딥러닝/클라우드

Chapter 11

Intro. Of Keras

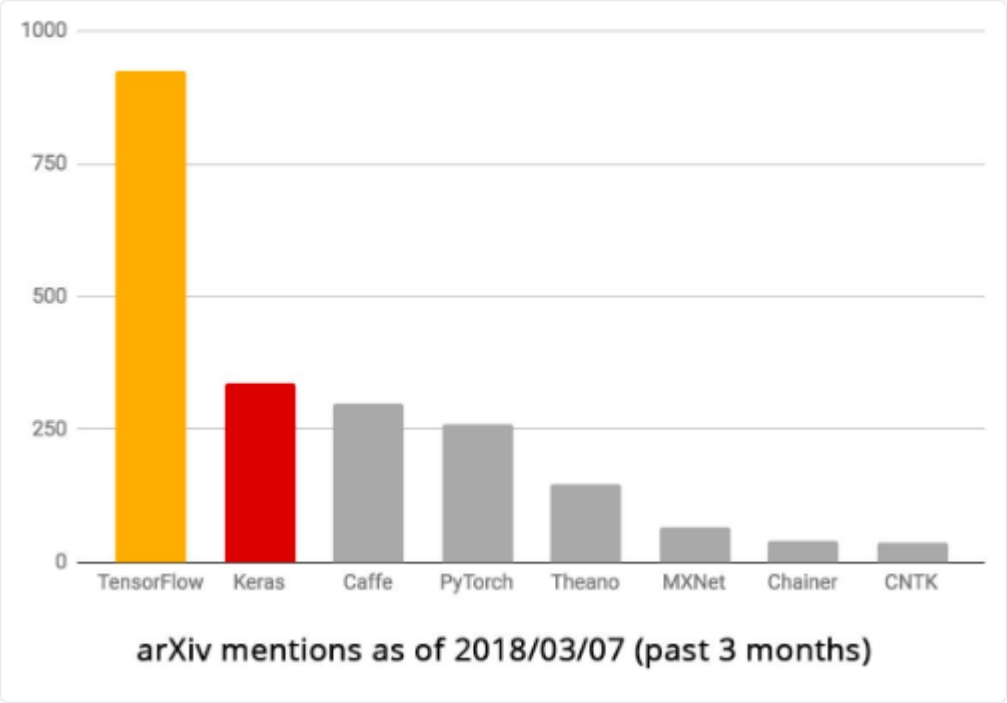
오 세 종

DANKOOK UNIVERSITY

Contents

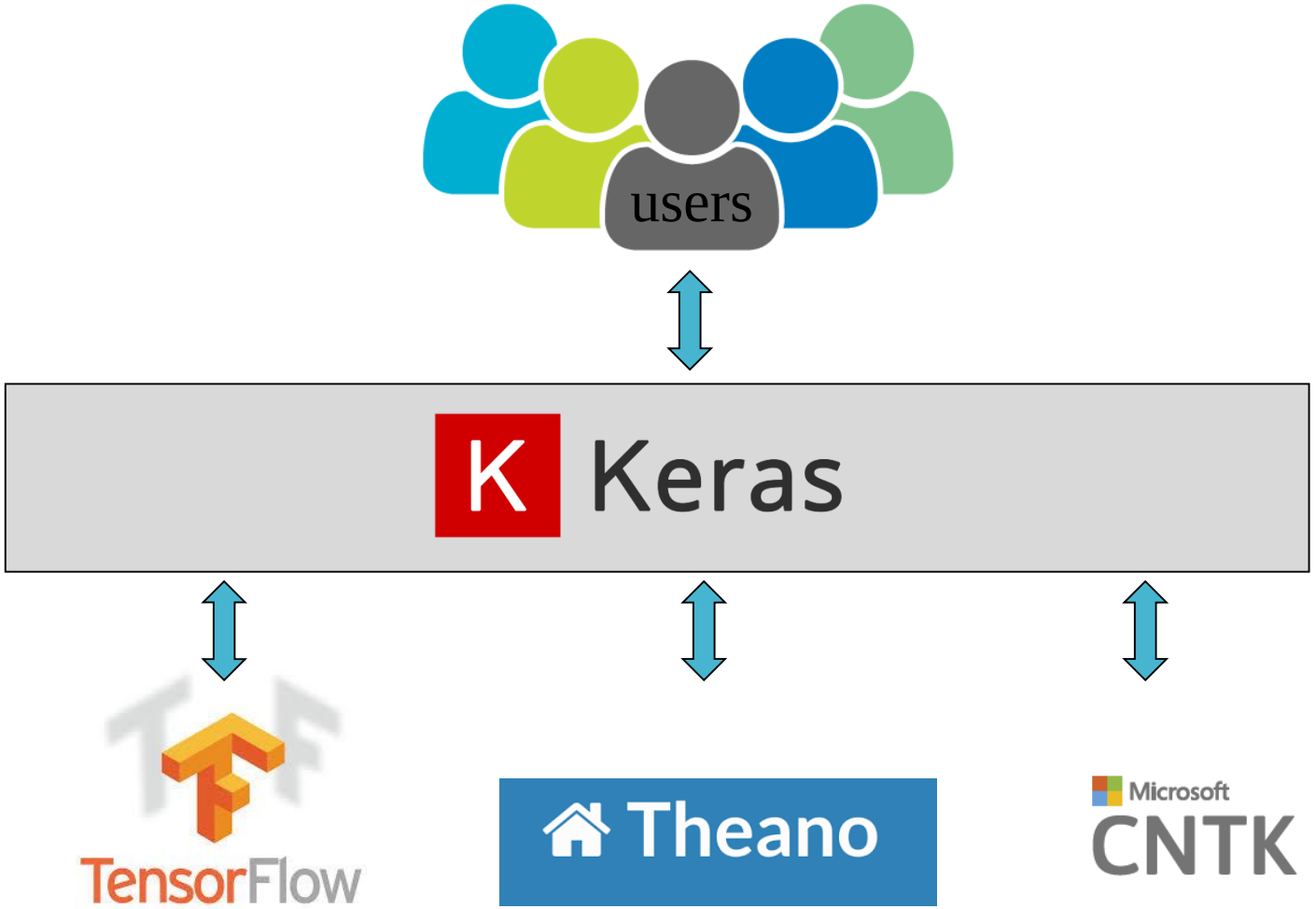
1. Keras 개요

- 파이썬으로 구현된 간결한 딥러닝 라이브러리
- 비전문가도 쉽게 딥러닝 모델을 개발하고 활용 가능
- 직관적 API 제공
- 내부적으로는 Tensorflow, Theano, CNTK 등의 딥러닝 엔진 사용 (사용자에게는 감추어져 있음)
- 구글 엔지니어인 프랑소와 쏠레(Francois Chollet)에 의해 개발, 유지보수됨



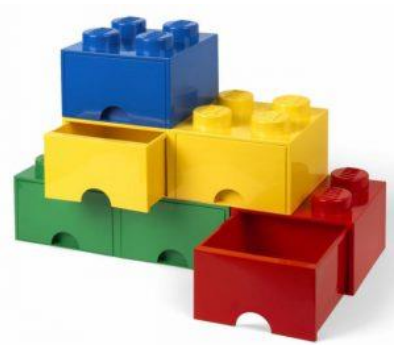
https://twitter.com/fchollet/status/971863128341323776

