UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

13강: 케라스와 신경망

인공지능 일반강좌 : 기계학습의 이해(L2-1)

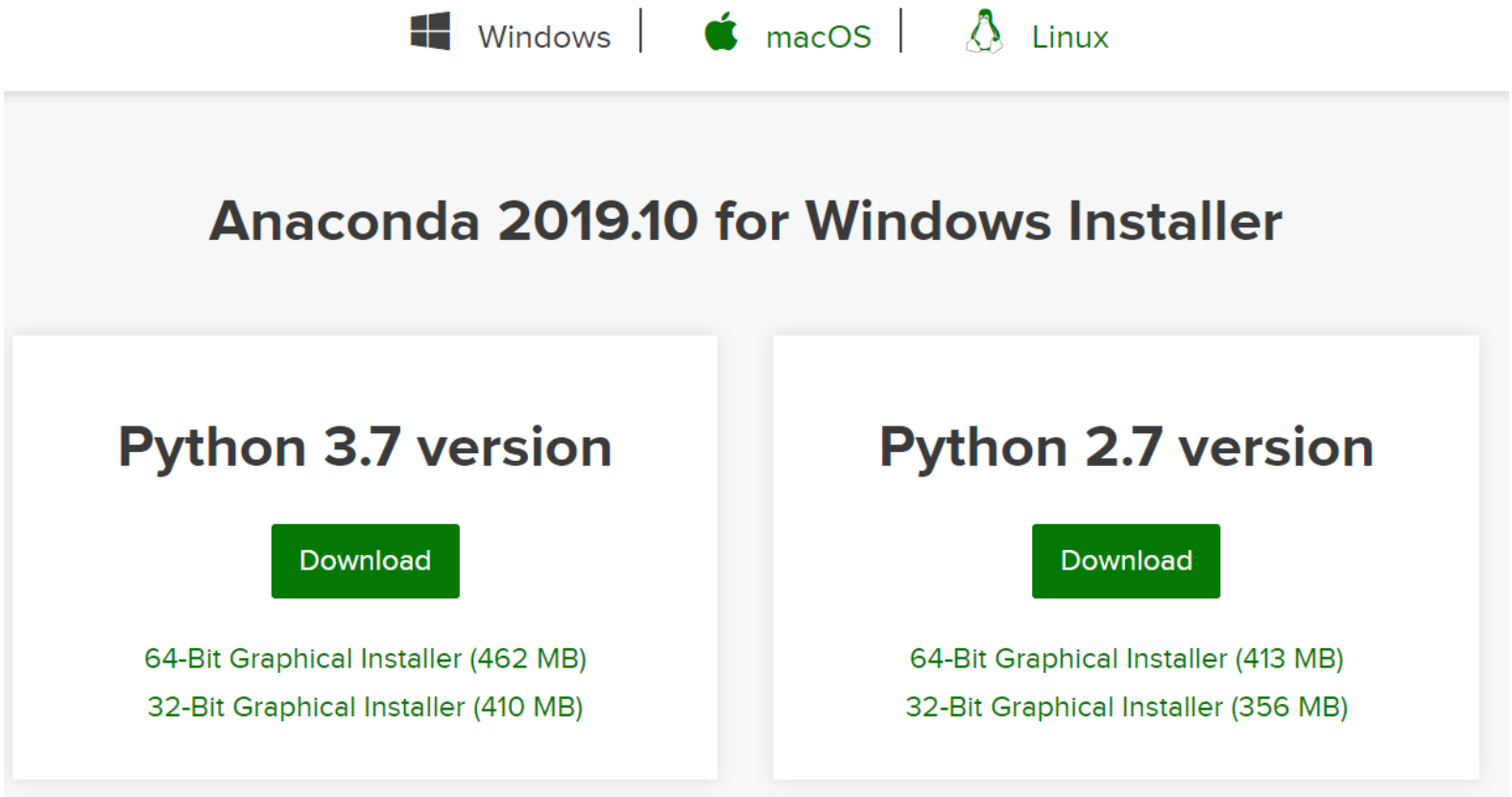
Contents

01.
텐서플로우 2.0 설치

이홍석 (hsyi@kisti.re.kr)

www.ust.ac.kr

아나콘다를 이용한 설치(1)



아나콘다를 이용한 설치(2)

텐서플로우 설치 사양 확인

System requirements

- Python 3.4–3.7
- pip 19.0 or later (requires `manylinux2010` support)
- Ubuntu 16.04 or later (64-bit)
- macOS 10.12.6 (Sierra) or later (64-bit) *(no GPU support)*
- Windows 7 or later (64-bit) *(Python 3 only)*
- Raspbian 9.0 or later
- GPU support requires a CUDA®-enabled card *(Ubuntu and Windows)*

아나콘다를 이용한 설치(3)

- 가상환경에 들어오기
  - ✓ 가상 환경 확인하기, base 만 있음.
  - ✓ 텐서플로우 2를 위한 가상환경 설정

```
(base) C:₩Users₩master>conda env list
# conda environments:
#
base                  *  C:₩Users₩master₩Anaconda3
```

```
(base) C:₩Users₩master>conda create --name tf2 python=3.7 pip
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:₩Users₩master₩Anaconda3₩envs₩tf2

  added / updated specs:
    - pip
    - python=3.7
```

아나콘다를 이용한 설치(4)

```
Downloading and Extracting Packages
python-3.7.6          | 14.8 MB   | ############################
pip-20.0.2            | 1.7 MB    | ############################
certifi-2019.11.28    | 154 KB    | ############################
vs2015_runtime-14.16  | 1.1 MB    | ############################
wheel-0.34.1          | 65 KB     | ############################
openssl-1.1.1d        | 4.8 MB    | ############################
setuptools-45.1.0     | 527 KB    | ############################
ca-certificates-2020  | 125 KB    | ############################
sqlite-3.30.1         | 627 KB    | ############################
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate tf2
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

아나콘다를 이용한 설치(5)

```
(base) C:₩Users₩master>conda activate tf2

(tf2) C:₩Users₩master>conda list
# packages in environment at C:₩Users₩master₩Anaconda3₩envs₩tf2:
#
# Name                    Version                   Build  Channel
ca-certificates           2020.1.1                      0
certifi                   2019.11.28                py37_0
openssl                   1.1.1d                he774522_3
pip                       20.0.2                    py37_1
python                    3.7.6                 h60c2a47_2
setuptools                45.1.0                    py37_0
sqlite                    3.30.1                he774522_0
vc                        14.1                  h0510ff6_4
vs2015_runtime            14.16.27012           hf0eaf9b_1
wheel                     0.34.1                    py37_0
wincertstore              0.2                       py37_0

(tf2) C:₩Users₩master>
```

아나콘다를 이용한 설치(6)

- 자동으로 최신 텐서플로우 2.0 GPU 버전 설치

```
(base) C:₩Users₩master>conda activate tf2gpu

(tf2gpu) C:₩Users₩master>pip install --ignore-installed --upgrade tensorflow-gpu==2.0
Collecting tensorflow-gpu==2.0
  Using cached tensorflow_gpu-2.0.0-cp37-cp37m-win_amd64.whl (285.3 MB)
```

```
Successfully built absl-py
Installing collected packages: numpy, six, h5py, keras-applications, tensorflow-estimator, gast, ke
ras-preprocessing, setuptools, markdown, pyasn1, rsa, cachetools, pyasn1-modules, google-auth, absl
-py, chardet, idna, urllib3, certifi, requests, grpcio, oauthlib, requests-oauthlib, google-auth-oa
uthlib, werkzeug, protobuf, wheel, tensorboard, google-pasta, wrapt, termcolor, opt-einsum, scipy,
astor, tensorflow
Successfully installed absl-py-0.9.0 astor-0.8.1 cachetools-4.0.0 certifi-2019.11.28 chardet-3.0.4
gast-0.2.2 google-auth-1.11.0 google-auth-oauthlib-0.4.1 google-pasta-0.1.8 grpcio-1.26.0 h5py-2.10
.0 idna-2.8 keras-applications-1.0.8 keras-preprocessing-1.1.0 markdown-3.1.1 numpy-1.18.1 oauthlib
-3.1.0 opt-einsum-3.1.0 protobuf-3.11.2 pyasn1-0.4.8 pyasn1-modules-0.2.8 requests-2.22.0 requests-
oauthlib-1.3.0 rsa-4.0 scipy-1.4.1 setuptools-45.1.0.post20200127 six-1.14.0 tensorboard-2.1.0 tens
orflow-2.1.0 tensorflow-estimator-2.1.0 termcolor-1.1.0 urllib3-1.25.8 werkzeug-0.16.1 wheel-0.34.2
 wrapt-1.11.2
```

아나콘다를 이용한 설치(7)

- 필수적인 라이브러리 설치

```
(tf2) C:₩Users₩master>
(tf2) C:₩Users₩master>pip install seaborn pandas matplotlib graphviz jupyter notebook
Collecting seaborn
  Downloading seaborn-0.10.0-py3-none-any.whl (215 kB)
     |████████████████████████████████| 215 kB 211 kB/s
Collecting pandas
  Downloading pandas-1.0.0-cp37-cp37m-win_amd64.whl (9.0 MB)
     |████████████████████████████████| 9.0 MB 819 kB/s
Collecting matplotlib
```

- 테스트하기
  - ✓ jupyter notebook

```
(tf2) C:₩Users₩master>jupyter notebook
```

아나콘다를 이용한 설치(8)

버전확인 1 : conda list

```
(tf2) C:₩Users₩user>
(tf2) C:₩Users₩user>conda list
# packages in environment at C:₩Users₩user₩Anaconda3₩envs₩tf2:
#
# Name                    Version                   Build  Channel
ca-certificates           2019.11.28           hecc5488_0    conda-forge
certifi                   2019.11.28               py38_0    conda-forge
openssl                   1.1.1d               hfa6e2cd_0    conda-forge
pip                       20.0.2                   py38_0    conda-forge
python                    3.8.1                he1f5543_1    conda-forge
setuptools                45.1.0                   py38_0    conda-forge
sqlite                    3.30.1               hfa6e2cd_0    conda-forge
vc                        14.1                 h0510ff6_4
vs2015_runtime            14.16.27012          hf0eaf9b_1
wheel                     0.33.6                   py38_0    conda-forge
wincertstore              0.2                   py38_1003    conda-forge

(tf2) C:₩Users₩user>
```

아나콘다를 이용한 설치(9)

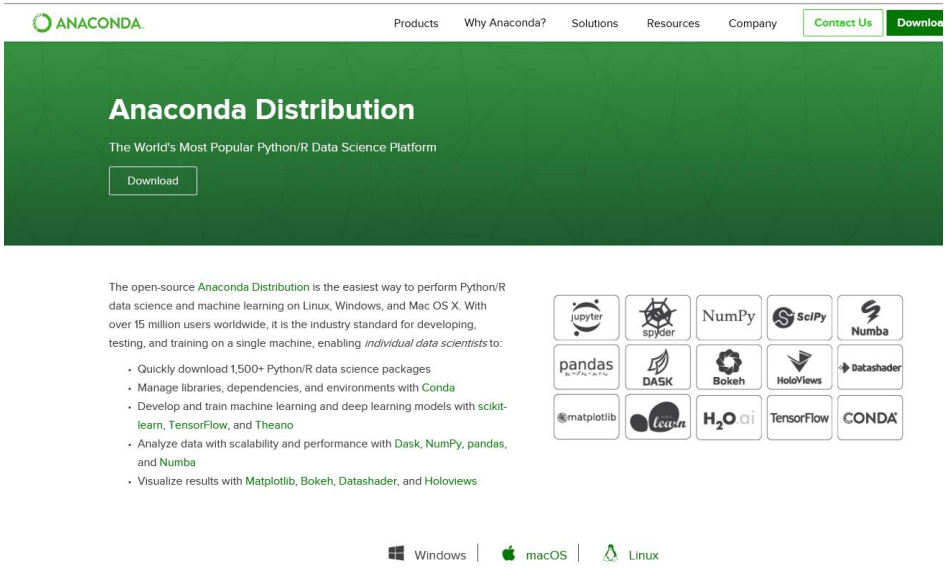
- **GPU 버전 관련 에러 발생할 수도 있음**
  - ✓ **Tensorflow 2.0에서는 CUDA 툴킷 라이브러 버전 확인**
    - 최신 CUDA는(2019년 2월 이후) cudart64_101.dll을 제공함
    - 하지만, Tensorflow 2.0은 cudart64_100.dll을 요구함
    - 해결방법은 cudart64_100.dll을 다운받아서 아래 위치에 저장함

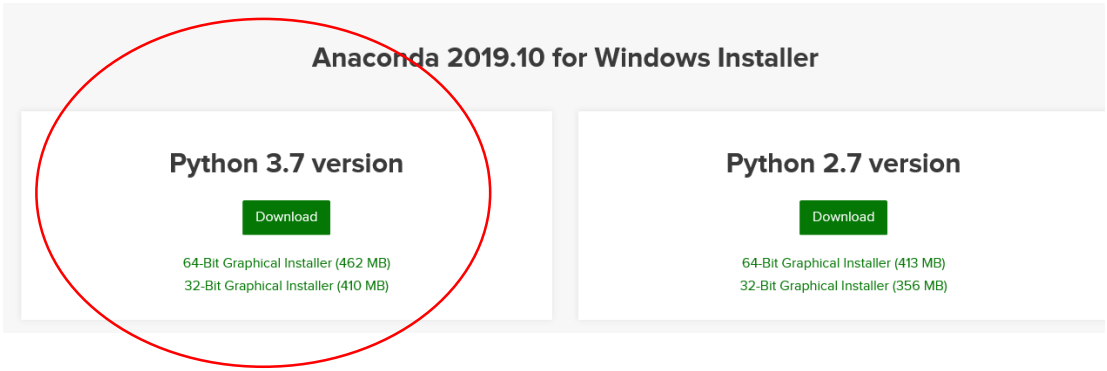
C:₩Program Files₩NVIDIA GPU Computing Toolkit₩CUDA₩v10.1₩bin

