

12

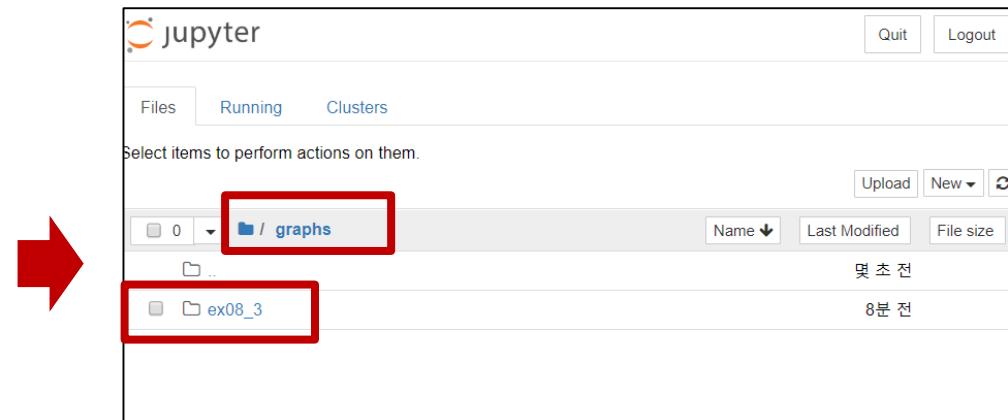
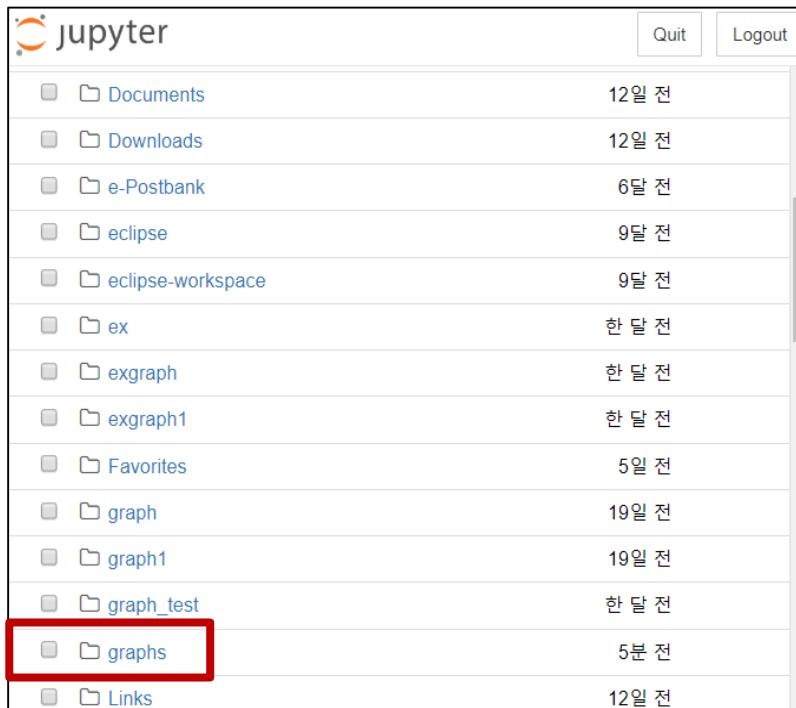
17

TensorBoard

ex08_3.ipynb

Notes

- jupyter home으로 다시 이동, **graphs/ex08_3** 디렉토리가 생성되었는지 확인
(**graphs/ex08_3** 디렉토리가 생성 안 되었다면 실행이 안된 것 → 다시 점검 필요)

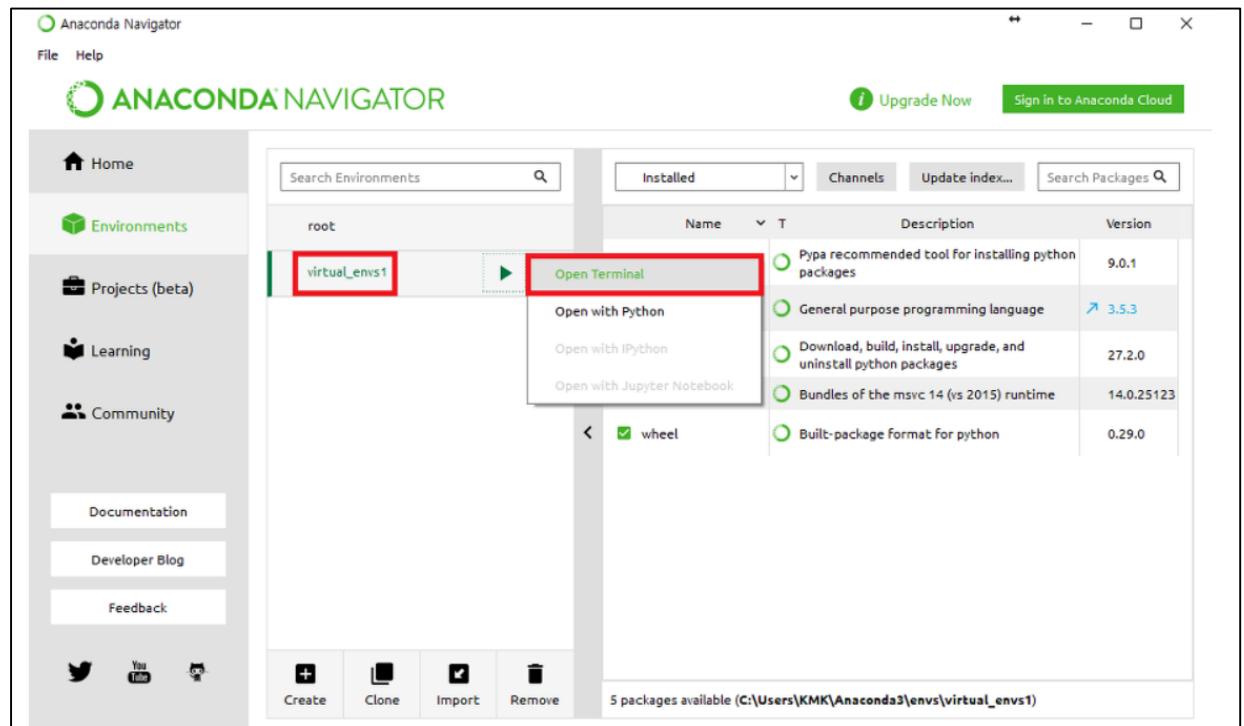


TensorBoard

ex08_3.ipynb

Notes

- 다시 Navigator로 이동
- Environments의 virtual_envs1 클릭
→ 화살표 클릭
→ Open Terminal 클릭



TensorBoard

graphs/ex08_3 디렉토리가 생성되었는지 다시 확인

```
(virtual_envs1) C:\Users\Park>cd graphs  
(virtual_envs1) C:\Users\Park\graphs>dir  
C 드라이브의 볼륨에는 이름이 없습니다.  
볼륨 일련 번호: BE2E-108E  
  
C:\Users\Park\graphs 디렉터리  
  
2018-07-07 오후 06:08 <DIR> .  
2018-07-07 오후 06:08 <DIR> ..  
2018-06-02 오전 09:30 <DIR> .ipynb_checkpoints  
2018-07-07 오후 06:01 <DIR> ex08_3  
0개 파일 0 바이트  
4개 디렉터리 2,309,861,376 바이트 남음  
  
(virtual_envs1) C:\Users\Park\graphs>
```