1. Keras 개요



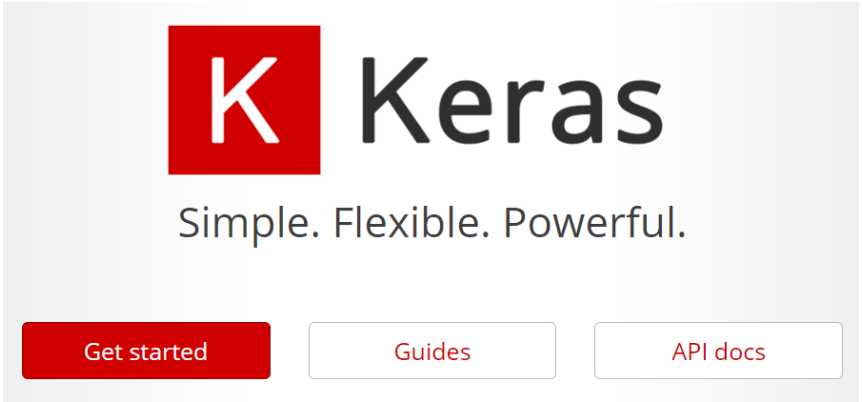
1. Keras 개요

- 주요 특징
  - 모듈화
    - Keras 에서 제공하는 모든 모듈은 독립적, 설정가능, 서로 연결 가능
    - 신경망층, 비용함수, 최적화, 활성함수, 정규화 기법 등이 모두 독립적 모듈로 제공
    - 이들을 조합하여 새로운 모델 구성
  - 최소주의
    - 각 모듈은 짧고 간결, 쉽게 이해 가능
  - 쉬운 확장성
    - 새로운 클래스나 함수로 모듈을 아주 쉽게 추가 가능
  - 파이썬 기반
    - 파이썬 내에서 모델의 구현 가능