```
[I 16:10:02.149 NotebookApp] Kernel started: 60c90c3f-7d12-46f6-8d5b-bd06315a1761
2020-01-31 16:10:14.970263: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load d
ynamic library 'cudart64_100.dll'; dlerror: cudart64_100.dll not found
```

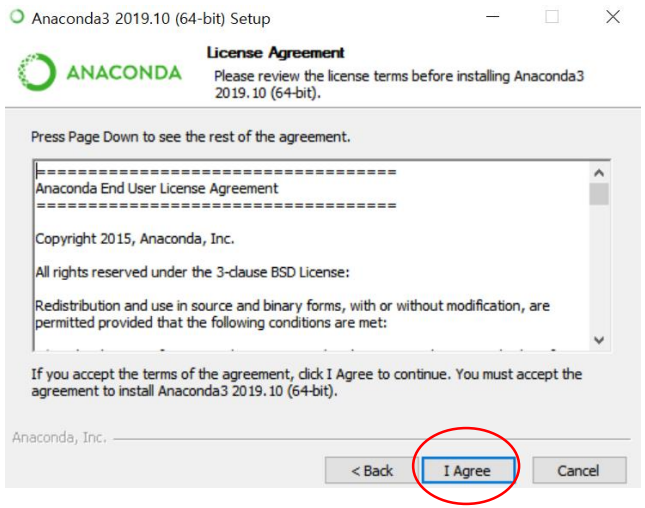
```
(tf2gpu) C:₩Users₩master₩work₩ust-hands-on₩ko>jupyter notebook
[I 16:18:39.890 NotebookApp] Serving notebooks from local directory: C:₩Users₩master₩work₩ust-hands-on₩ko
[I 16:18:39.891 NotebookApp] The Jupyter Notebook is running at:
[I 16:18:39.893 NotebookApp] http://localhost:8888/?token=038352f53bd341d4b97dc857fb036ab8effb9bb8045d669e
[I 16:18:39.894 NotebookApp]  or http://127.0.0.1:8888/?token=038352f53bd341d4b97dc857fb036ab8effb9bb8045d66
9e
[I 16:18:39.895 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip conf
irmation).
```

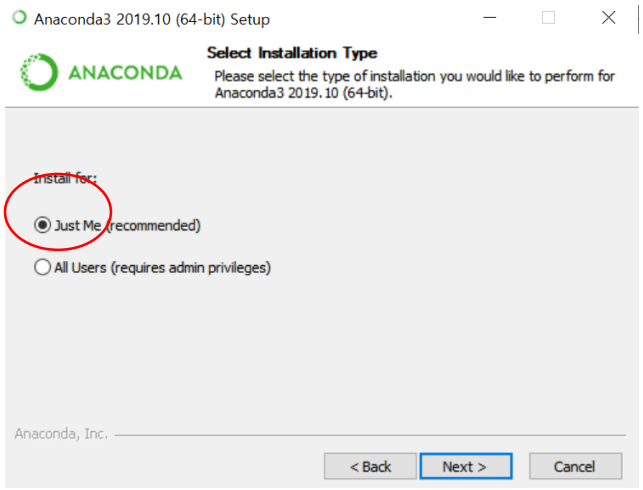
Navigator를 이용한 설치(1)