TensorBoard

graphs 디렉토리에서

tensorboard --logdir=". / ex08_3" --port 6006 입력

```
(virtual_envs1) C:\Users\Park\graphs\tensorboard --logdir=". / ex08_3" --port 6006  
Starting TensorBoard at 'http://127.0.0.1:6006'  
(You can navigate to http://[REDACTED]:6006)
```

드래그하여 copy (ctrl + c)

TensorBoard

ex08_3.ipynb

Notes

웹 브라우저 실행(크롬 또는 익스플로러 낮은 버전은 지원 안 됨.)

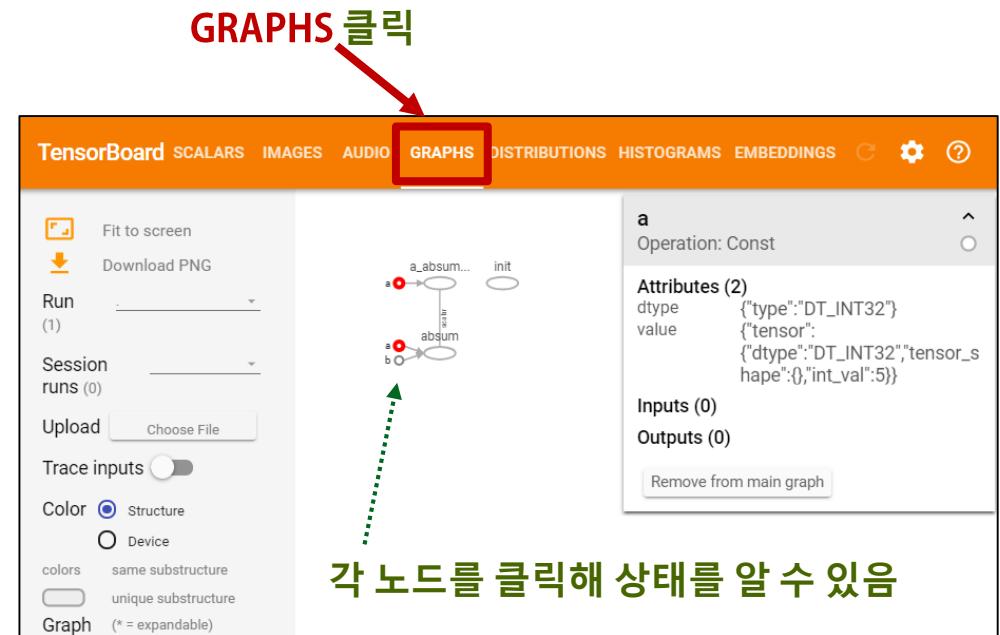
- copy한 주소를 브라우저 주소 입력창에 paste (ctrl + v) → http://***.***.*.*:6006 주소 입력한다는 뜻

```
import tensorflow as tf

a = tf.constant(5, name='a')
b = tf.constant(7, name='b')

absum = tf.add(a,b, name='absum')
a_absum_sum = tf.add(a,absum, name='a_absum_sum')

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    print(sess.run(absum))
    print(sess.run(a_absum_sum))
writer = tf.summary.FileWriter("./graphs/ex08_3")
writer.add_graph(sess.graph)
```



Old fashion: print, print, print

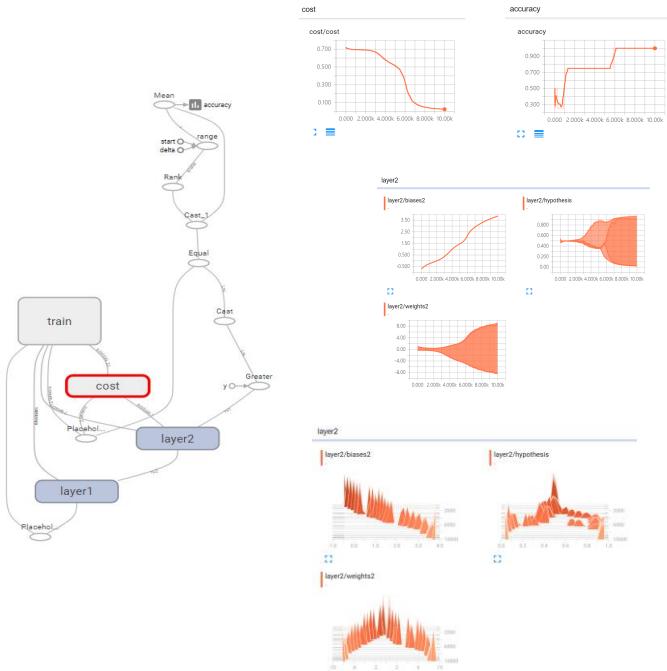
```

9400 0.0151413 [array([[ 6.21692038,  6.05913448],
   [-6.33773184, -5.75189114]], dtype=float32), array([[ 9.93581772,
   [-9.43034935]], dtype=float32)]
9500 0.014909 [array([[ 6.22498751,  6.07049847],
   [-6.34637976, -5.76352596]], dtype=float32), array([[ 9.96414757,
   [-9.45942593]], dtype=float32)]
9600 0.0146836 [array([[ 6.23292685,  6.08166742],
   [-6.35489035, -5.77496052]], dtype=float32), array([[ 9.99207973,
   [-9.48807526]], dtype=float32)]
9700 0.0144647 [array([[ 6.24074268,  6.09264851],
   [-6.36326933, -5.78619957]], dtype=float32), array([[ 10.01962471,
   [-9.51631165]], dtype=float32)]
9800 0.0142521 [array([[ 6.24843407,  6.10344648],
   [-6.37151814, -5.79724932]], dtype=float32), array([[ 10.04679298,
   [-9.54414845]], dtype=float32)]
9900 0.0140456 [array([[ 6.25601053,  6.11406422],
   [-6.3796401 , -5.80811596]], dtype=float32), array([[ 10.07359505,
   [-9.57159519]], dtype=float32)]
10000 0.0138448 [array([[ 6.26347113,  6.12451124],
   [-6.38764334, -5.81880617]], dtype=float32), array([[ 10.10004139,
   [-9.59866238]], dtype=float32)]

```



New way : TensorBoard



5 steps of using TensorBoard

- 1 어떤 tensor를 그래프로 기록할지 결정하는 단계

```
w2_hist = tf.summary.histogram("weights2", W2)
cost_summ = tf.summary.scalar("cost", cost)
```