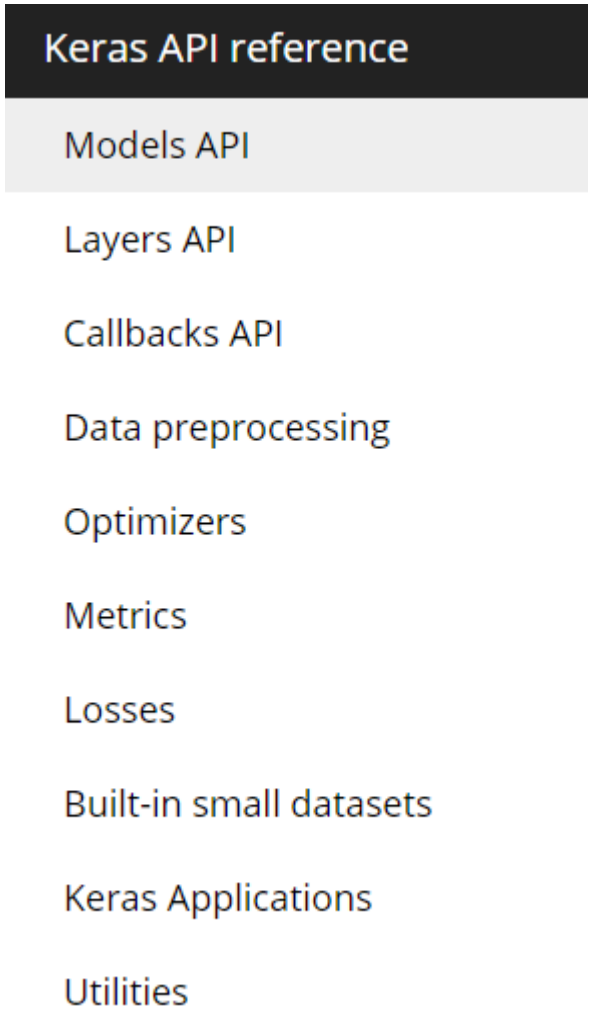


1. Keras 개요

- https://keras.io/

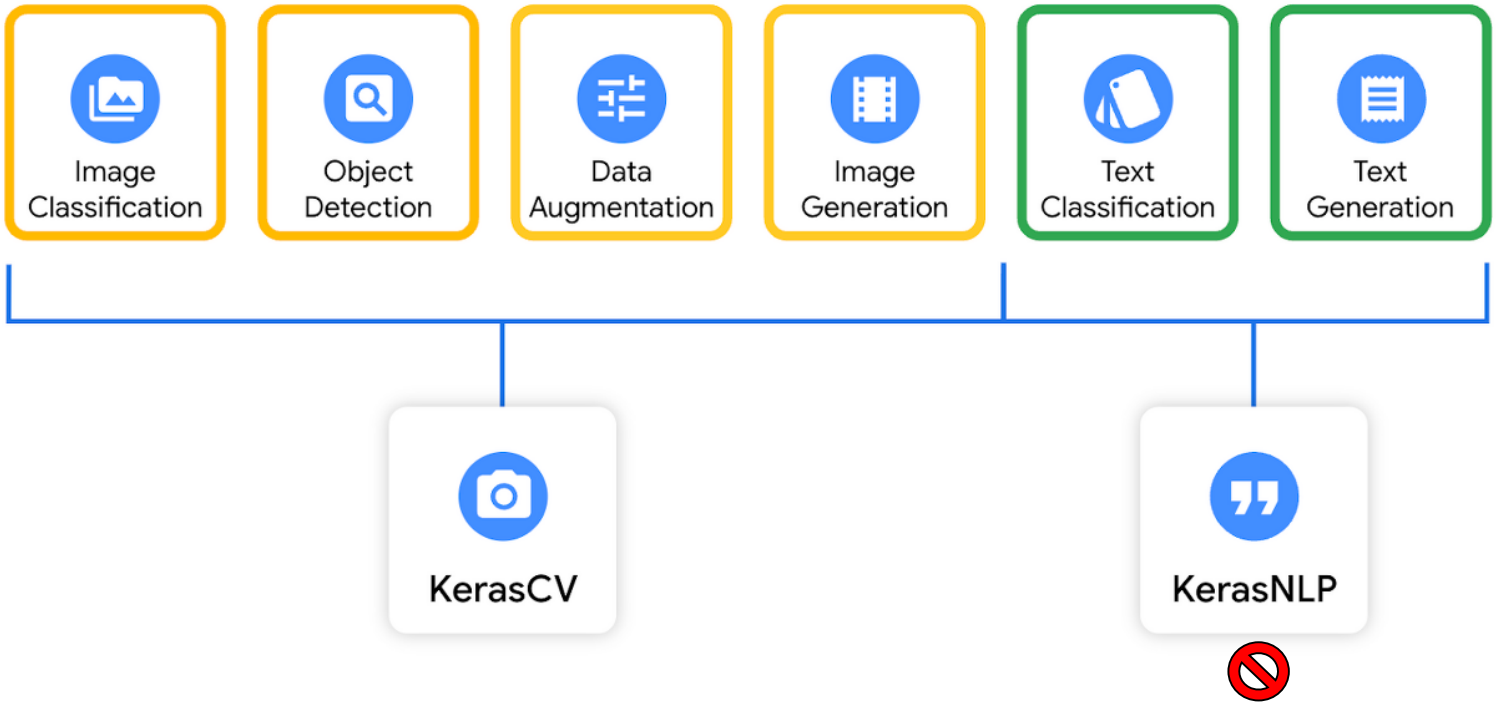


- https://keras.io/ko



1. Keras 개요

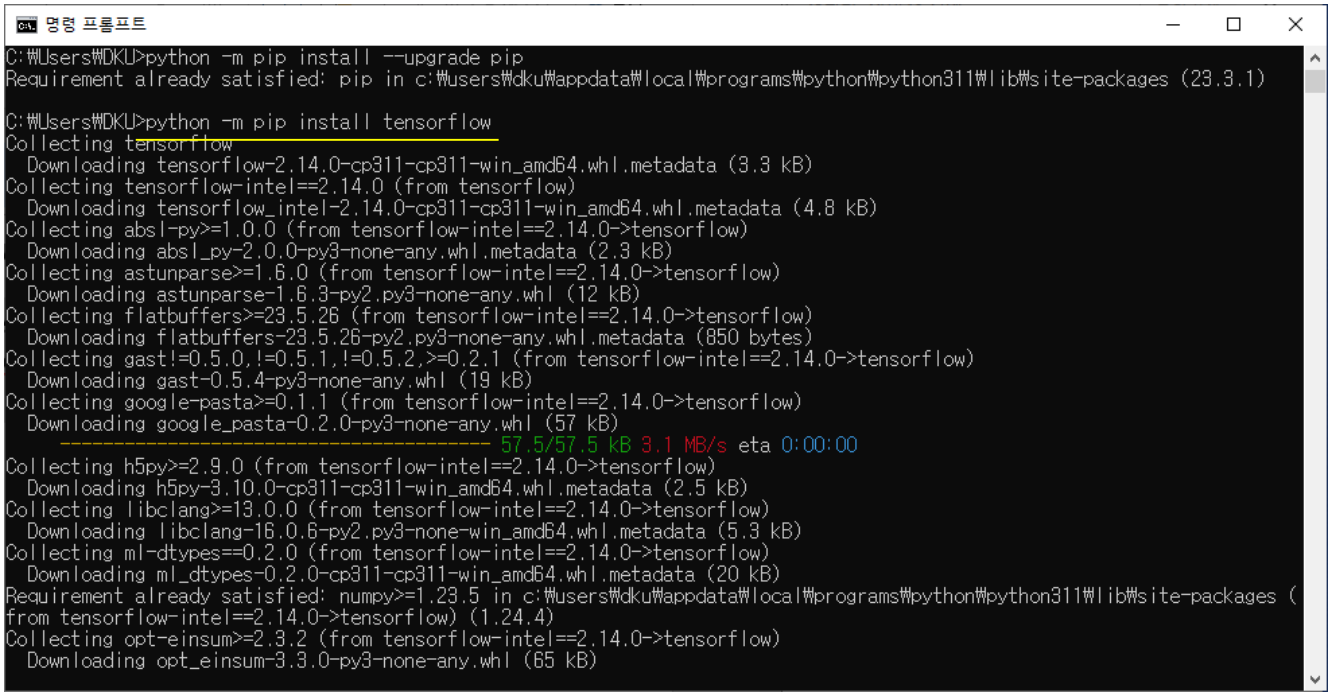
- KerasCV와 KerasNLP
  - 다양한 테스크의 최신 모델을 코드 몇 줄로 사용 가능



2. Keras 설치

- (2) window prompt 에서 다음 명령어 실행

```
> pip install --upgrade pip
> Pip install keras
> pip install tensorflow
```



GPU 가 있는 컴퓨터는 Tensorflow GPU 버전을 설치 하는 것이 유리

- Note
  - Python, keras, tensorflow 의 버전이 잘 맞아야 설치가 정상적으로 이루어짐
  - Python 3.11.9 에서는 정상 설치됨을 확인

```
C:₩Users₩DKU>python --version
Python 3.11.9

C:₩Users₩DKU>pip show keras
Name: keras
Version: 3.6.0
Summary: Multi-backend Keras.
Home-page: https://github.com/keras-team/keras
Author: Keras team
Author-email: keras-users@googlegroups.com
License: Apache License 2.0
Location: d:₩python₩python311₩Lib₩site-packages
Requires: absl-py, h5py, ml-dtypes, namex, numpy, optree, packaging, rich
Required-by: tensorflow-intel

C:₩Users₩DKU>pip show tensorflow
Name: tensorflow
Version: 2.17.0
Summary: TensorFlow is an open source machine learning framework for everyone.
```