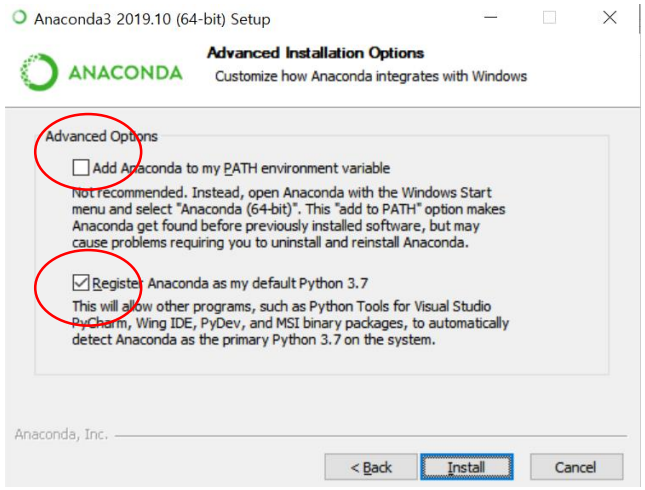


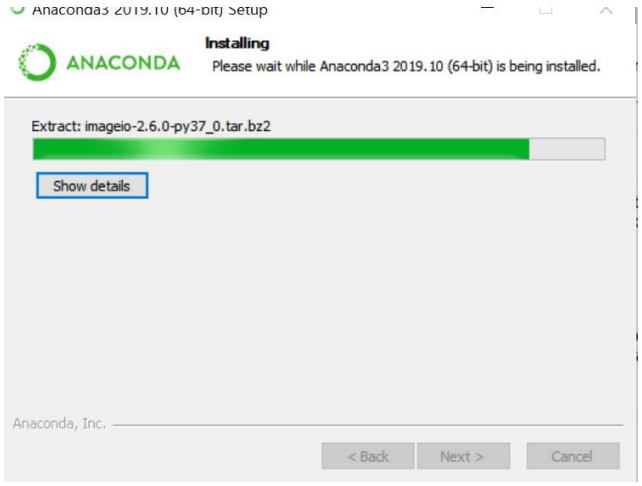


Navigator를 이용한 설치(2)

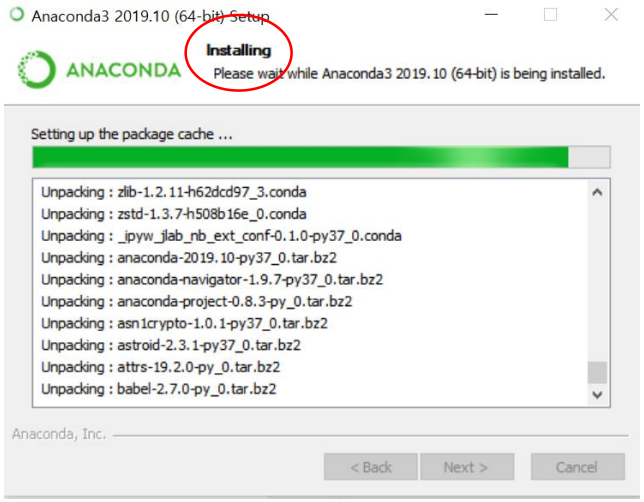


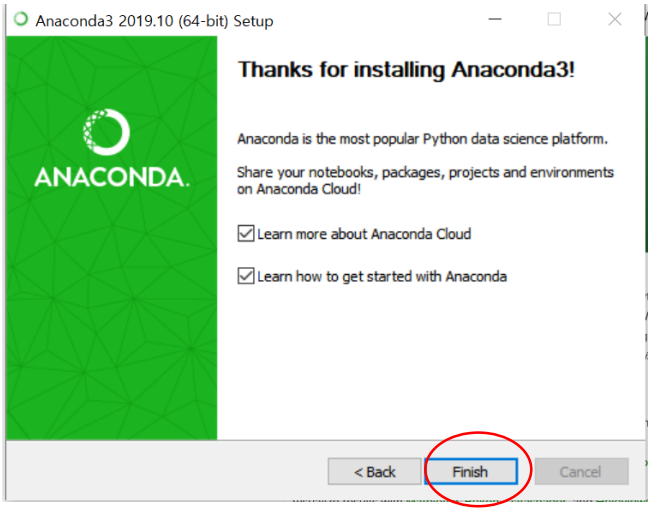




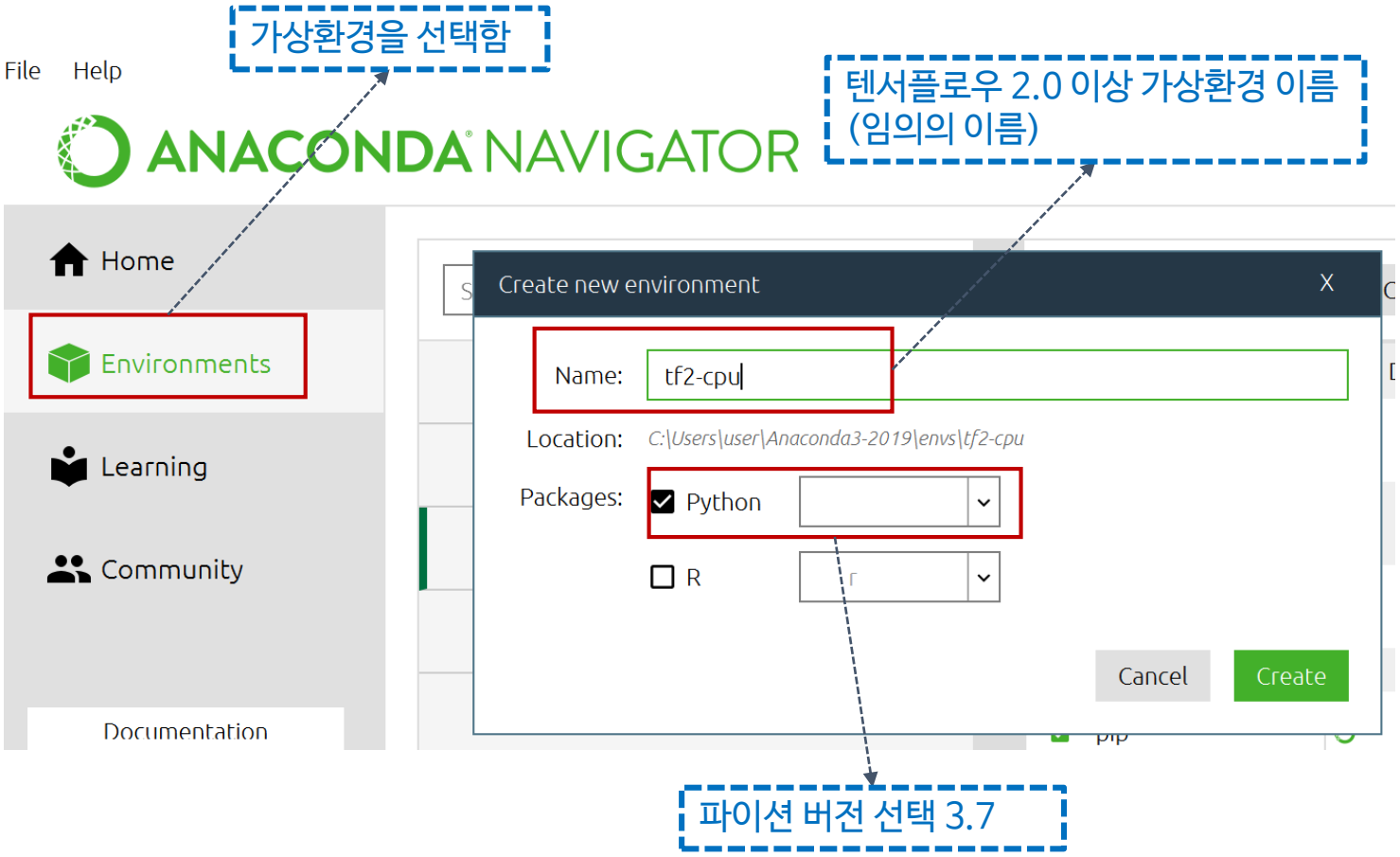


Navigator를 이용한 설치(3)

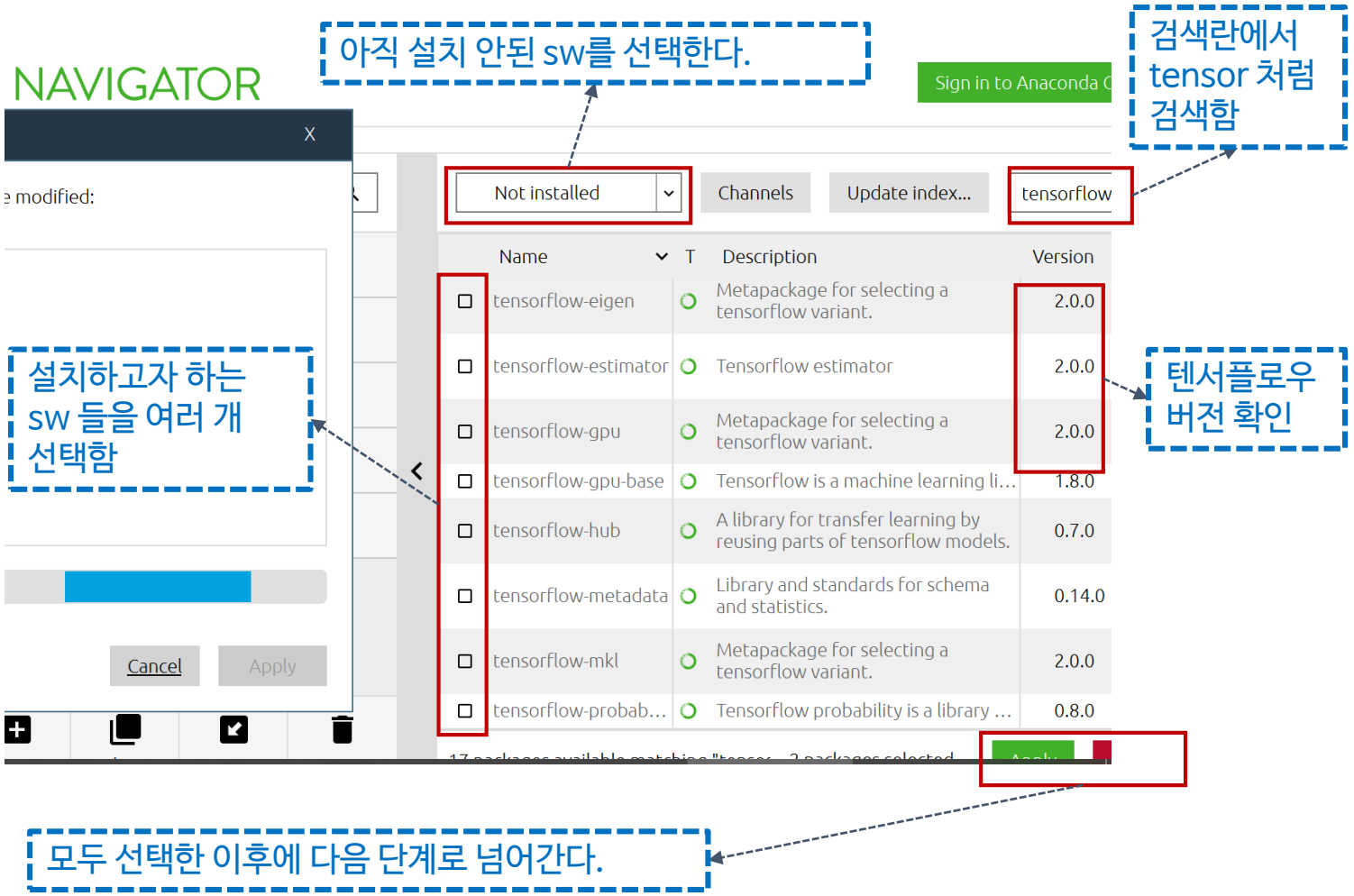




Navigator를 이용한 설치(5)

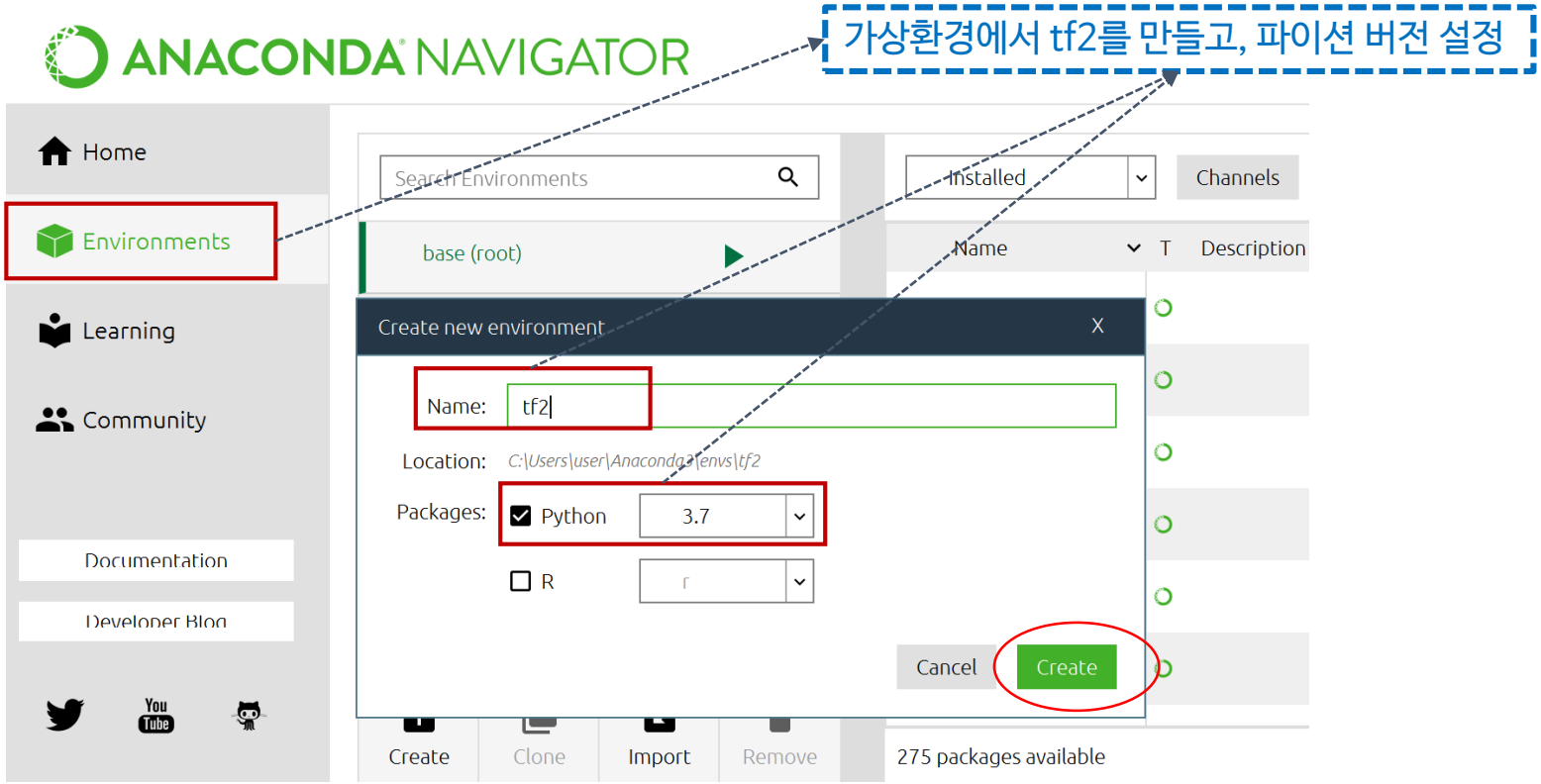


Navigator를 이용한 설치(6)

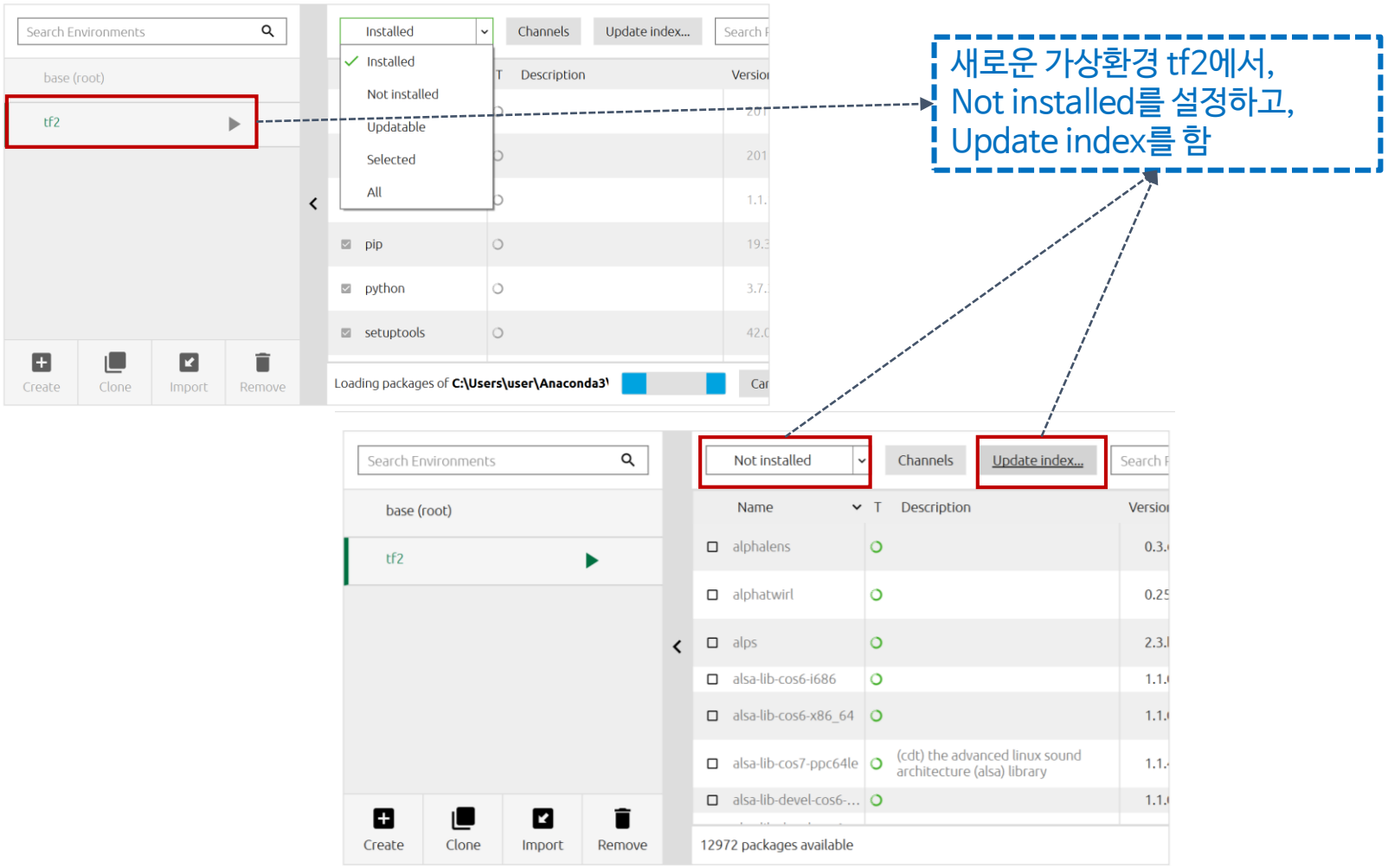


Navigator를 이용한 설치(7)

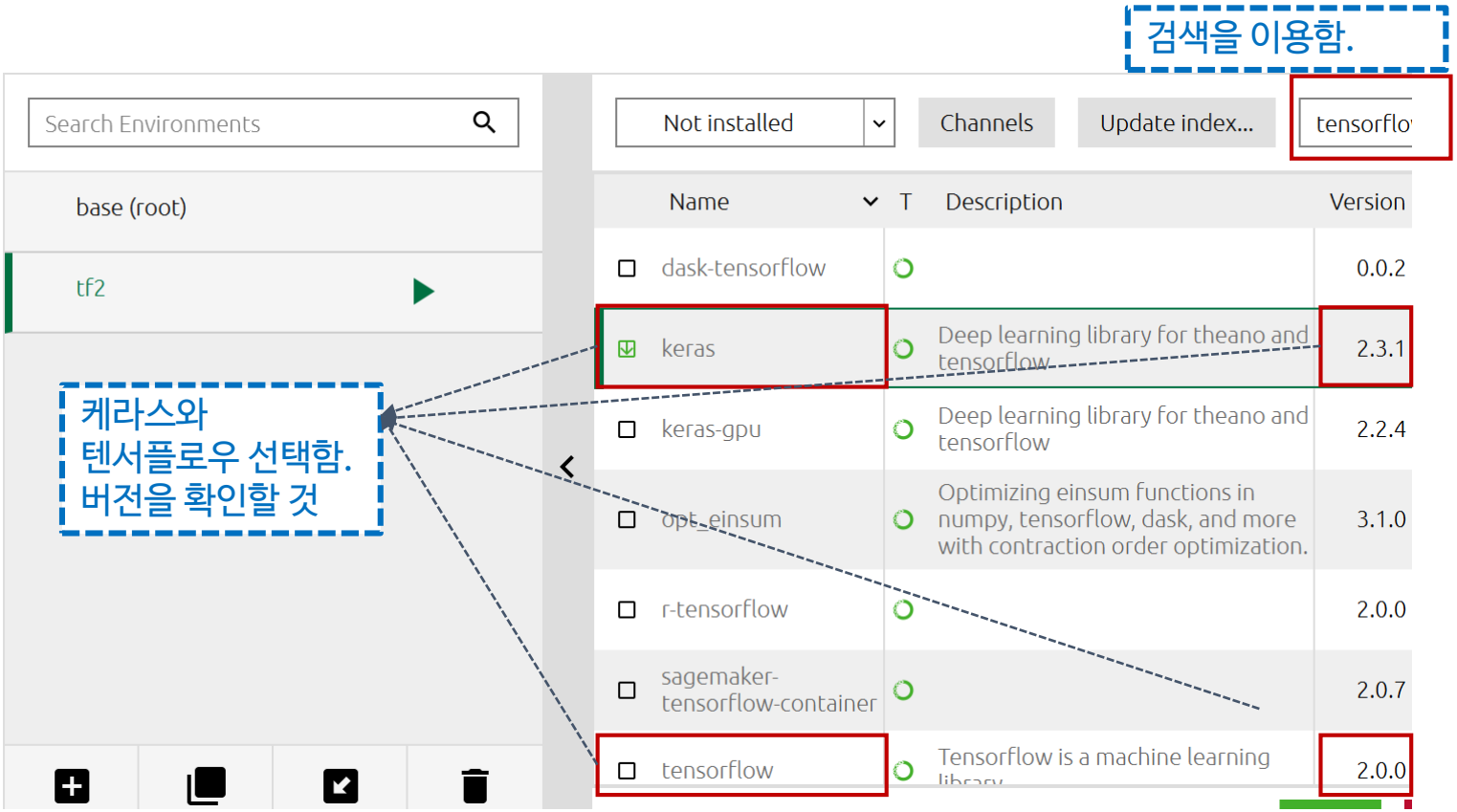
가상환경을 만들자. 이름은 임의로 작성해도 됨



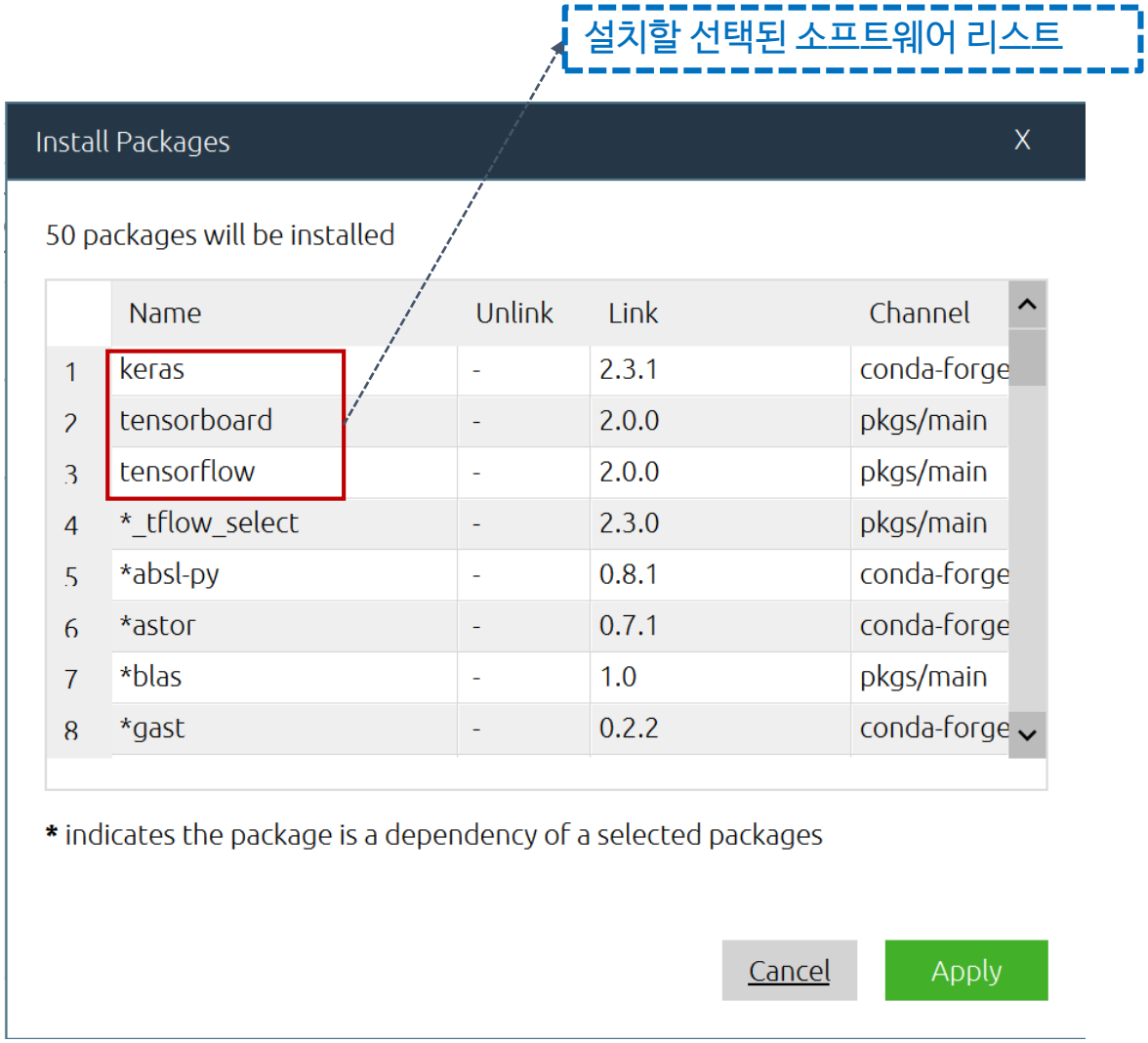
Navigator를 이용한 설치(8)



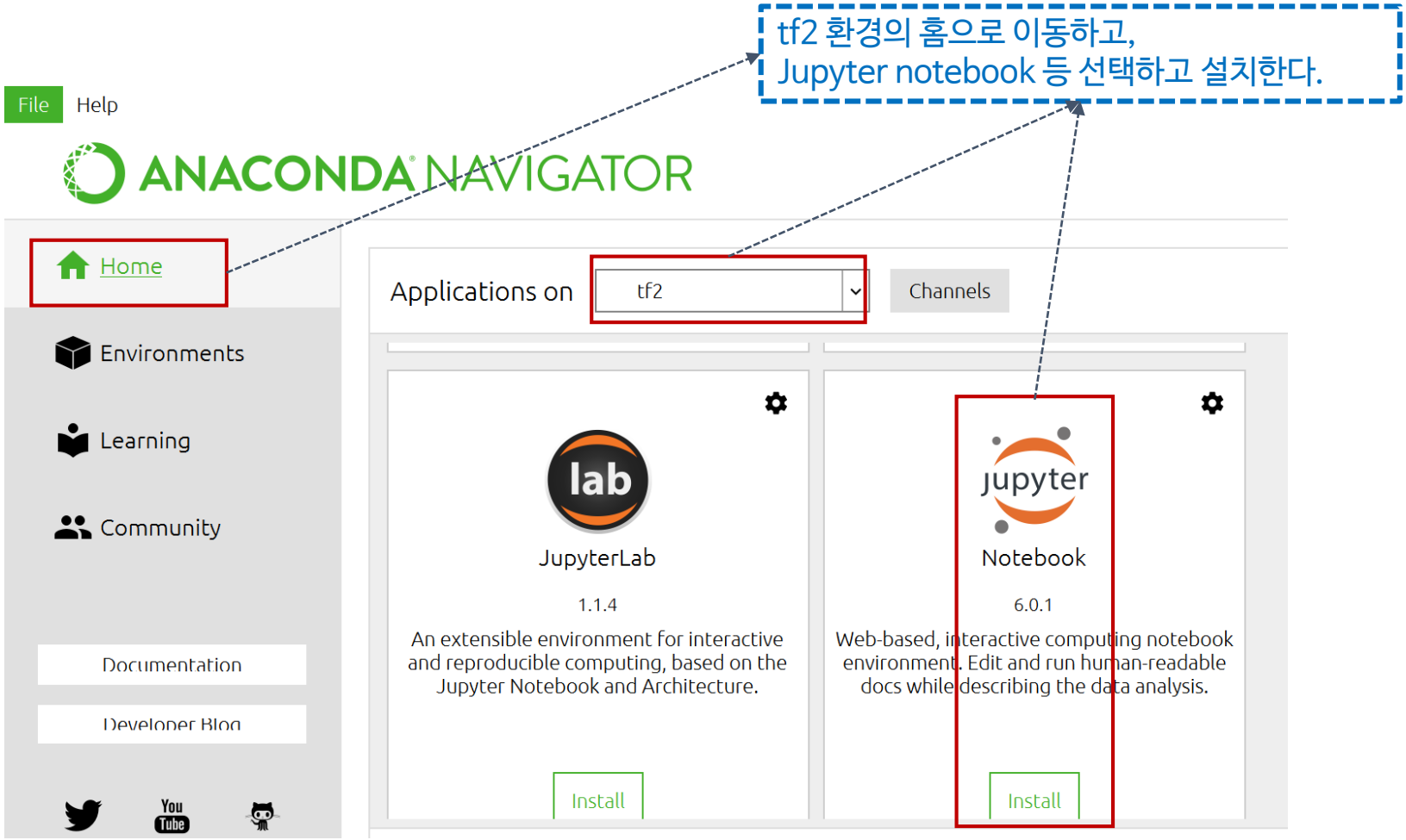
Navigator를 이용한 설치(9)



Navigator를 이용한 설치(10)



Navigator를 이용한 설치(11)



02.
Jupyter notebook 테마 변경

이홍석 (hsyi@kisti.re.kr)

www.ust.ac.kr

jupyter notebook 테마 변경(1)

- jupyterthemes 설치
  - ✓ $pip install --upgrade jupyterthemes
  - ✓ $pip install –upgrade notebook
  - ✓ jt –l(알파벳 L)