- 2 1단계의 summary 정보를 merge 하는 단계

```
summary = tf.summary.merge_all()
```

- 3 Writer에 디렉토리를 생성하고, graph를 추가하는 단계

```
# Create summary writer
writer = tf.summary.FileWriter('./graphs/xor_tb')
writer.add_graph(sess.graph)
```

- 4 실행 단계 : 2단계의 summary merge 와 add_summary 를 실행

```
s, _ = sess.run([summary, optimizer], feed_dict=feed_dict)
writer.add_summary(s, global_step=global_step)
```

- 5 TensorBoard를 Launch 하는 단계

```
Ex) tensorboard --logdir="./graphs/xor_tb" --port 6006
```

```

import tensorflow as tf
import numpy as np
x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y_data = np.array([[0], [1], [1], [0]], dtype=np.float32)
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)

with tf.name_scope("layer1"):
    W1 = tf.Variable(tf.random_normal([2, 2]), name='weight1')
    b1 = tf.Variable(tf.random_normal([2]), name='bias1')
    layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

```

1 w1_hist = **tf.summary.histogram("weights1", W1)** # 1단계 : W1 히스토그램 그래프 결정
 b1_hist = **tf.summary.histogram("biases1", b1)** # 1단계 : b1 히스토그램 그래프 결정
 layer1_hist = **tf.summary.histogram("layer1", layer1)** # 1단계 : Layer1 히스토그램 그래프 결정

```

with tf.name_scope("layer2"):
    W2 = tf.Variable(tf.random_normal([2, 1]), name='weight2')
    b2 = tf.Variable(tf.random_normal([1]), name='bias2')
    hypothesis = tf.sigmoid(tf.matmul(layer1, W2) + b2)

```

1 w2_hist = **tf.summary.histogram("weights2", W2)** # 1단계 : W2 히스토그램 그래프 결정
 b2_hist = **tf.summary.histogram("biases2", b2)** # 1단계 : b2 히스토그램 그래프 결정
 hypothesis_hist = **tf.summary.histogram("hypothesis", hypothesis)** # 1단계 : hypothesis 히스토그램 그래프 결정

1 # cost/loss function
with tf.name_scope("cost"):
 cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
 cost_summ = tf.summary.scalar("cost", cost) # 1단계 : cost 스칼라 그래프 결정

with tf.name_scope("train"):
 train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

Accuracy computation
True if hypothesis>0.5 else False
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

1 accuracy_sum = tf.summary.scalar("accuracy", accuracy) # 1단계 : accuracy 스칼라 그래프 결정

```
# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
```

2 merged_summary = tf.summary.merge_all() # 2단계 : 1단계의 summary 정보를 merge 하는 단계

3 writer = tf.summary.FileWriter("./graphs/ex08_2_tb") # 3단계 : 디렉토리를 생성
writer.add_graph(sess.graph) # 3단계 : graph를 추가하는 단계

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

```
for step in range(10001):
```

4 # 4단계 : summary merge 실행 단계

```
summary, _ = sess.run([merged_summary, train], feed_dict={X: x_data, Y: y_data})
```

4

4단계 : add_summary 실행 단계

```
writer.add_summary(summary, global_step=step)
```

```
if step % 100 == 0:
```

```
    print("step : ", step, "cost :", sess.run(cost, feed_dict={X: x_data, Y: y_data}))
```

```
    print("W1 : \n", sess.run(W1), "\nb1: \n", sess.run(b1))
```

```
    print("Layer 1: \n", sess.run(layer1, feed_dict={X: x_data, Y: y_data}))
```

```
    print("W2 : \n", sess.run(W2), "\nb2: \n", sess.run(b2))
```

```
    print("hypothesis : \n", sess.run(hypothesis, feed_dict={X:x_data}),"\\n" , "=*50)
```

Accuracy report

```
p, a = sess.run([predicted, accuracy],feed_dict={X: x_data, Y: y_data})
print("• Correct: \\n", p, "\\n• Accuracy: \\n", a)
```

5

```
tensorboard --logdir="../ex08_2_tb" --port 6006
```



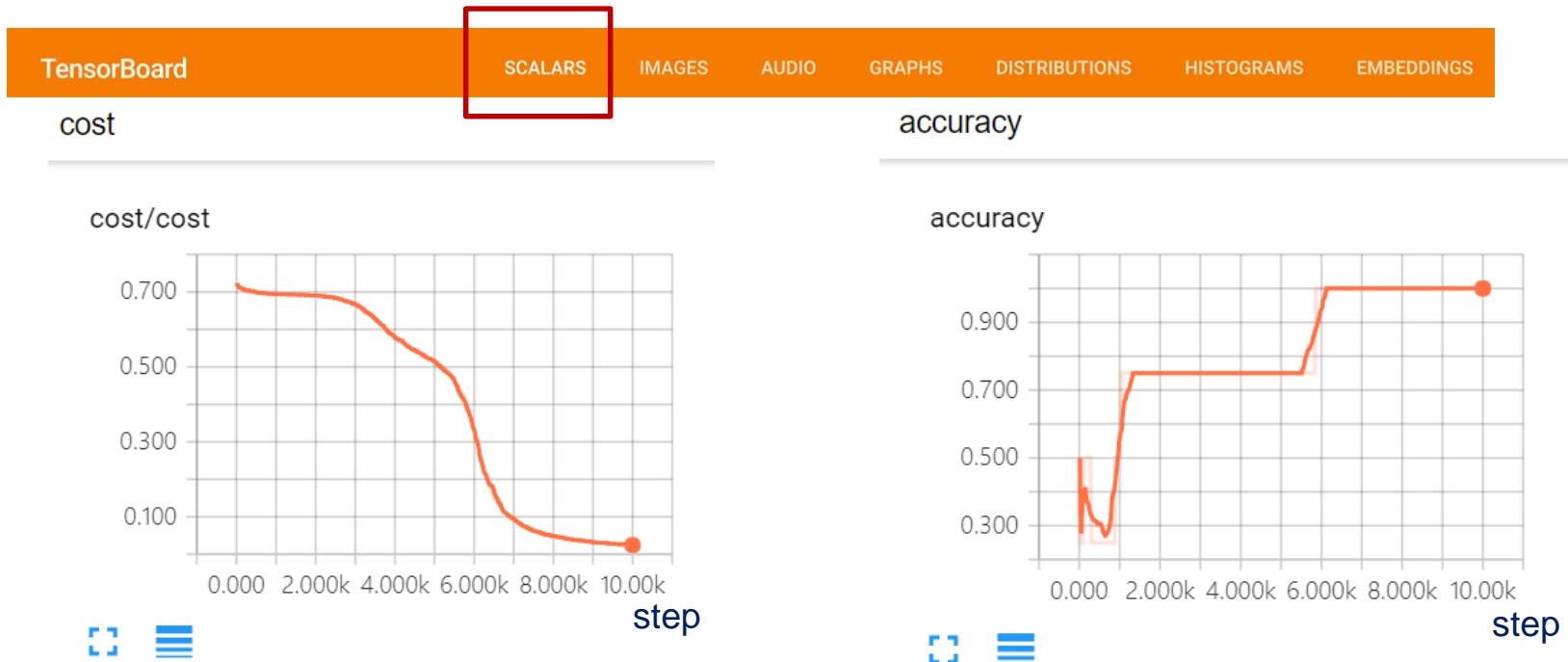
```
(virtual_envs1) C:\Users\Park\graphs>tensorboard --logdir="../ex08_2_tb" --port 6006
WARNING:tensorflow:Found more than one graph event per run, or there was a metagraph
one or more graph events. Overwriting the graph with the newest event.
Starting TensorBoard b'41' on port 6006
(You can navigate to http://[REDACTED]:6006)
```

드래그하여 copy (ctrl + c)

New way!