2. Keras 설치

- (3) keras 테스트

11.dnn_basic_iris.py

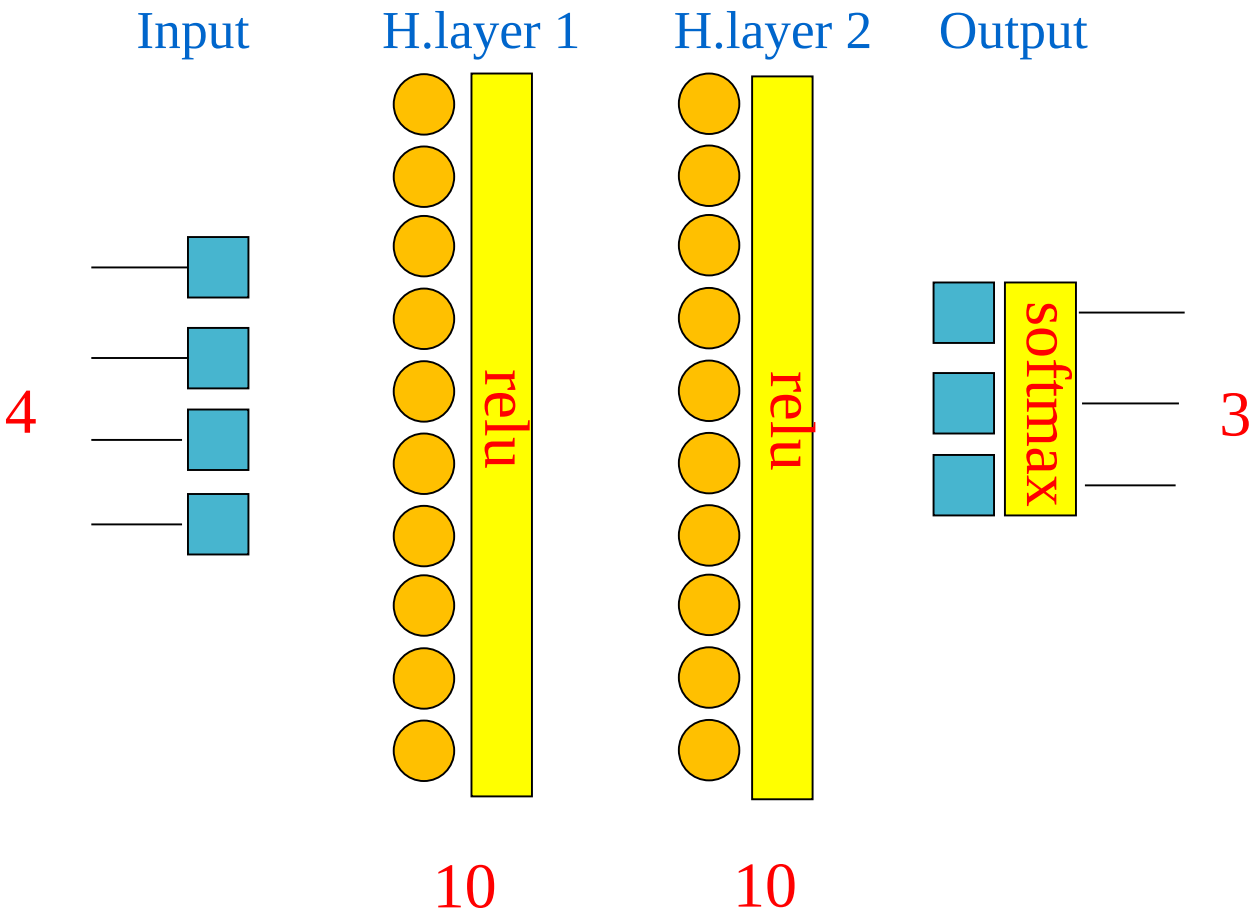
```
import tensorflow as tf
import keras

print('tensorflow ' + tf.__version__)
print('keras ' + keras.__version__)
```

```
>>> print('tensorflow ' + tf.__version__)
tensorflow 2.17.0
>>> print('keras ' + keras.__version__)
keras 3.6.0
>>>
```