```
Available Themes:
   chesterish
   grade3
   gruvboxd
   gruvboxl
   monokai
   oceans16
   onedork
   solarizedd
   solarizedl
```

02.
인공신경망 소개

이홍석 (hsyi@kisti.re.kr)

www.ust.ac.kr

인공신경망 소개

- 인공신경망의 핵심은 딥러닝
- ANN은 다재다능하고 확장성이 좋다
  - ✓ 수 백 개의 이미지를 분류 (구글 이미지)
  - ✓ 음성 인식 서비스 성능을 높임(애플의 시리)
  - ✓ 매일 수억 명의 사용자에게 맞춤형 가장 좋은 비디오를 제공 (유튜브)
  - ✓ 수 백만 개 기보를 학습하고 스스로 학습하면서 성장 (알파고)

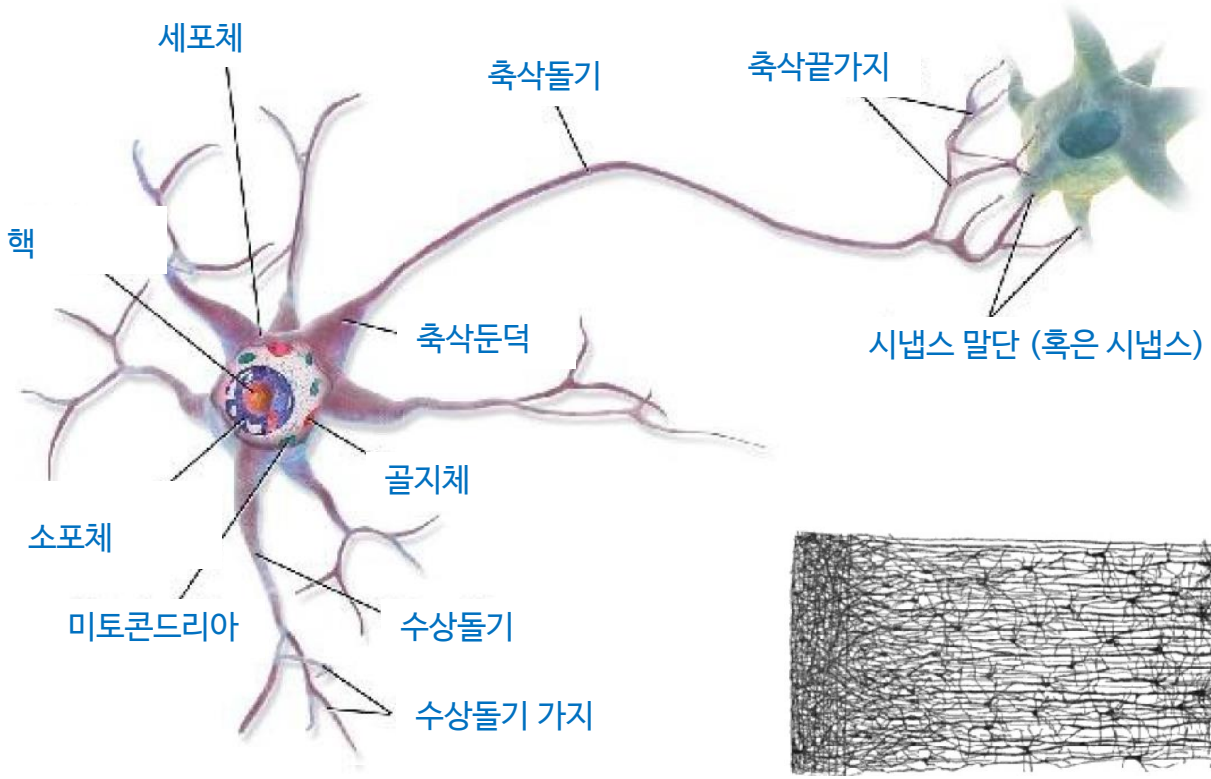
생물학적 뉴런에서 인공 뉴런까지

- **인공 신경망을 처음 소개 (1943년)**
  - ✓ **신경생리학자 워런 멕컬록(McCulloch)와 수학자 월터 피츠(Pitts)가 처음 소개**
    - 명제 논리를 사용해 동물 뇌의 생물학적 뉴런이 복잡한 계산을 위해 어떻게 상호작용하는지에 대한 간단한 계산 모델을 제시 함
- **인공 신경망의 초기 성과 (1960년대까지)**
  - ✓ **지능을 가진 기계와 대화를 기대. 하지만 긴 침체기가 옴**
  - ✓ **1980년대 초에 새로운 네트워크 구조가 발견되고, ANN 관심을 다시 받음**
  - ✓ **하지만 1990년대까지 SVM 처럼 머신러닝 기법을 선호함**
    - 머신러닝이 더 나은 결과를 만들어 주었고, 이론적 기반이 탄탄함

인공 신경망이 부흥은 생활과 밀접

- **인공 신경망이 성공할 수 있는 요인들**
  - ✓ **훈련을 위한 데이터가 엄청나게 많다.**
  - ✓ **대규모 신경망 훈련이 가능할 정도의 계산 능력 기술이 발달함. (GPU 컴퓨팅 등)**
  - ✓ **훈련 알고리즘이 향상되었다.**
    - 1990년대와 비교하여 조금 변경된 것이지만 이 작은 변화는 커다란 영향을 끼침
  - ✓ **일부 인공신경망의 이론상 제한이 실전에서는 문제가 되지 않는다고 밝혀짐**
    - 국소최소점(local minima) 문제는 매우 드물고, 전역 최소점과 가까이 있음
  - ✓ **인공신경망 분야에 투자와 진보의 선순환 시기이다.**
    - 인공신경망 기반 제품들이 나오며, 더 많은 투자가 일어나서 기술이 향상됨

생물학적 뉴런

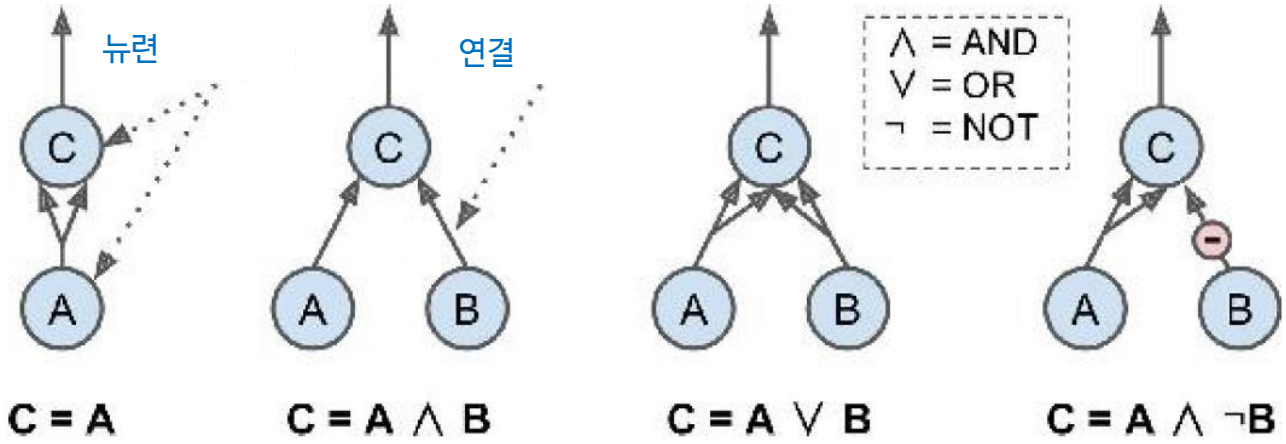


-전기 자극 신호를 받음.
-시냅스는 또 다른 수상돌기와 연결

생물학적 신경망의 여러 층(인간 뇌의 일부)

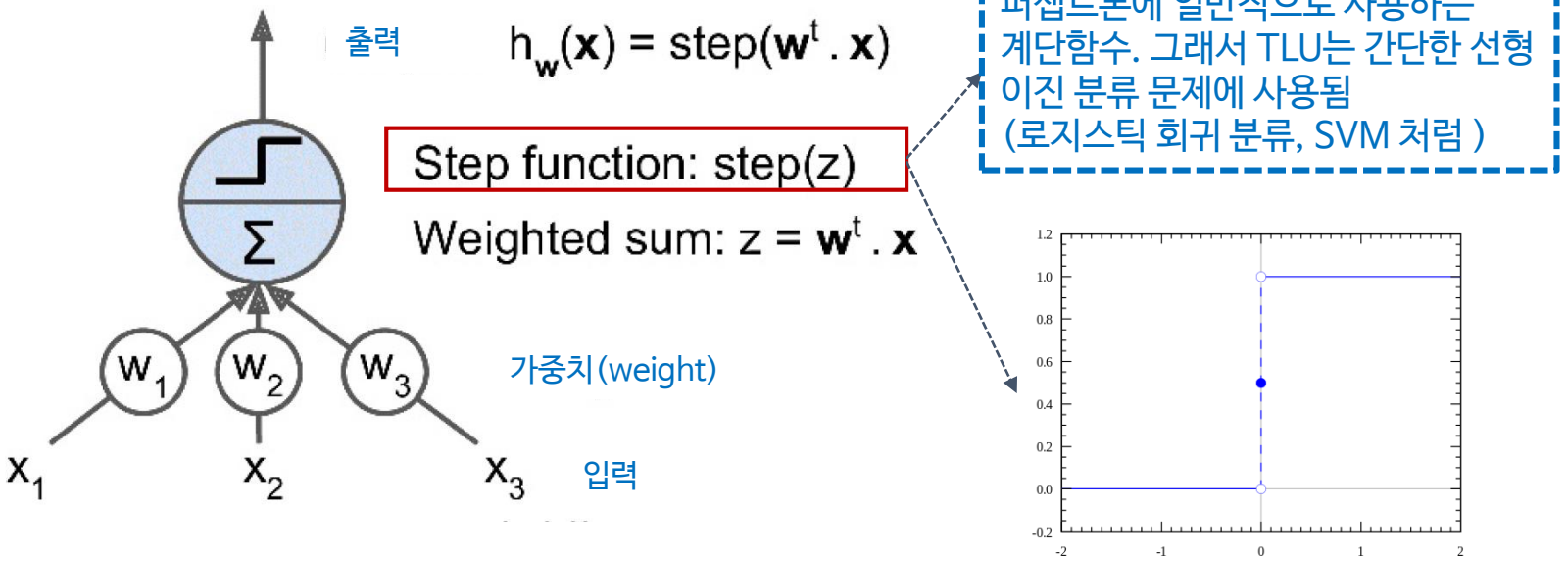
인공 뉴런을 사용한 논리 연산

- 워런 맥컬록과 워터 피츠가 제안한 인공 뉴런
  - ✓ 하나 이상의 이진(on/off) 입력과 하나의 출력으로 모델을 구성함



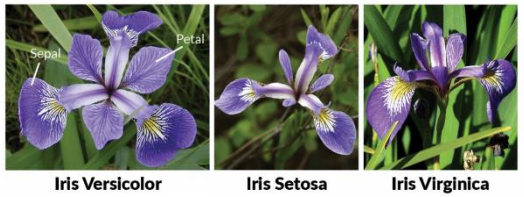
퍼셉트론 Perceptron

- **퍼셉트론(Perceptron)은 가장 간단한 인공 신경망 모델**
  - ✓ **1957년 프랭크 로젠블라트가 제안**
  - ✓ **퍼셉트론은 TLU(Threshold logic unit) 형태의 인공 뉴런을 기반**
    - 입력과 출력은 가중치(weight)와 관련이 있다.



퍼셉트론 응용: 다중 출력 분류기

- **퍼셉트론 인공신경망을 붓꽃(IRIS)에 적용해보자**
  - ✓ 붓꽃의 정보는 2개 (길이와 넓이)로, 3종류의 꽃을 맞추경우 정확도는?



가중치(weight)
퍼셉트론 훈련은 어떻게?
헤브학습(Hebbian Learning)
- 서로 활성화되는 세포가 서로 연결된다는 규칙을 따름
- 즉, 가중치 업데이트

퍼셉트론 응용: 다중 출력 분류기

1.1 퍼셉트론

- 사이킷런 Pereceptron()과 `max_iter` 와 `tol` 의미

붓꽃 데이터

In [3]:
```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron

iris = load_iris()
X = iris.data[:, (2, 3)]  # 꽃잎(petal) 길이와 너비
y = (iris.target == 0).astype(np.int)

per_clf = Perceptron(max_iter=1000, tol=1e-3, random_state=42)
per_clf.fit(X, y)

y_pred = per_clf.predict([[2, 0.5]])
```

perceptron는 sklearn 분류 알고리즘
- max_iter는 이포크 (epochs)
- tol - stop criterion.

In [4]:
```
y_pred
```

Out[4]:
```
array([1])
```

퍼셉트론 응용: 다중 출력 분류기

In [6]:
```
a = -per_clf.coef_[0][0] / per_clf.coef_[0][1]
b = -per_clf.intercept_ / per_clf.coef_[0][1]
```

In [7]:
```
per_clf.coef_
```

Out[7]:
```
array([[-1.4, -2.2]])
```

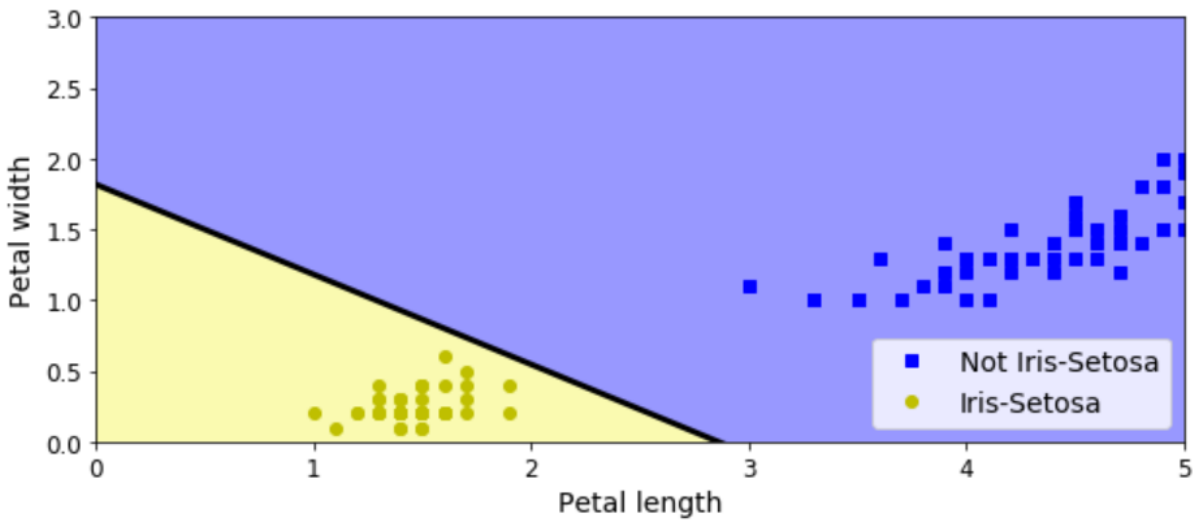
In [14]:
```
per_clf.intercept_
```

Out[14]:
```
array([4.])
```

In [15]:
```
axes = [0, 5, 0, 3]

x0, x1 = np.meshgrid(
        np.linspace(axes[0], axes[1], 500).reshape(-1, 1),
        np.linspace(axes[2], axes[3], 200).reshape(-1, 1),
    )
X_new = np.c_[x0.ravel(), x1.ravel()]
y_predict = per_clf.predict(X_new)
zz = y_predict.reshape(x0.shape)
```

퍼셉트론 응용: 다중 출력 분류기

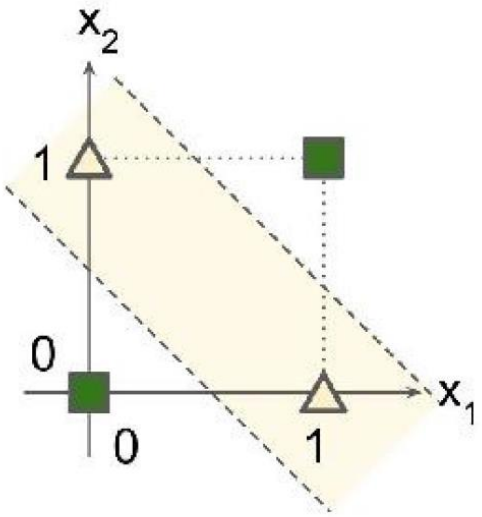
In [16]:
```
plt.figure(figsize=(10, 4))
plt.plot(X[y==0, 0], X[y==0, 1], "bs", label="Not Iris-Setosa")
plt.plot(X[y==1, 0], X[y==1, 1], "yo", label="Iris-Setosa")

plt.plot([axes[0], axes[1]], [a * axes[0] + b, a * axes[1] + b], "k-", linewidth=3)
from matplotlib.colors import ListedColormap
custom_cmap = ListedColormap(['#9898ff', '#fafab0'])

plt.contourf(x0, x1, zz, cmap=custom_cmap)
plt.xlabel("Petal length", fontsize=14)
plt.ylabel("Petal width", fontsize=14)
plt.legend(loc="lower right", fontsize=14)
plt.axis(axes)

plt.show()
```



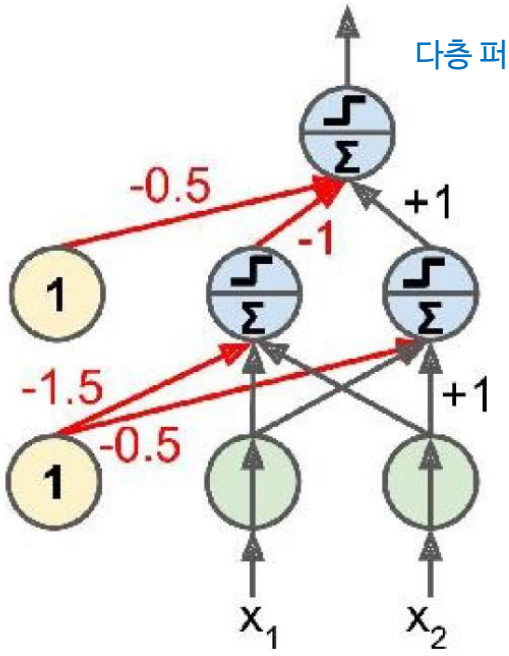
MLP 다층 퍼셉트론의 등장

- XOR 문제는 퍼셉트론으론 풀수 없다. (1969년) 1차 암흑기
  - ✓ XOR(배타적 논리적 합) 분류 문제로 로지스틱 회귀도 이 문제는 풀 수 없다.
  - ✓ 마빈 민스키와 시모어 페퍼트는 퍼셉트론으로 XOR 문제를 풀 수 없다고 함
  - ✓ 하지만, 다층 퍼셉트론으로 XOR 문제를 해결함

XOR 문제

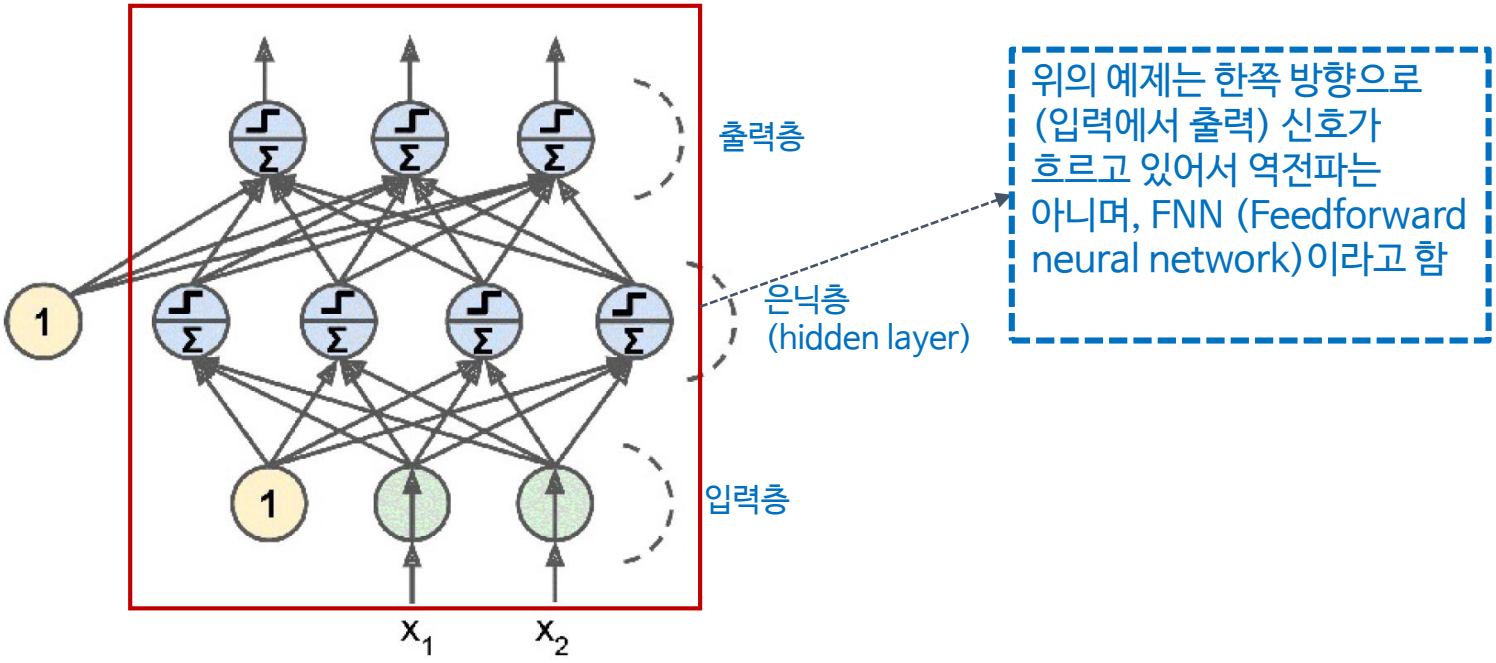


다층 퍼셉트론(2층)



다층 퍼셉트론과 역전파 (1)

- 심층신경망(DNN, Deep Neural Network)
  - ✓ 인공 신경망 은닉층이 2개 이상일 때 신경망임.
- 역전파(Backpropagation) (1986년, Hinton 교수)



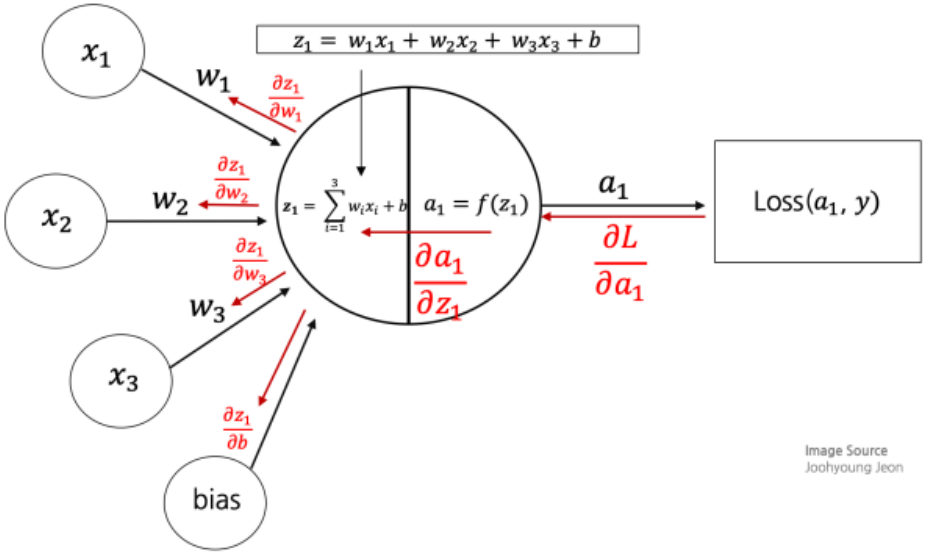
다층 퍼셉트론과 역전파 (2)

- DNN (Deep Neural Network)
  - ✓ 여러 개의 은닉층을 포함한 ANN
  - ✓ 하지만 MLP의 훈련은 매우 어려웠고 잘 안되었음
- 역전파(Backpropagation) 1986년 by Hinton
  - ✓ 역전파 훈련 알고리즘 (1986, Rumelhart, Hinton), 경사 하강법을 사용함
  - ✓ 활성함수를 계단함수에서 시그모이드 혹은 로지스틱 함수로 변경함

$$\sigma(z) = \frac{1}{1+\exp(-z)}$$

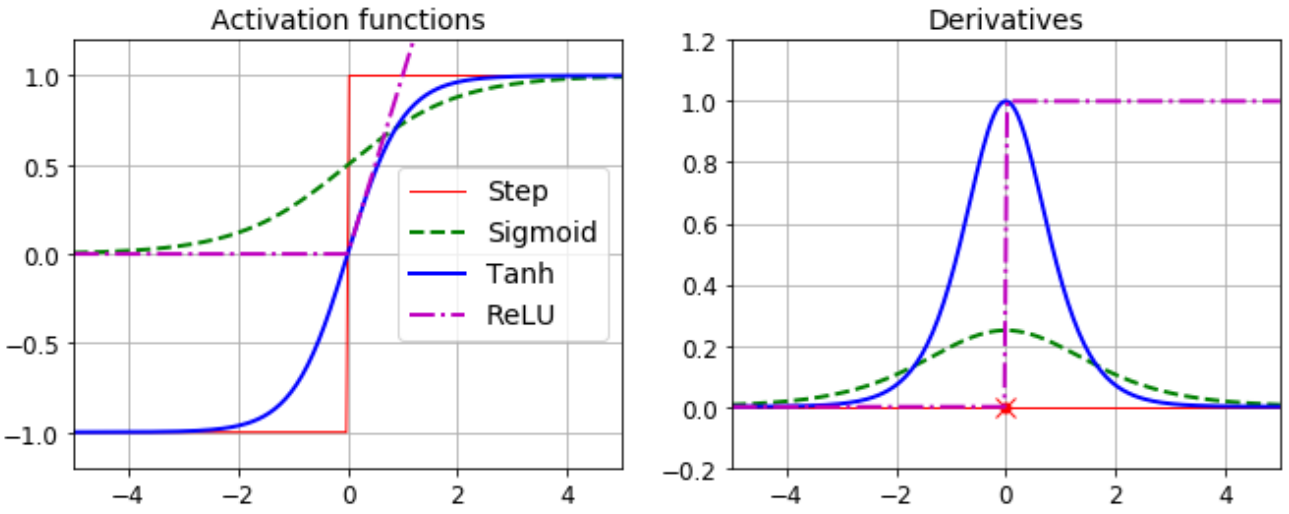
역전파 알고리즘

- 역전파 알고리즘
  - ✓ 각 훈련 샘플에 대해 먼저 예측을 만들고 (정방향 계산)
  - ✓ 오차를 측정 하며
  - ✓ 역방향으로 각 층을 거치면서 각 연결이 오차에 기인한 정도를 측정(역방향 계산)
  - ✓ 마지막으로 이 오차가 감소하도록 가중치를 조금씩 조정함(경사 하강법 스텝)



활성함수

- 역전파 알고리즘에서는 중요한 변화는 활성함수의 도입
  - ✓ 다중 퍼셉트론의 계단 함수에서 미분 가능한 시그모이드 함수
- 최근 활성함수는 여러 종류가 있음
  - ✓ 하이퍼볼릭 탄젠트 함수, ReLU (Rectifiled Linear Unit)



03.
MLP 회귀/분류

이홍석 (hsyi@kisti.re.kr)

www.ust.ac.kr

회귀 MLP (1)

- **다층 퍼셉트론 회귀 예제로 주택 가격 예측을 보자**
  - ✓ **출력은 1개의 값이됨**
  - ✓ **회귀에서는 출력 값에 활성함수를 사용하지 않는다. 따라서 값은 이의의 수가 됨**
  - ✓ **만일 주택 값이 항상 0보다 크면, ReLU 활성함수를 출력 값에 사용할 수 있음**
  - ✓ **비용함수 (loss)는 알반적으로 평균제곱에러(MSE)를 사용하지만,**
  - ✓ **이상치(outliers)가 많을 경우 평균절대값에러(MAE)를 사용을 권장함**
    - 참고로, 후버(Huber) 비용함수는 위의 2개를 혼합한 것이다.
    - 후버 비용은 오차가 기준값(1) 보다 작으면 2차 모양을, 기준값보다 크면 선형이다.

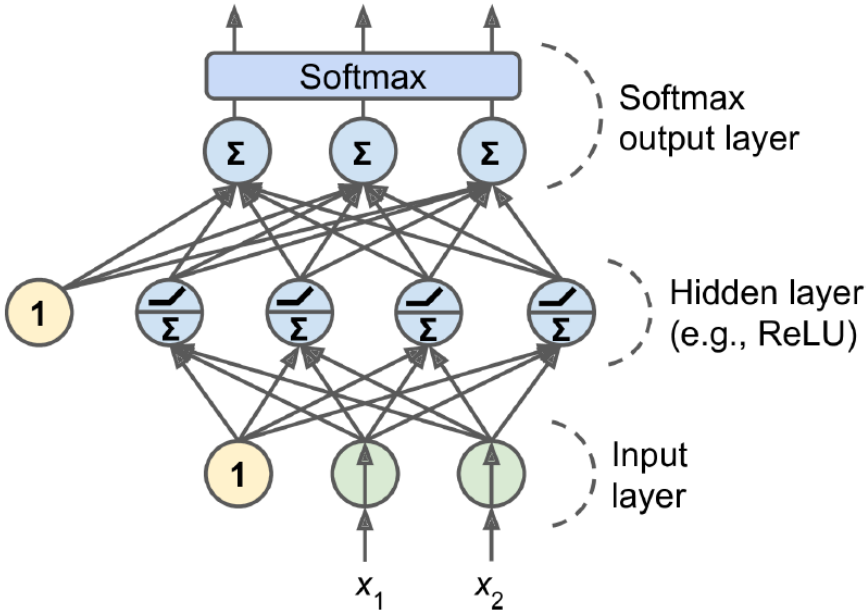
회귀 MLP (2)

Table 10-1. Typical Regression MLP Architecture

Hyperparameter	Typical Value
# input neurons	One per input feature (e.g., 28 x 28 = 784 for MNIST)
# hidden layers	Depends on the problem. Typically 1 to 5.
# neurons per hidden layer	Depends on the problem. Typically 10 to 100.
# output neurons	1 per prediction dimension
Hidden activation	ReLU (or SELU, see Chapter 11)
Output activation	None or ReLU/Softplus (if positive outputs) or Logistic/Tanh (if bounded outputs)
Loss function	MSE or MAE/Huber (if outliers)

분류 MLP (1)

- 이진분류 문제
  - ✓ 1개의 출력값, 로지스틱 활성함수 이용함
- 멀티라벨 이진분류 문제 : 멀티클래스 분류
  - ✓ (예) 메일이 햄인지 스팸인지, 동시에 급한것인지 아닌지를 소프트 맥스로 처리함



분류 MLP (2)

- 분류 MLP에서 비용함수 종류
  - ✓ 비용함수는 확률분포를 예측하므로
  - ✓ 크로스-엔트로비 (cross-entropy) 혹은 로그 로스(log(loss))를 사용함

Table 10-2. Typical Classification MLP Architecture

Hyperparameter	Binary classification	Multilabel binary classification	Multiclass classification
Input and hidden layers	Same as regression	Same as regression	Same as regression
# output neurons	1	1 per label	1 per class
Output layer activation	Logistic	Logistic	Softmax
Loss function	Cross-Entropy	Cross-Entropy	Cross-Entropy

03.
이미지 분류 예제

이홍석 (hsyi@kisti.re.kr)

www.ust.ac.kr

tf.keras.losses 클래스 소개

`class BinaryCrossentropy` : Computes the cross-entropy loss between true labels and predicted labels.

`class CategoricalCrossentropy` : Computes the crossentropy loss between the labels and predictions.

`class CategoricalHinge` : Computes the categorical hinge loss between `y_true` and `y_pred`.

`class CosineSimilarity` : Computes the cosine similarity between `y_true` and `y_pred`.

`class Hinge` : Computes the hinge loss between `y_true` and `y_pred`.

`class Huber` : Computes the Huber loss between `y_true` and `y_pred`.

`class KLDivergence` : Computes Kullback-Leibler divergence loss between `y_true` and `y_pred`.

`class LogCosh` : Computes the logarithm of the hyperbolic cosine of the prediction error.

`class Loss` : Loss base class.

`class MeanAbsoluteError` : Computes the mean of absolute difference between labels and predictions.

`class MeanAbsolutePercentageError` : Computes the mean absolute percentage error between `y_true` and `y_pred`.

`class MeanSquaredError` : Computes the mean of squares of errors between labels and predictions.

`class MeanSquaredLogarithmicError` : Computes the mean squared logarithmic error between `y_true` and `y_pred`.

`class Poisson` : Computes the Poisson loss between `y_true` and `y_pred`.

`class Reduction` : Types of loss reduction.

`class SparseCategoricalCrossentropy` : Computes the crossentropy loss between the labels and predictions.

`class SquaredHinge` : Computes the squared hinge loss between `y_true` and `y_pred`.

keras와 tensorflow 2 적용

- 이번 장의 목표
  - ✓ fashion NMIST dataset
  - ✓ keras.datasets에 있는 fashion NMIST 데이터 세트의 이미지 분류에 대하여 tensorflow2.+이상을 적용해 보자