[ex08_2\(tb1.ipynb\)](#)

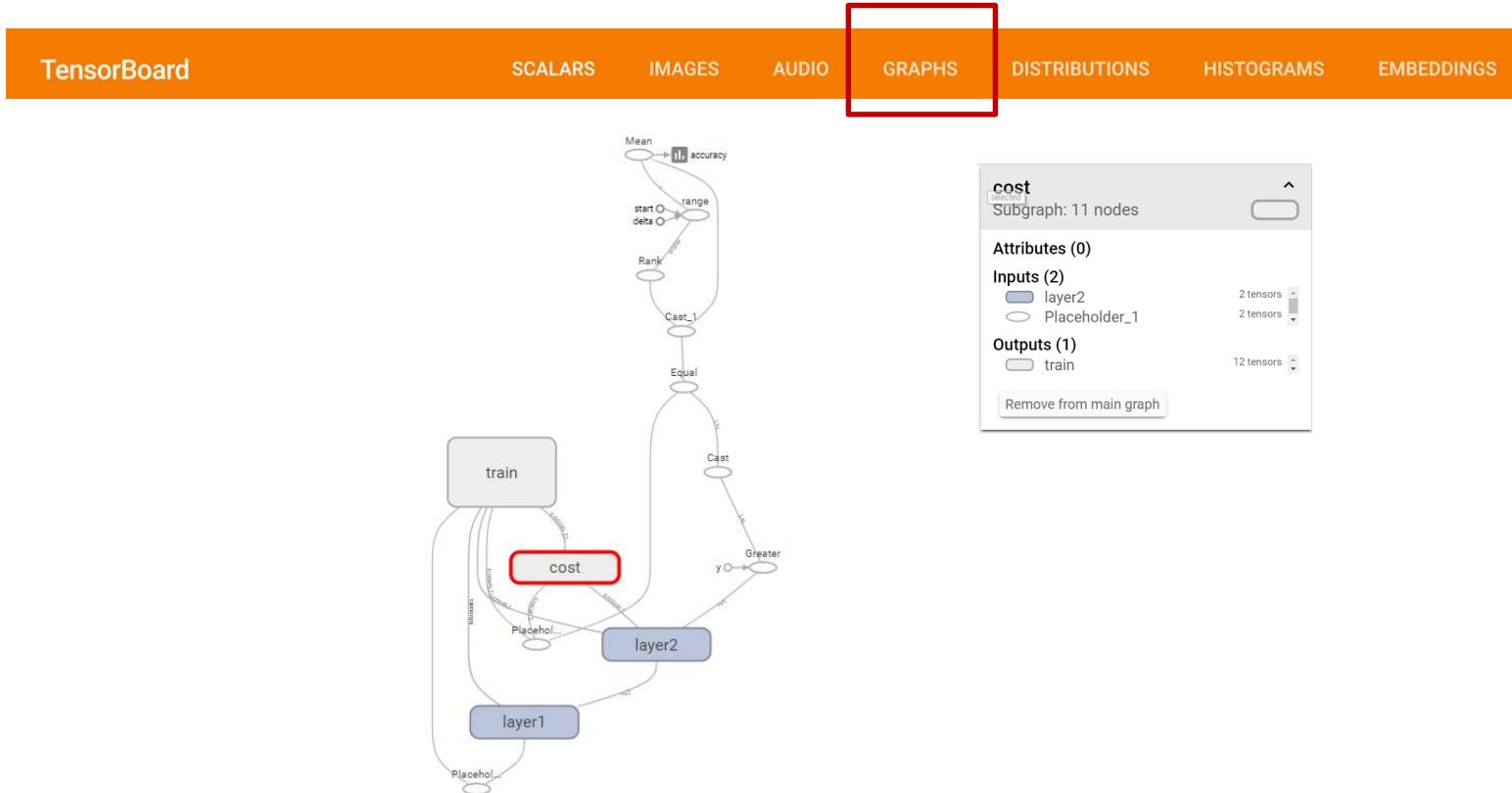
Notes



New way!

ex08_2(tb1.ipynb)