3. 다층신경망(DNN) 예제

- iris dataset 이용 품종 예측
  - Training 60%, test(validation) 40%
  - Two layer neural network



3. 다층신경망(DNN) 예제

- iris dataset

iris.csv

	A	B	C	D	E
1	Sepal.Leng	Sepal.Widt	Petal.Leng	Petal.Widtl	Species
2	5.1	3.5	1.4	0.2	setosa
3	4.9	3	1.4	0.2	setosa
4	4.7	3.2	1.3	0.2	setosa
5	4.6	3.1	1.5	0.2	setosa
6	5	3.6	1.4	0.2	setosa
7	5.4	3.9	1.7	0.4	setosa
8	4.6	3.4	1.4	0.3	setosa
9	5	3.4	1.5	0.2	setosa
10	4.4	2.9	1.4	0.2	setosa
11	4.9	3.1	1.5	0.1	setosa
12	5.4	3.7	1.5	0.2	setosa
13	4.8	3.4	1.6	0.2	setosa
14	4.8	3	1.4	0.1	setosa
15	4.3	3	1.1	0.1	setosa
16	5.8	4	1.2	0.2	setosa
17	5.7	4.4	1.5	0.4	setosa

3. 다층신경망(DNN) 예제

```
# load required modules
from keras.models import Sequential
from keras.layers import Dense, Input
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

3. 다층신경망(DNN) 예제

```
# load dataset
dataframe = pd.read_csv("D:/Rworks/iris.csv")
dataset = dataframe.values
X = dataset[:,0:4].astype(float)
Y = dataset[:,4]

# encode class values as integers
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)

# one hot encoding
dummy_y = to_categorical(encoded_Y)
# Divide train, test
train_X, test_X, train_y, test_y = train_test_split(X,
dummy_y, test_size=0.4, random_state=321)
```

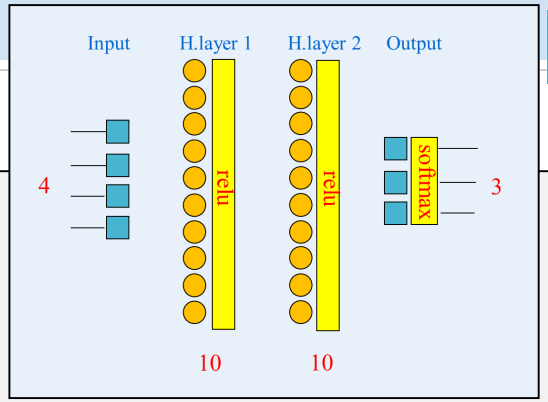
3. 다층신경망(DNN) 예제

```
In [91]: Y
Out[91]:
array(['setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
```

```
In [92]: encoded_Y
Out[92]:
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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
>>> dummy_y
array([[1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
```

3. 다층신경망(DNN) 예제




```
# define model (DNN structure)
epochs = 50
batch_size = 10

model = Sequential()
model.add(Input(shape=(4,)))        # input layer (4 features)
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(3, activation='softmax'))

model.summary()  # show model structure

# Compile model
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

optimizer 학습속도를 빠르고 안정적이게 만드는 역할 (cf. 네비게이션)

https://gomguard.tistory.com/187

3. 다층신경망(DNN) 예제

weight values (4*10) + nodes(10)

```
>>> model.summary()  # show model structure
Model: "sequential"
```