```
In [21]: import tensorflow as tf
         from tensorflow import keras

In [22]: tf.__version__
Out[22]: '2.0.0'

In [23]: keras.__version__
Out[23]: '2.2.4-tf'
```

```
fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
```

패션 NMINST 데이터(1)

- 패션 NMINST는 총 60,000개 훈련 데이터, 픽셀 크기는 28x28
- 테스트 데이터 개수는 10,000개

```
In [18]: X_train_full.shape
Out[18]: (60000, 28, 28)
```

```
In [21]: X_test.shape
Out[21]: (10000, 28, 28)
```

```
In [19]: X_train_full.dtype
Out[19]: dtype('uint8')
```

```
In [29]: X_valid.shape
Out[29]: (5000, 28, 28)
```

```
In [30]: X_test.shape
Out[30]: (10000, 28, 28)
```

- 검증 데이터 5000개 사용
- 훈련은 55,000개 모두 0~1로 스케일

```
In [20]: X_valid, X_train = X_train_full[:5000] / 255., X_train_full[5000:] / 255.
         y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
         X_test = X_test / 255.
```

패션 NMINST 데이터(2)

```
plt.imshow(X_train[0], cmap="binary")
plt.axis('off')
plt.show()
```

```
plt.imshow(X_train[2], cmap="binary")
plt.axis('off')
plt.show()
```

In [23]:
```
y_train
```

Out[23]:
```
array([4, 0, 7, ..., 3, 0, 5], dtype=uint8)
```

In [24]:
```
class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
               "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]
```

So the first image in the training set is a coat:

In [28]:
```
class_names[y_train[0]]
```

Out[28]:
```
'Coat'
```

패션 NMINST 데이터(3)

```
for row in range(n_rows):
    for col in range(n_cols):
        index = n_cols * row + col
        plt.subplot(n_rows, n_cols, index + 1)
        plt.imshow(X_train[index], cmap="binary", interpolation="nearest")
        plt.axis('off')
        plt.title(class_names[y_train[index]], fontsize=12)
```

케라스 Sequential 모델 (1)

- **텐서플로 2.0의 권장 사항**

  1)작은 함수로 코드를 리팩토링 하자

  - 텐서플로 1.x의 일반적인 사용패턴은 모든 연산을 결합하여 준비한 다음
  - session.run()을 텐서를 평가 함
  - 텐서플로 2.x에서는 필요할 때 호출할 수 있는 작은 함수로 코드를 리팩토링 함

  2) 케라스 층과 모델을 사용해 변수를 관리하자

  - 케라스 모델과 층은 재귀적으로 의존하는 모든 변수를 수집하여 variables와 trainable_variables 속성으로 제공한다. 지역 범위로 변수 관리가 쉽다.

케라스 모델 (2)

- **케라스 사용**

```
import tensorflow as tf

from tensorflow import keras
```

- **Sequential 모델 만들기**

  ✓ **케라스에서는 층(layer)을 조합하여 모델을 만듬. 모델은 층의 그래프이다.**

  - 가장 흔한 모델 구조는 층을 차례대로 쌓는 tf.keras.Sequential 모델이다.

케라스 모델 구성 및 층 설정 (3)

05. 케라스 사용하기

```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))
```

층을 차례로 쌓는 모델을 만들자

input을 일렬화 한다. 1개의 연속 메모리 784개 1차원 배열로 변환. 학습되는 가중치는 없고 데이터를 변환하기만 한다.

keras.layers.Dense 층이 연속적으로 연결되어 완전연결(fully-connected layer) 층이라고 부름.

첫 번째 Dense 층은 300개 노드(뉴런)을 가진다.
두 번째 층은 100개, 마지막 층은 10개의 뉴런이 소프트맥스 활성함수를 통해서 확률적으로 반환함. 전체 확률합은 당연히 1이다.

케라스 모델 (4)

```
keras.backend.clear_session()
np.random.seed(42)
tf.random.set_seed(42)
```

케라스는 고수준의 API를 모듈 방식으로 제공하며, 여러 다른 백엔드 엔진들을 연결함. clear 세션은 현재 케라스 그래프를 없애고, 새로운 그래프를 만들자

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 300)               235500
_________________________________________________________________
dense_1 (Dense)              (None, 100)               30100
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
```

케라스 모델 summary (5)

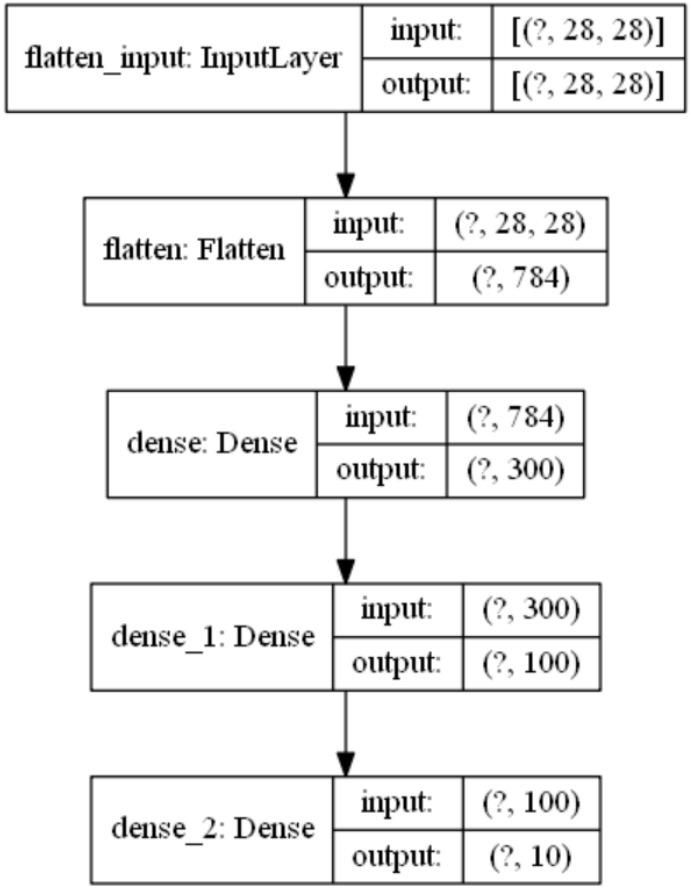
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 300)               235500
_________________________________________________________________
dense_1 (Dense)              (None, 100)               30100
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
```

케라스 모델 구조 (6)

pydot과 graphviz가 설치가 안되어 있다면 ! pip install pydot graphviz

```
keras.utils.plot_model(model, "my_mnist_model.png", show_shapes=True)
```



케라스 Sequential 모델 (7)

```
In [43]: hidden1 = model.layers[1]
         hidden1.name
```

```
Out[43]: 'dense'
```

```
In [44]: model.get_layer(hidden1.name) is hidden1
```

```
Out[44]: True
```

```
In [45]: weights, biases = hidden1.get_weights()
```

```
In [46]: weights
```

```
Out[46]: array([[ 0.02448617, -0.00877795, -0.02189048, ..., -0.02766046,
                  0.03859074, -0.06889391],
                [ 0.00476504, -0.03105379, -0.0586676 , ...,  0.00602964,
                 -0.02763776, -0.04165364],
                [-0.06189284, -0.06901957,  0.07102345, ..., -0.04238207,
                  0.07121518, -0.07331658],
                ...,
                [-0.03048757,  0.02155137, -0.05400612, ..., -0.00113463,
                  0.00228987,  0.05581069],
                [ 0.07061854, -0.06960931,  0.07038955, ..., -0.00384101,
                  0.00034875,  0.02878492],
                [-0.06022581,  0.01577859, -0.02585464, ..., -0.00527829,
                  0.00272203, -0.06793761]], dtype=float32)
```

케라스 Sequential 모델 (8)

```
weights.shape
```

```
(784, 300)
```

```
biases
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

```
biases.shape
```

```
(300,)
```

케라스 Sequential 모델 (9)

```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
```

```
model.compile(loss=keras.losses.sparse_categorical_crossentropy,
              optimizer=keras.optimizers.SGD(),
              metrics=[keras.metrics.sparse_categorical_accuracy])
```

```
history = model.fit(X_train, y_train, epochs=30,
                    validation_data=(X_valid, y_valid))
```

```
Train on 55000 samples, validate on 5000 samples
Epoch 1/30
Epoch 27/30
55000/55000 [==============================] - 3s 59us/sample - loss: 0.2362 - accuracy: 0.9145 - val_loss: 0.3002 - val_accuracy: 0.8890
Epoch 28/30
55000/55000 [==============================] - 3s 59us/sample - loss: 0.2331 - accuracy: 0.9157 - val_loss: 0.3021 - val_accuracy: 0.8926
Epoch 29/30
55000/55000 [==============================] - 3s 59us/sample - loss: 0.2289 - accuracy: 0.9182 - val_loss: 0.2985 - val_accuracy: 0.8924
Epoch 30/30
55000/55000 [==============================] - 3s 59us/sample - loss: 0.2254 - accuracy: 0.9181 - val_loss: 0.3329 - val_accuracy: 0.8806
```

케라스 Sequential 모델 (10)