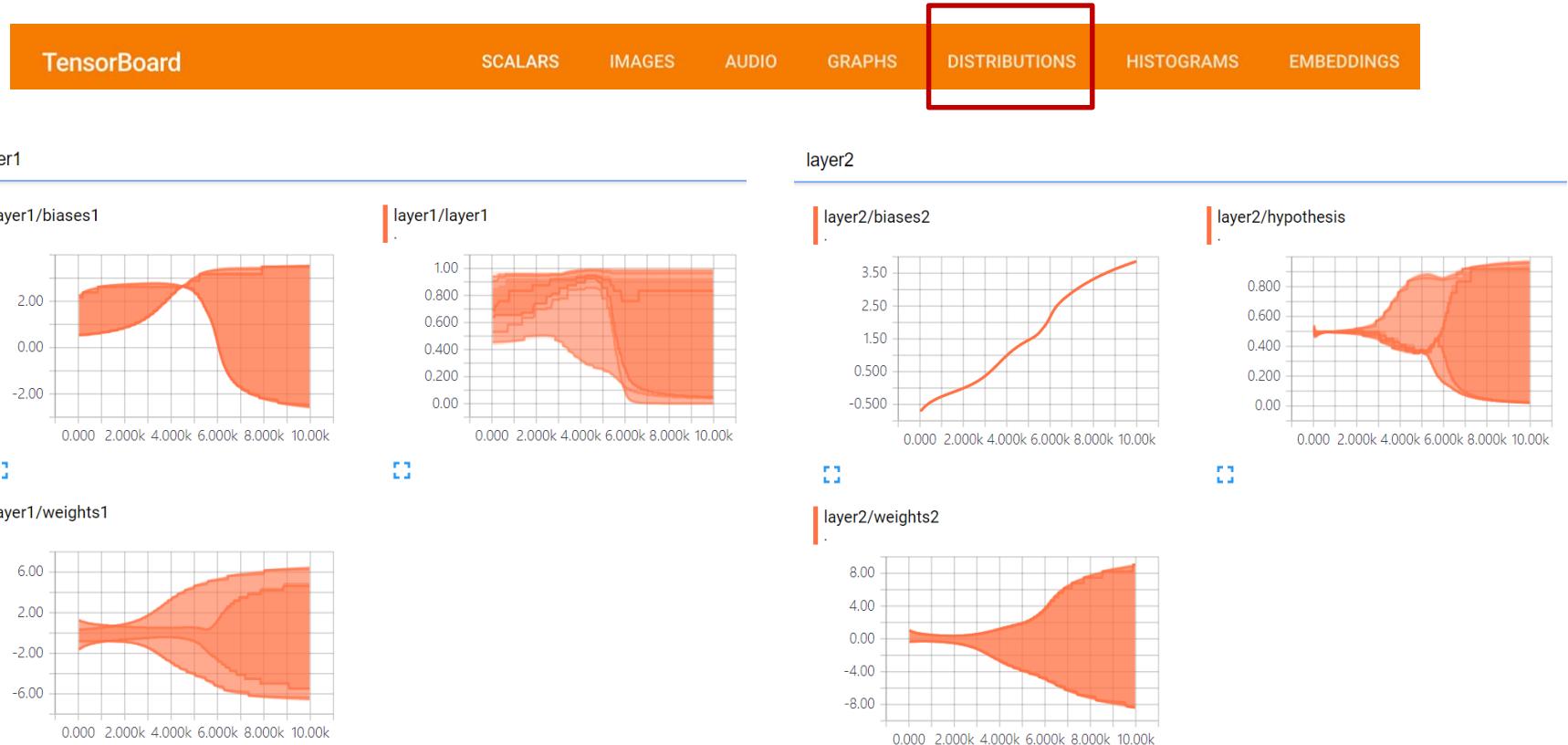
Notes



New way!

[ex08_2\(tb1.ipynb\)](#)

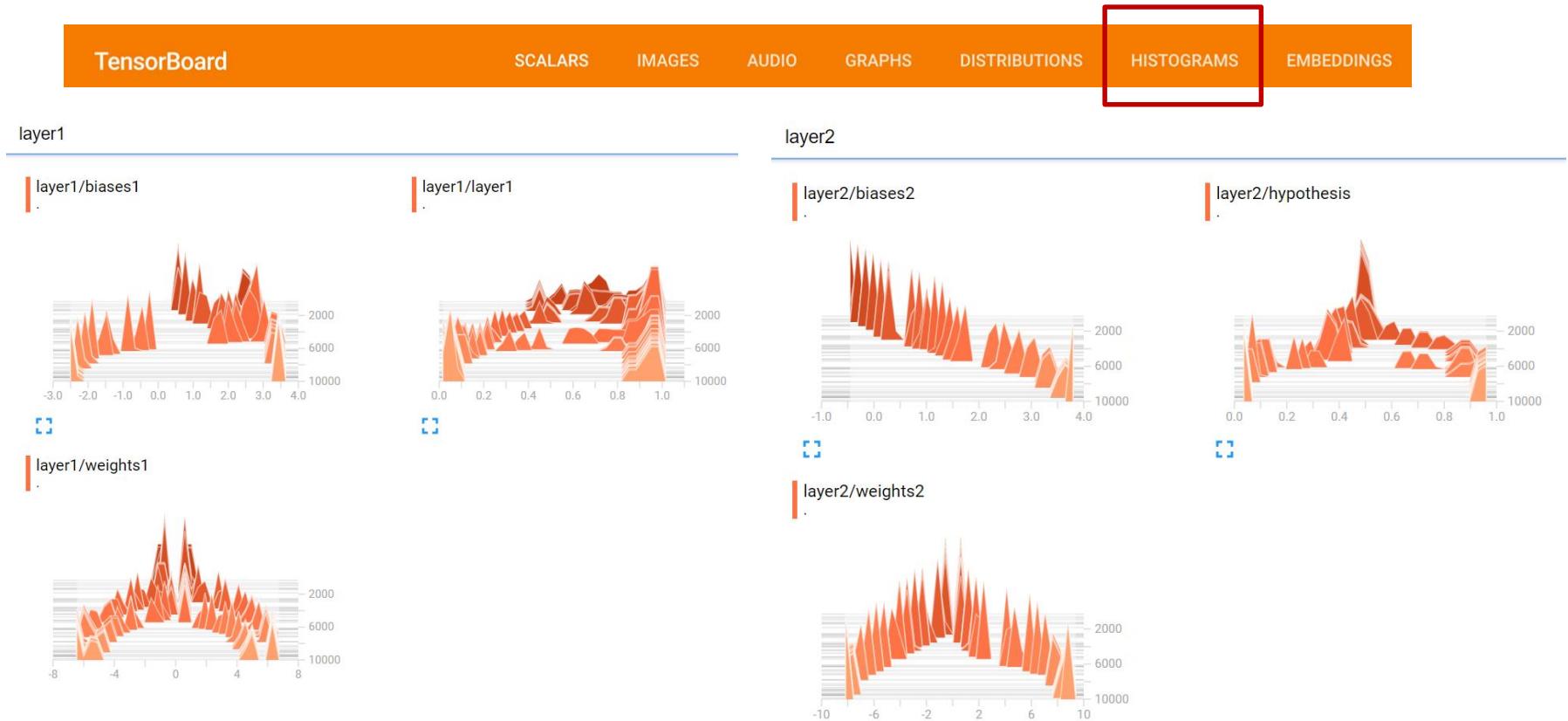
Notes



New way!

[ex08_2\(tb1.ipynb\)](#)

Notes



```

import tensorflow as tf
import numpy as np
x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y_data = np.array([[0], [1], [1], [0]], dtype=np.float32)
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)

with tf.name_scope("layer1"):
    W1 = tf.Variable(tf.random_normal([2, 2]), name='weight1')
    b1 = tf.Variable(tf.random_normal([2]), name='bias1')
    layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

```

1 w1_hist = **tf.summary.histogram("weights1", W1)** # 1단계 : W1 히스토그램 그래프 결정
 b1_hist = **tf.summary.histogram("biases1", b1)** # 1단계 : b1 히스토그램 그래프 결정
 layer1_hist = **tf.summary.histogram("layer1", layer1)** # 1단계 : Layer1 히스토그램 그래프 결정

```

with tf.name_scope("layer2"):
    W2 = tf.Variable(tf.random_normal([2, 1]), name='weight2')
    b2 = tf.Variable(tf.random_normal([1]), name='bias2')
    hypothesis = tf.sigmoid(tf.matmul(layer1, W2) + b2)

```

1 w2_hist = **tf.summary.histogram("weights2", W2)** # 1단계 : W2 히스토그램 그래프 결정
 b2_hist = **tf.summary.histogram("biases2", b2)** # 1단계 : b2 히스토그램 그래프 결정
 hypothesis_hist = **tf.summary.histogram("hypothesis", hypothesis)** # 1단계 : hypothesis 히스토그램 그래프 결정

```
# cost/loss function
with tf.name_scope("cost"):
    cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
```

1

```
cost_summ = tf.summary.scalar("cost", cost) # 1단계 : cost 스칼라 그래프 결정
```

```
with tf.name_scope("train"):
    train_a = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
    train_b = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)
```

```
# Accuracy computation
```

```
# True if hypothesis>0.5 else False
```

```
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))
```

1

```
accuracy_sum = tf.summary.scalar("accuracy", accuracy) # 1단계 : accuracy 스칼라 그래프 결정
```

```
# Launch graph
with tf.Session() as sess:
```

2

```
# Initialize TensorFlow variables
merged_summary_a = tf.summary.merge_all() # 2단계 : 1단계의 summary 정보를 merge 하는 단계
merged_summary_b = tf.summary.merge_all() # 2단계 : 1단계의 summary 정보를 merge 하는 단계
```

3

```
writer_a = tf.summary.FileWriter("./gr/a") # 3단계 : 닉도리를 생성
writer_b = tf.summary.FileWriter("./gr/b") # 3단계 : 디렉토리를 생성
writer_a.add_graph(sess.graph) # 3단계 : graph를 추가하는 단계
writer_b.add_graph(sess.graph) # 3단계 : graph를 추가하는 단계
```

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

```
for step in range(10001):
```

4 summary_a, _ = sess.run([merged_summary_a, train_a], feed_dict={X: x_data, Y: y_data}) # 4단계 : summary merge 실행 단계
writer_a.add_summary(summary_a, global_step=step) # 4단계 : add_summary 실행 단계

```
if step % 100 == 0:
```

```
    print("<< learning_rate=0.1 >> step : ", step, "\tcost :", sess.run(cost, feed_dict={X: x_data, Y: y_data}))
    print("W1 : \n", sess.run(W1), "\nb1: \n", sess.run(b1))
    print("Layer 1: \n", sess.run(layer1, feed_dict={X: x_data, Y: y_data}))
    print("W2 : \n", sess.run(W2), "\nb2: \n", sess.run(b2))
    print("hypothesis : \n", sess.run(hypothesis, feed_dict={X:x_data}),"\n" , "="*70)
```

```
p, a = sess.run([predicted, accuracy],feed_dict={X: x_data, Y: y_data})
```

```
print("• Correct: \n", p, "\n• Accuracy: \n", a)
```

```
print("★" * 35)
```

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

```
for step in range(10001):
```

4 summary_b, _ = sess.run([merged_summary_b, train_b], feed_dict={X: x_data, Y: y_data}) # 4단계 : summary merge 실행 단계
writer_b.add_summary(summary_b, global_step=step) # 4단계 : add_summary 실행 단계

```
if step % 100 == 0:
```

```
    print("<< learning_rate=0.01 >> step : ", step, "\tcost :", sess.run(cost, feed_dict={X: x_data, Y: y_data}))
    print("W1 : \n", sess.run(W1), "\nb1: \n", sess.run(b1))
    print("Layer 1: \n", sess.run(layer1, feed_dict={X: x_data, Y: y_data}))
    print("W2 : \n", sess.run(W2), "\nb2: \n", sess.run(b2))
    print("hypothesis : \n", sess.run(hypothesis, feed_dict={X:x_data}),"\n" , "="*70)
```

```
p, a = sess.run([predicted, accuracy],feed_dict={X: x_data, Y: y_data})
```

```
print("• Correct: \n", p, "\n• Accuracy: \n", a)
```

5

tensorboard --logdir=". /gr" --port 6006

```
(virtual_envs1) C:\Users\Park>cd gr  
(virtual_envs1) C:\Users\Park\gr>dir  
C 드라이브의 볼륨에는 이름이 없습니다.  
볼륨 일련 번호: BE2E-108E  
  
C:\Users\Park\gr 디렉터리  
  
2018-07-08 오후 12:58 <DIR> .  
2018-07-08 오후 12:58 <DIR> ..  
2018-07-08 오후 12:58 <DIR> a  
2018-07-08 오후 12:58 <DIR> b  
    0개 파일          0 바이트  
    4개 디렉터리   3,749,269,504 바이트 남음  
  
(virtual_envs1) C:\Users\Park\gr>cd..
```

```
(virtual_envs1) C:\Users\Park>tensorboard --logdir=". /gr" --port 6006  
Starting TensorBoard b'41' on port 6006  
(You can navigate to http://[REDACTED]:6006)
```

드래그하여 copy (ctrl + c)

ex08_2.ipynb

Notes

```
<< learning_rate=0.1 >> step : 0    cost : 0.6969122
W1 :
[[ 0.57145184 -1.9287304 ]
[-0.36804724 -0.20055488]]
b1:
[-0.19053441 -0.8570321 ]
Layer 1:
[[0.45250997 0.2979598 ]
[0.36387572 0.25777087]
[0.59409434 0.05809841]
[0.5032175 0.04804785]]
W2 :
[[ 1.594015 ]
[-0.37062126]]
b2:
[-0.8315152]
hypothesis :
[[0.44506323]
[0.4141035 ]
[0.5234696 ]
[0.48820558]]
=====
<< learning_rate=0.1 >> step : 100   cost : 0.69377667
W1 :
[[ 0.5053635 -1.9310143 ]
[-0.35291854 -0.2753199 ]]
b1:
[-0.19600268 -0.8664619 ]
Layer 1:
[[0.45115557 0.29599103]
[0.36611474 0.24199337]
[0.5767293 0.0574607 ]
[0.48911226 0.04424345]]
W2 :
[[ 1.6182425]
[-0.3585805]]
b2:
[-0.72064316]
hypothesis :
[[0.47584367]
[0.44646725]
[0.5478631 ]
[0.5137451 ]]
.
```

```
<< learning_rate=0.1 >> step : 10000 cost : 0.015991995
W1 :
[[ -6.366684  4.6767774]
[-6.6102934  4.6733856]]
b1:
[ 2.6041815 -7.2471156]
Layer 1:
[[9.3113023e-01 7.1171887e-04]
[1.7878572e-02 7.0848368e-02]
[1.7421037e-02 7.1071967e-02]
[2.3871640e-05 8.9119893e-01]]
W2 :
[[ -9.980243]
[-10.138073]]
b2:
[4.935925]
hypothesis :
[[0.01256499]
[0.9826936 ]
[0.9827328 ]
[0.01631287]]
=====
● Correct:
[[0.]
[1.]
[1.]
[0.]]
● Accuracy:
1.0
*****
<< learning_rate=0.01 >> step : 0    cost : 0.7752353
W1 :
[[ 1.4691817  0.649985 ]
[ 0.4234174 -0.5053114]]
b1:
[ 0.20487474 1.7478535 ]
Layer 1:
[[0.5510403  0.8516818 ]
[0.6521021  0.77600616]
[0.84211594 0.91666234]
[0.8906574  0.86904335]]
```

```
W2 :
[[0.44440997]
[0.45067126]]
b2:
[0.1143962]
hypothesis :
[[0.6776761 ]
[0.6800324 ]
[0.71130985]
[0.71133286]]
=====
<< learning_rate=0.01 >> step : 10000 cost : 0.67262733
W1 :
[[ 1.787963  0.7047492]
[ 1.2240211 -0.570787 ]]
b1:
[ 0.2386115 1.725391 ]
Layer 1:
[[0.5593714  0.8488219 ]
[0.811935  0.7603508 ]
[0.88355917 0.91909695]
[0.9626945  0.8652215 ]]
W2 :
[[ 0.99129164]
[-0.4037459 ]]
b2:
[-0.4383457]
hypothesis :
[[0.44360256]
[0.51487815]
[0.5166033 ]
[0.5415628 ]]
=====
● Correct:
[[0.]
[1.]
[1.]
[1.]]
● Accuracy:
0.75
```