In [52]:
```
history.params
```

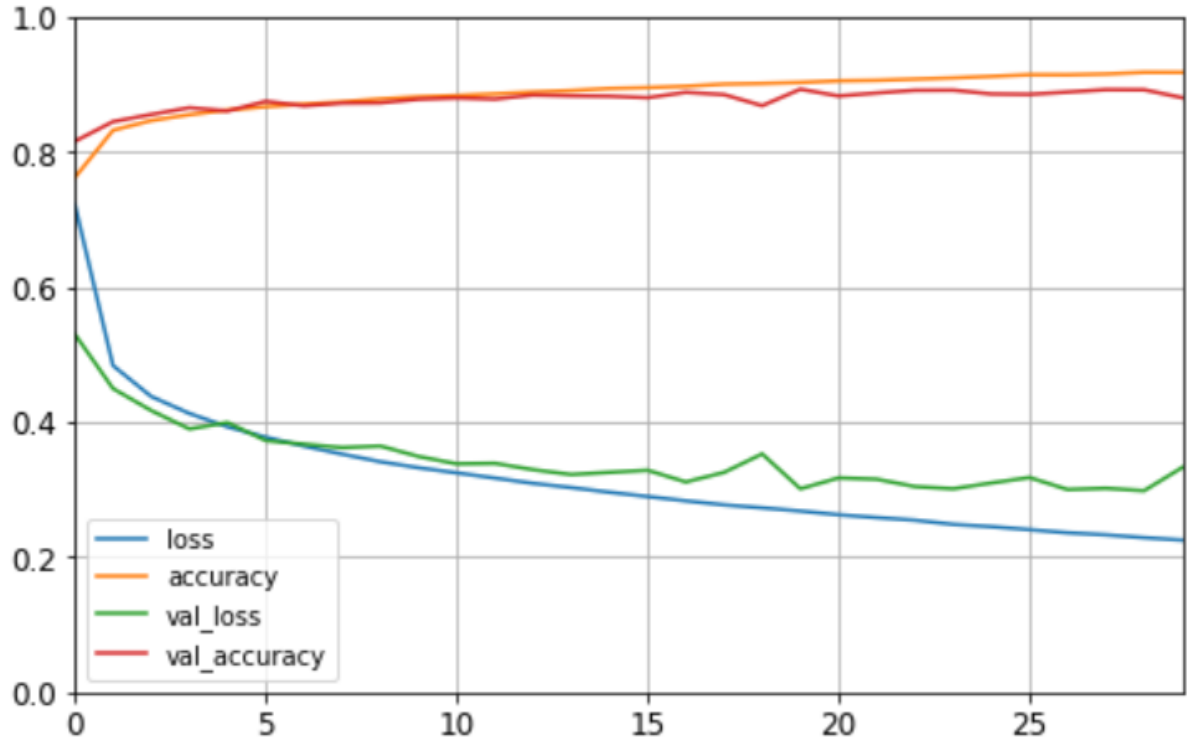
Out[52]:
```
{'batch_size': 32,
 'epochs': 30,
 'steps': 1719,
 'samples': 55000,
 'verbose': 0,
 'do_validation': True,
 'metrics': ['loss', 'accuracy', 'val_loss', 'val_accuracy']}
```

In [53]:
```
print(history.epoch)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,

 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]
```

케라스 Sequential 모델 (11)

```
import pandas as pd
pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
```



케라스 Sequential 모델 (12)

```
In [60]: model.evaluate(X_test, y_test)
```

```
39us/sample - loss: 0.3276 - accuracy: 0.8664

[0.3684497040987015, 0.8664]
```

```
In [61]: X_new = X_test[:3]
         y_proba = model.predict(X_new)
         y_proba.round(2)
```

```
Out[61]: array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.01, 0.  , 0.98],
                [0.  , 0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
                [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ]],
               dtype=float32)
```

케라스 Sequential 모델 (13)

```
In [69]: y_pred = model.predict_classes(X_new);y_pred
Out[69]: array([9, 2, 1], dtype=int64)
```

```
In [63]: np.array(class_names)[y_pred]
Out[63]: array(['Ankle boot', 'Pullover', 'Trouser'], dtype='<U11')
```

```
In [68]: y_new = y_test[:3];y_new
Out[68]: array([9, 2, 1], dtype=uint8)
```

케라스 Sequential 모델 (14)

In [66]:
```
plt.figure(figsize=(7.2, 2.4))
for index, image in enumerate(X_new):
    plt.subplot(1, 3, index + 1)
    plt.imshow(image, cmap="binary", interpolation="nearest")
    plt.axis('off')
    plt.title(class_names[y_test[index]], fontsize=12)
plt.subplots_adjust(wspace=0.2, hspace=0.5)

plt.show()
```

06. 다중 회귀 퍼셉트론 (1)

- California housing dataset

In [1]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
```

In [2]:
```
import pandas as pd
```

In [3]:
```
keras.__version__
```

Out[3]: '2.2.4-tf'

In [4]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

06. 다중 회귀 퍼셉트론 (2)

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

housing = fetch_california_housing()

X_train_full, X_test, y_train_full, y_test = train_test_split(housing.data,
    housing.target, random_state=42)

X_train, X_valid, y_train, y_valid = train_test_split(
    X_train_full, y_train_full, random_state=42)

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_valid = scaler.transform(X_valid)
X_test = scaler.transform(X_test)
```

06. 다중 회귀 퍼셉트론 (2)

```
model = keras.models.Sequential([
    keras.layers.Dense(30, activation="relu",
                       input_shape=X_train.shape[1:]),
    keras.layers.Dense(1)
])
```

```
model.compile(loss="mean_squared_error",
    optimizer=keras.optimizers.SGD(lr=1e-3))
```

```
history = model.fit(X_train, y_train,
    epochs=30, validation_data=(X_valid, y_valid))
```

06. 다중 회귀 퍼셉트론 (2)

```
Train on 11610 samples, validate on 3870 samples
Epoch 1/30
11610/11610 [==============================] - 3s 263us/sample - loss: 1.6321 - val_loss: 1.5903
Epoch 2/30
11610/11610 [==============================] - 1s 77us/sample - loss: 0.7124 - val_loss: 0.6539
Epoch 3/30
11610/11610 [==============================] - 1s 75us/sample - loss: 0.6348 - val_loss: 0.6039
Epoch 4/30
11610/11610 [==============================] - 1s 81us/sample - loss: 0.5983 - val_loss: 0.5738
Epoch 5/30
11610/11610 [==============================] - 1s 83us/sample - loss: 0.5713 - val_loss: 0.5406
Epoch 6/30
11610/11610 [==============================] - 1s 80us/sample - loss: 0.5482 - val_loss: 0.5221
 Epoch 26/30
 11610/11610 [==============================] - 1s 77us/sample - loss: 0.4122 - val_loss: 0.3865
 Epoch 27/30
 11610/11610 [==============================] - 1s 75us/sample - loss: 0.4102 - val_loss: 0.3807
 Epoch 28/30
 11610/11610 [==============================] - 1s 79us/sample - loss: 0.4083 - val_loss: 0.3818
 Epoch 29/30
 11610/11610 [==============================] - 1s 85us/sample - loss: 0.4065 - val_loss: 0.3795
 Epoch 30/30
 11610/11610 [==============================] - 1s 87us/sample - loss: 0.4050 - val_loss: 0.3762
```

06. 다중 회귀 퍼셉트론 (2)

```
mse_test = model.evaluate(X_test, y_test)
```

```
5160/1 [==============================================================================================================================================================================================================================================================================================================================] - 0s 30us/sample - loss: 0.3272
```

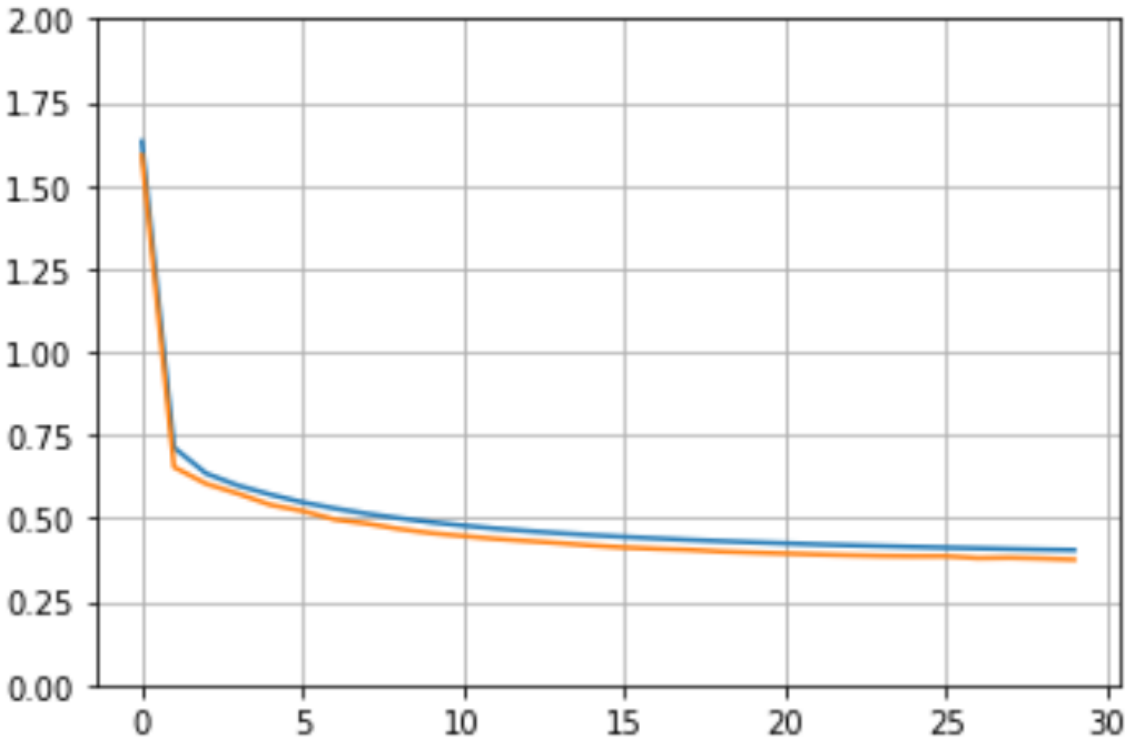
```
X_new = X_test[:3]

y_pred = model.predict(X_new)
```

```
y_pred
```

```
array([[0.39039648],
       [1.7924039 ],
       [3.2830522 ]], dtype=float32)
```

06. 다중 회귀 퍼셉트론 (2)

```
plt.plot(pd.DataFrame(history.history))
plt.grid(True)
plt.gca().set_ylim(0, 2)
plt.show()
```



07. Saving and Restoring

```
model = keras.models.Sequential([
    keras.layers.Dense(30, activation="relu", input_shape=[8]),
    keras.layers.Dense(30, activation="relu"),
    keras.layers.Dense(1)
])
```

```
model = keras.models.Sequential([
    keras.layers.Dense(30, activation="relu", input_shape=[8]),
    keras.layers.Dense(30, activation="relu"),
    keras.layers.Dense(1)
])
```

```
Train on 11610 samples, validate on 3870 samples
Epoch 1/10
11610/11610 [==============================] - 3s 285us/sample - loss: 0.2769 - val_loss: 0.2770
Epoch 2/10
==========================================================================================
==========================================================================================
==================================] - 0s 70us/sample - loss: 0.2228
```

07. Saving and Restoring

```
model.save("my_keras_model.h5")
```

```
model = keras.models.load_model("my_keras_model.h5")
```

```
model.predict(X_new)
```

```
array([[0.54909724],
       [1.6584849 ],
       [3.0271606 ]], dtype=float32)
```

```
model.save_weights("my_keras_weights.ckpt")
```

```
model.load_weights("my_keras_weights.ckpt")
```

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x203728f63c8>
```

08. 콜백

08. 훈련중 콜백 사용하기

```
keras.backend.clear_session()
np.random.seed(42)
tf.random.set_seed(42)
```

```
model = keras.models.Sequential([
    keras.layers.Dense(30, activation="relu", input_shape=[8]),
    keras.layers.Dense(30, activation="relu"),
    keras.layers.Dense(1)
])
```

06. 다중 회귀 퍼셉트론 (2)

```
model.compile(loss="mse", optimizer=keras.optimizers.SGD(lr=1e-3))

checkpoint_cb = keras.callbacks.ModelCheckpoint("my_keras_model.h5",
                save_best_only=True)
history = model.fit(X_train, y_train, epochs=10,
                    validation_data=(X_valid, y_valid),
                    callbacks=[checkpoint_cb])

model = keras.models.load_model("my_keras_model.h5")
# rollback to best model

mse_test = model.evaluate(X_test, y_test)
```

```
-----------------------------------
=] - 0s 68us/sample - loss: 0.3719
```

06. 다중 회귀 퍼셉트론 (2)

```
model.compile(loss="mse", optimizer=keras.optimizers.SGD(lr=1e-3))

early_stopping_cb = keras.callbacks.EarlyStopping(patience=10,
                    restore_best_weights=True)

history = model.fit(X_train, y_train, epochs=100,
                    validation_data=(X_valid, y_valid),
                    callbacks=[checkpoint_cb, early_stopping_cb)

mse_test = model.evaluate(X_test, y_test)
```

```
Train on 11610 samples, validate on 3870 samples
Epoch 1/100
11610/11610 [==============================] - 3s 219us/sample - loss: 0.4388 - val_loss: 0.4122
Epoch 2/100
11610/11610 [==============================] - 1s 85us/sample - loss: 0.4319 - val_loss: 0.4046
```

08. 콜백

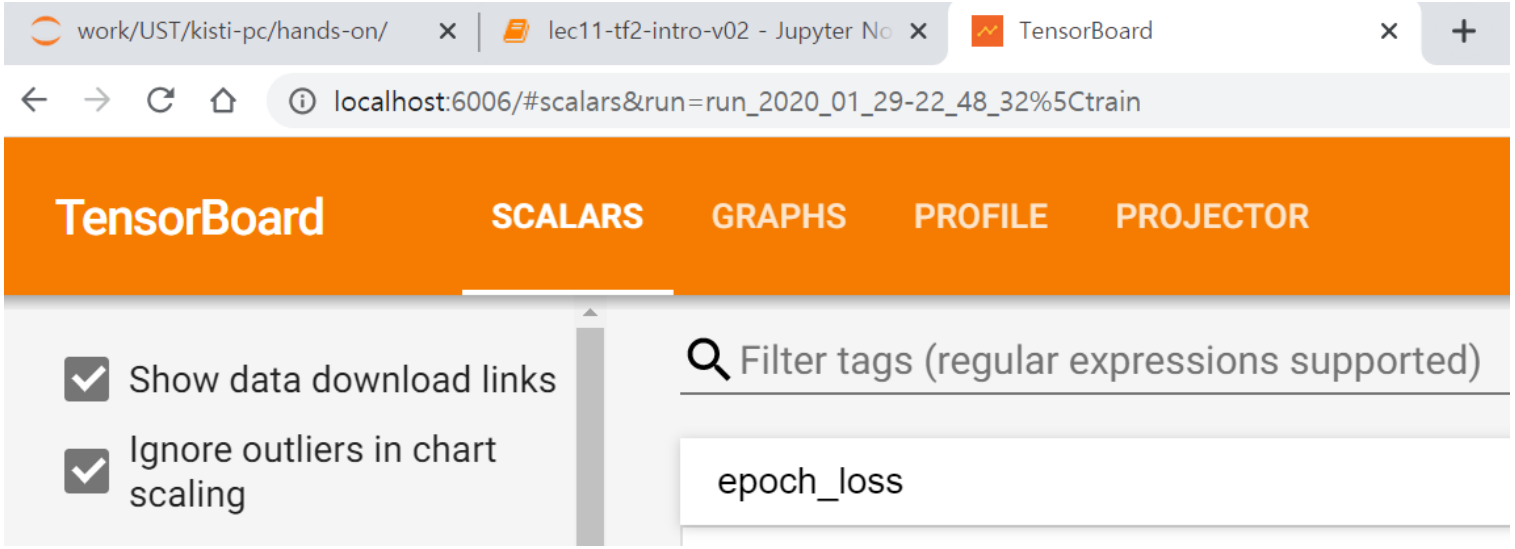
```
class PrintValTrainRatioCallback(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs):
        print("₩nval/train: {:.2f}".format(logs["val_loss"] / logs["loss"]))
```