Multiple runs

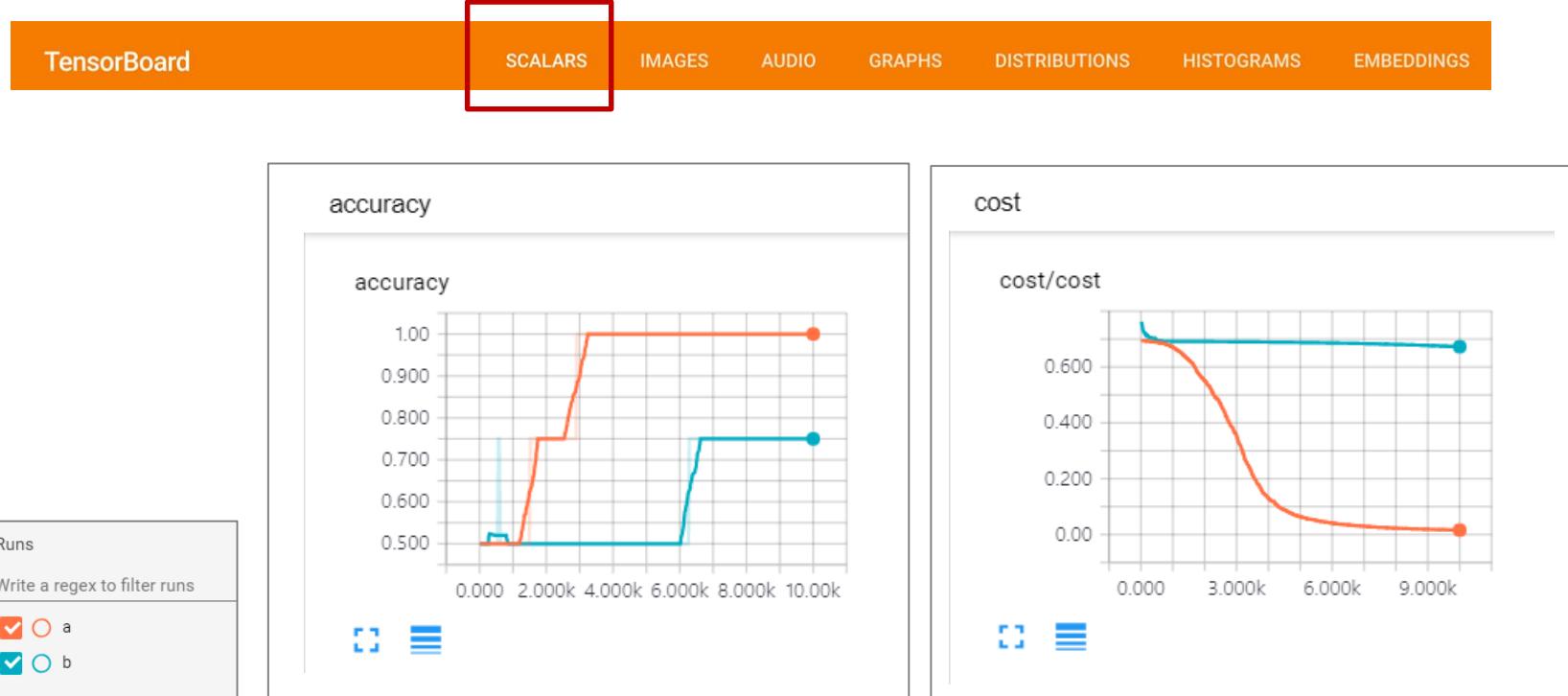
a

learning_rate=0.1 vs learning_rate=0.01

b

ex08_2(tb2).ipynb

Notes



Multiple runs

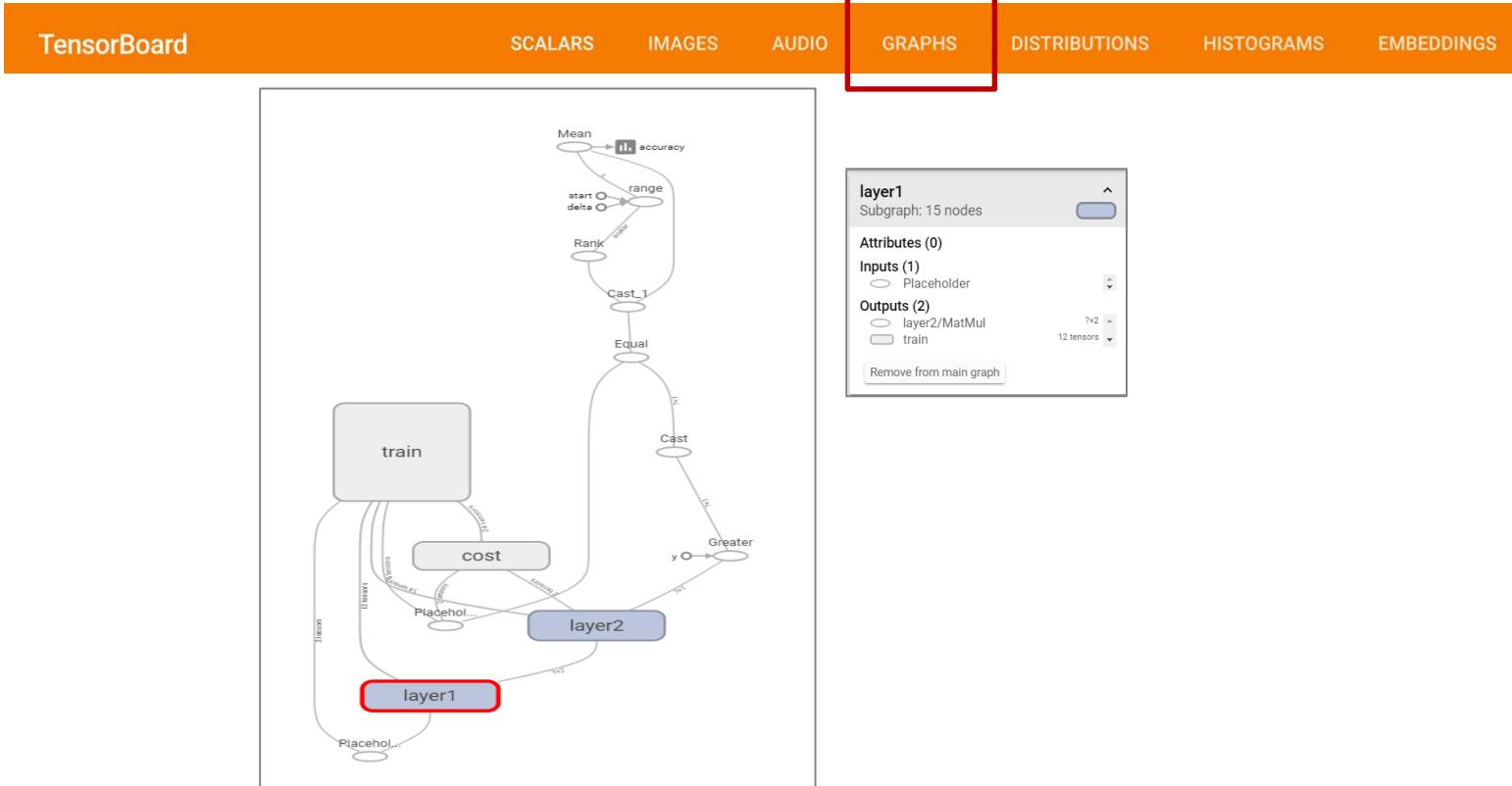
a

learning_rate=0.1 vs learning_rate=0.01

b

ex08_2(tb2).ipynb

Notes

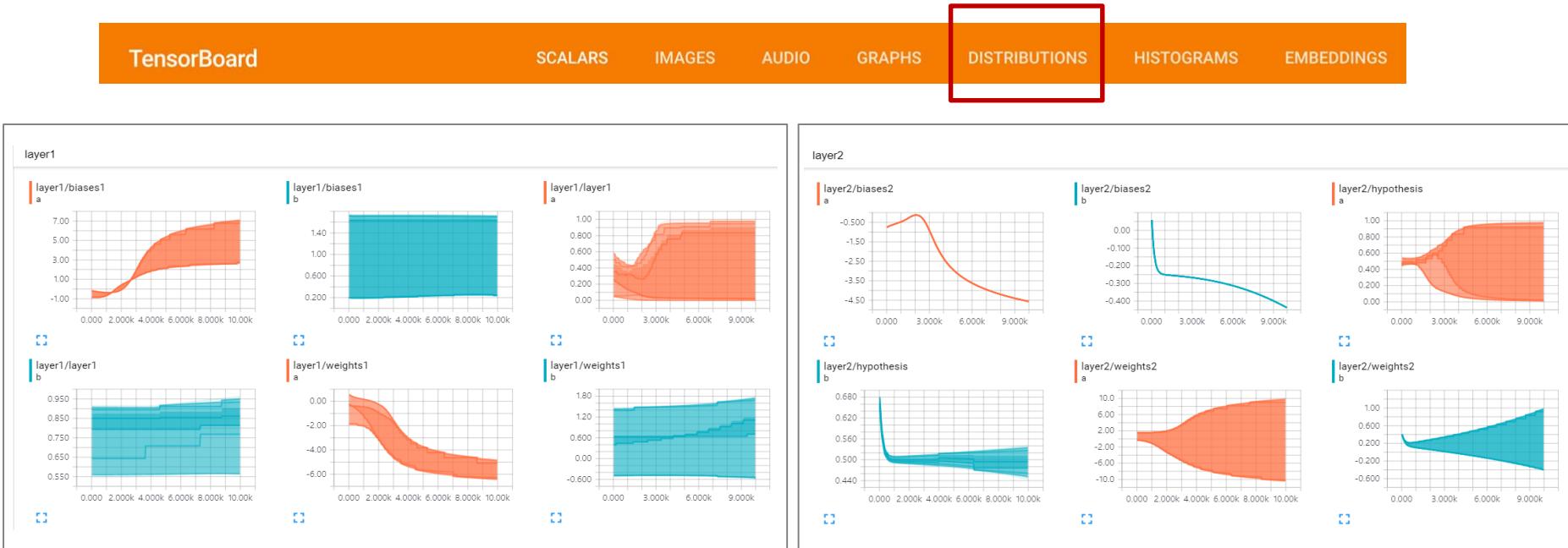


Multiple runs

a learning_rate=0.1 vs b learning_rate=0.01

ex08_2(tb2).ipynb

Notes



Runs

Write a regex to filter runs

<input checked="" type="checkbox"/> a	<input type="radio"/> b
<input checked="" type="checkbox"/> b	<input type="radio"/> a

Multiple runs

a

learning_rate=0.1 vs learning_rate=0.01

b

ex08_2(tb2).ipynb

Notes

TensorBoard

SCALARS

IMAGES

AUDIO

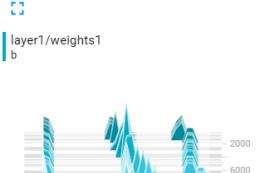
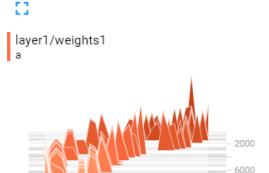
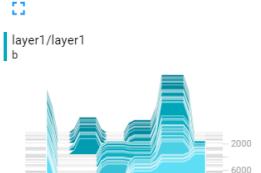
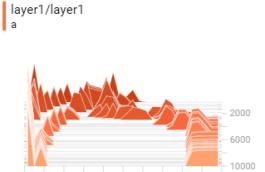
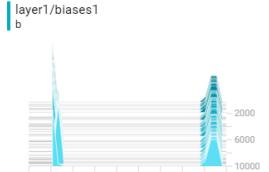
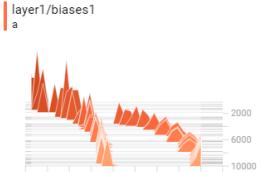
GRAPHS

DISTRIBUTIONS

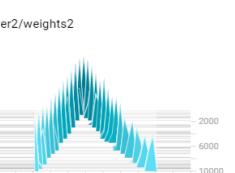
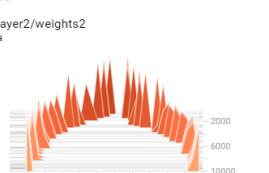
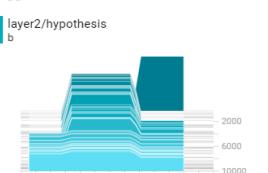
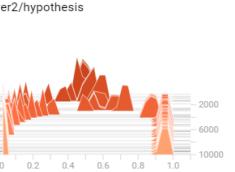
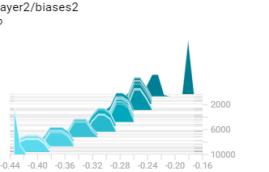
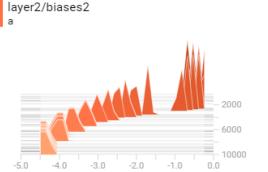
HISTOGRAMS

EMBEDDINGS

layer1



layer2



Runs

Write a regex to filter runs

 a b