```
val_train_ratio_cb = PrintValTrainRatioCallback()
history = model.fit(X_train, y_train, epochs=1,
                    validation_data=(X_valid, y_valid),
                    callbacks=[val_train_ratio_cb])
```

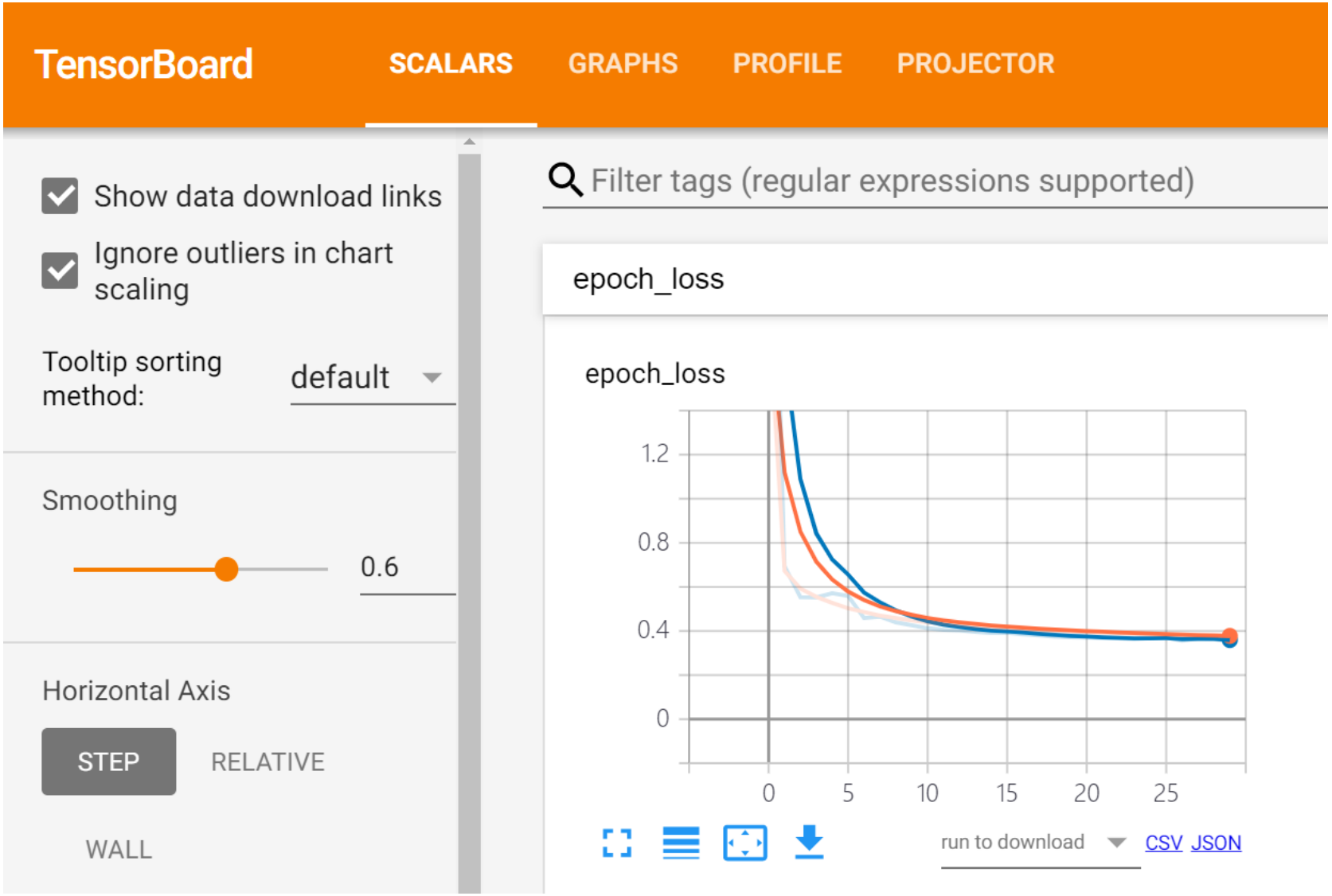
```
Train on 11610 samples, validate on 3870 samples
11168/11610 [============================>..] - ETA: 0s - loss: 0.3450
val/train: 1.10
11610/11610 [==============================] - 1s 95us/sample - loss: 0.3465 - val_loss: 0.3825
```

09. tensorboard()

- 텐서보드 실행
  - ✓ $ tensorboard --logdir=./my_logs --port=6006
- 웹브라우저 (MS 엣지, 크롬) 주소 입력
  - ✓ localhost:6006

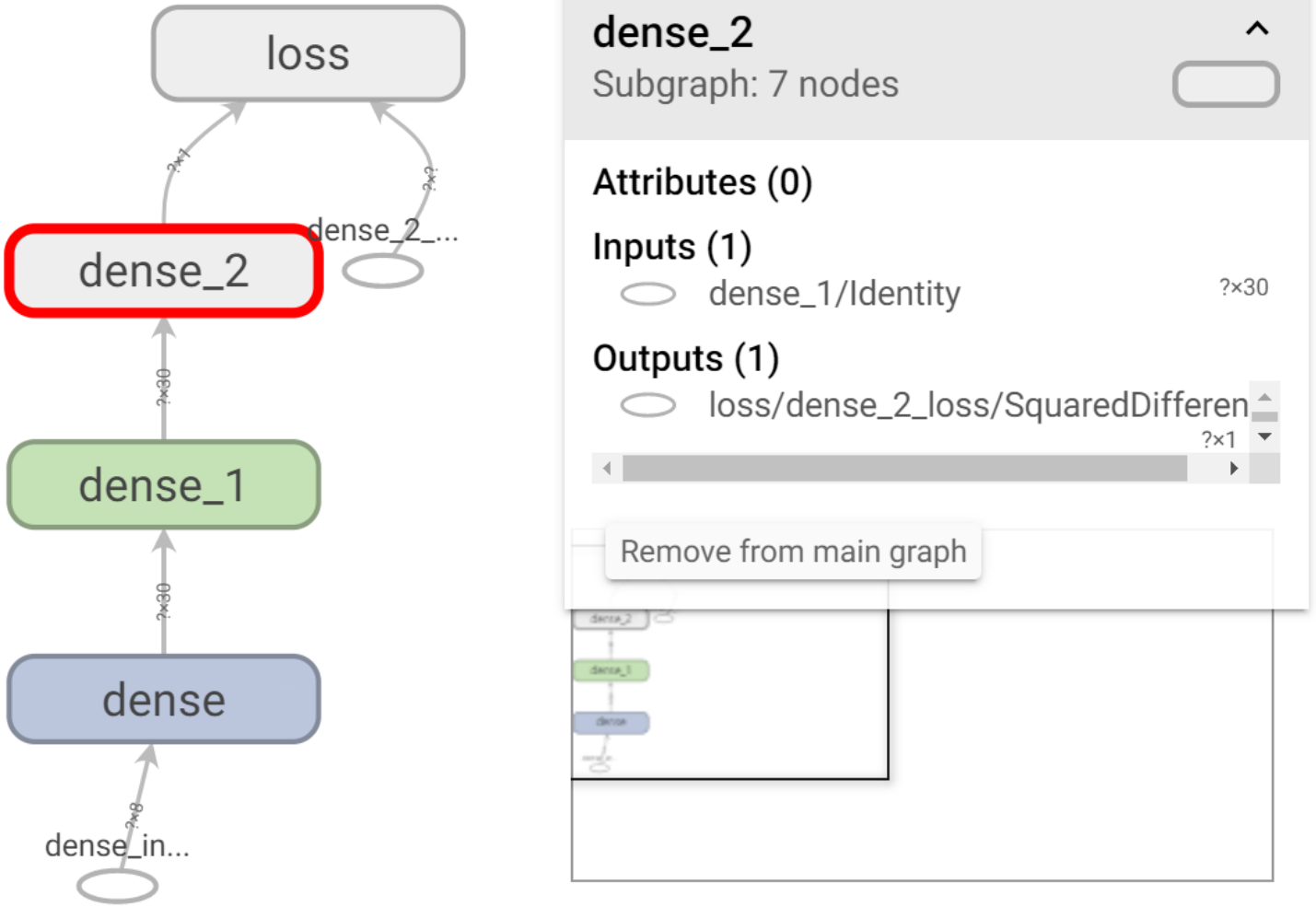


tensorboard()

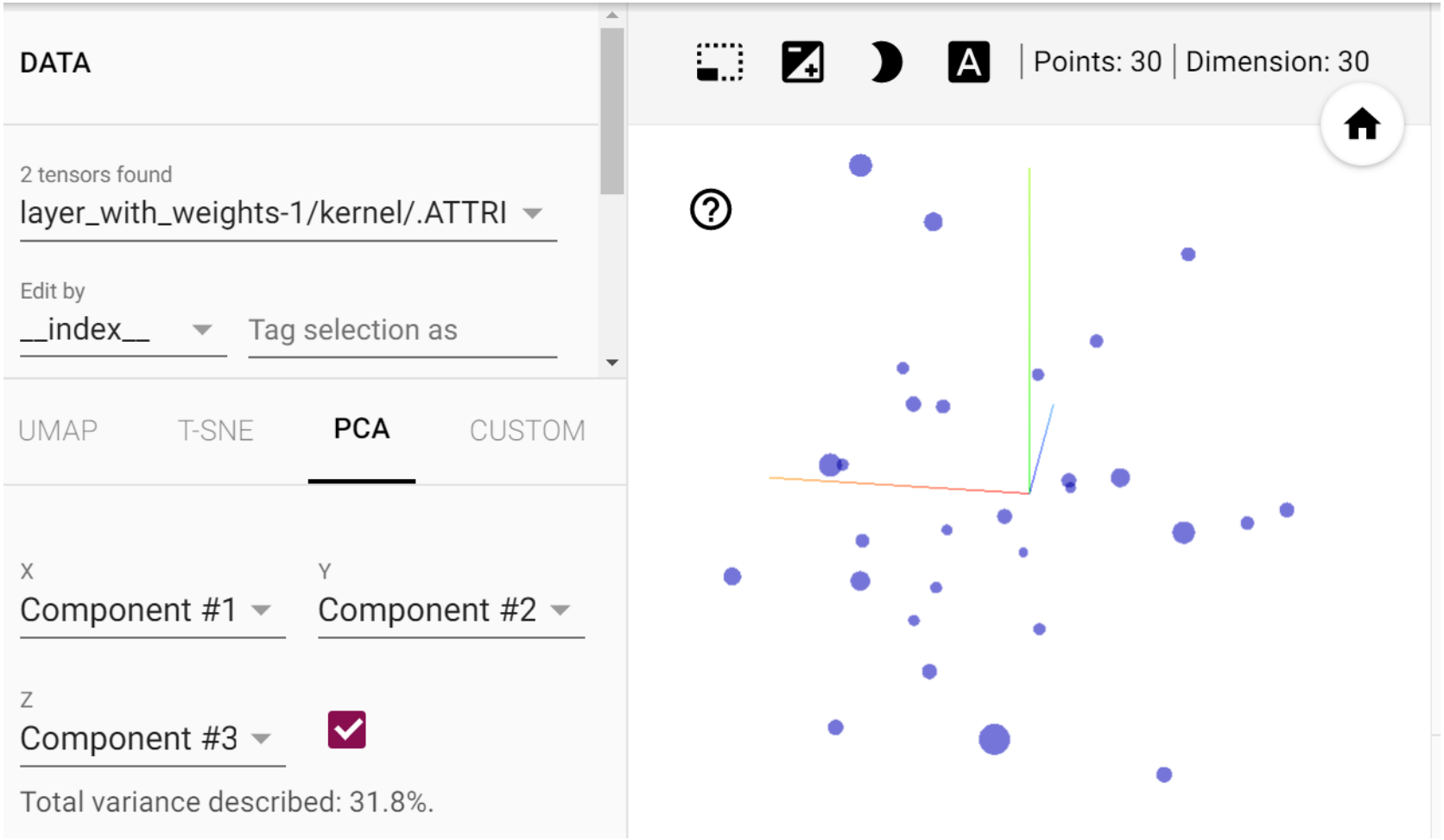


tensorboard()

tensorboard()



tensorboard()



10. 하이퍼 파라미터 튜닝(1)

```
keras.backend.clear_session()
np.random.seed(42)
tf.random.set_seed(42)
```

```
def build_model(n_hidden=1, n_neurons=30,
                learning_rate=3e-3, input_shape=[8]):

    model = keras.models.Sequential()
    model.add(keras.layers.InputLayer(input_shape=input_shape))

    for layer in range(n_hidden):
        model.add(keras.layers.Dense(n_neurons, activation="relu"))

    model.add(keras.layers.Dense(1))
    optimizer = keras.optimizers.SGD(lr=learning_rate)
    model.compile(loss="mse", optimizer=optimizer)
    return model
```

10. 하이퍼 파라미터 튜닝(1)

```
keras_reg = keras.wrappers.scikit_learn.KerasRegressor(build_model)
```

```
keras_reg.fit(X_train, y_train, epochs=100,
              validation_data=(X_valid, y_valid),
              callbacks=[keras.callbacks.EarlyStopping(patience=10)])
```

```
Train on 11610 samples, validate on 3870 samples
Epoch 1/100
11610/11610 [==============================] - 2s 150us/sample - loss: 1.1815 - val_loss: 20.7837
Epoch 2/100
11610/11610 [==============================] - 1s 80us/sample - loss: 0.7013 - val_loss: 0.6646


Epoch 26/100
11610/11610 [==============================] - 1s 94us/sample - loss: 0.3721 - val_loss: 0.3986

<tensorflow.python.keras.callbacks.History at 0x203705a1d48>
```

10. 하이퍼 파라미터 튜닝(1)

```
mse_test = keras_reg.score(X_test, y_test)
```

```
================================================================
====] - 0s 35us/sample - loss: 0.2821
```

```
In [73]: y_pred = keras_reg.predict(X_new)
         y_pred
```

```
Out[73]: array([0.6270608, 1.7558155, 3.7607713], dtype=float32)
```

10. 하이퍼 파라미터 튜닝(1)

```
from scipy.stats import reciprocal
from sklearn.model_selection import RandomizedSearchCV

param_distribs = {
    "n_hidden": [0, 1, 2, 3],
    "n_neurons": np.arange(1, 100),
    "learning_rate": reciprocal(3e-4, 3e-2),
}
```

10. 하이퍼 파라미터 튜닝(1)

```
rnd_search_cv = RandomizedSearchCV(keras_reg,
            param_distribs, n_iter=10, cv=3, verbose=2)
```

```
rnd_search_cv.fit(X_train, y_train, epochs=20,
                  validation_data=(X_valid, y_valid),
                  callbacks=[keras.callbacks.EarlyStopping(patience=10)])
```

Thank You!

www.ust.ac.kr