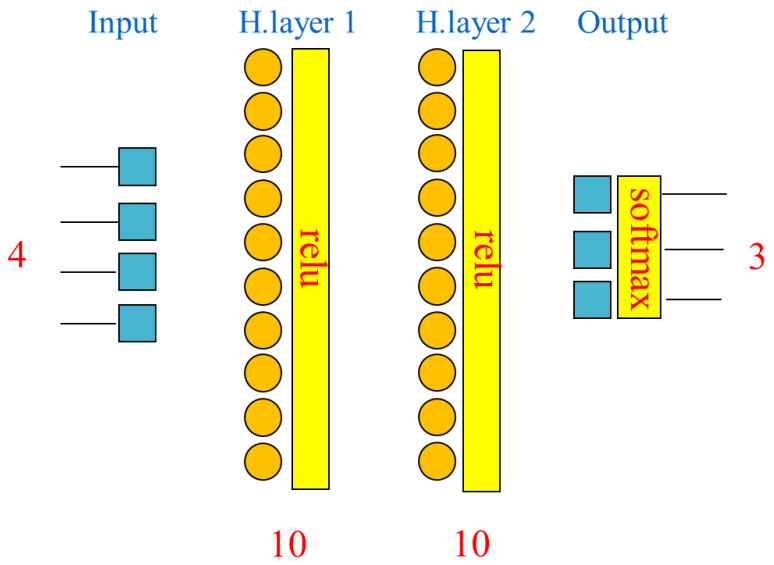
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 10)	50
dense_1 (Dense)	(None, 10)	110
dense_2 (Dense)	(None, 3)	33

H.Layer 1

H.Layer 2

Output Layer

```
Total params: 193 (772.00 B)
Trainable params: 193 (772.00 B)
Non-trainable params: 0 (0.00 B)
```



3. 다층신경망(DNN) 예제

```
# model fitting (learning)
disp = model.fit(train_X, train_y,
                 batch_size=batch_size,
                 epochs=epochs,
                 verbose=1,          # print fitting process
                 validation_data=(test_X, test_y))
```

```
Epoch 1/50
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9657 - loss: 0.1959 - val_accuracy: 0.9667 - val_loss: 0.
2089
Epoch 2/50
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9772 - loss: 0.2257 - val_accuracy: 0.9667 - val_loss: 0.1
994
Epoch 3/50
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9463 - loss: 0.2225 - val_accuracy: 0.9667 - val_loss: 0.1
957
```

training

validation

3. 다층신경망(DNN) 예제

```
# Test model
pred = model.predict(test_X)
print(pred)
y_classes = [np.argmax(y, axis=None, out=None) for y in pred]
print(y_classes)   # result of prediction

# model performance
score = model.evaluate(test_X, test_y, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

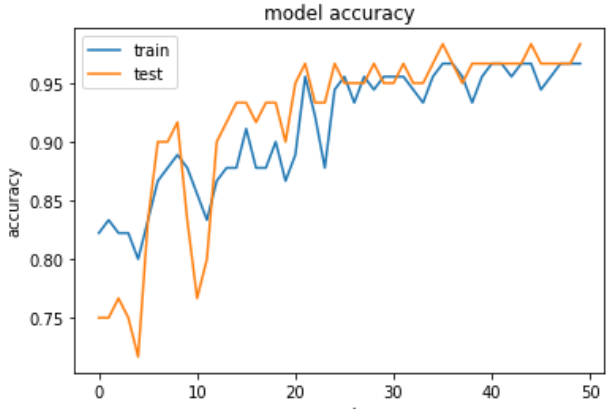
```
>>> print(pred)
[[9.96481419e-01 3.51755484e-03 9.66665880e-07]
 [2.25141406e-07 4.30988930e-02 9.56900835e-01]
 [1.20285108e-06 6.39933348e-02 9.36005414e-01]
 [3.62874917e-03 9.69242632e-01 2.71285754e-02]
 [9.98669505e-01 1.33024890e-03 1.95104420e-07]
 [2.68770289e-03 9.69599783e-01 2.77125258e-02]
 [4.07093609e-07 2.72629037e-02 9.72736597e-01]
```

```
>>> print(y_classes)   # result of prediction
[0, 2, 2, 1, 0, 1, 2, 0, 1, 0, 0, 0, 1, 1, 2, 1, 2, 0, 1, 0, 2, 1, 1, 0, 1
, 1, 1, 2, 0, 1, 2, 1, 0, 1, 2, 0, 2, 2, 0, 0, 1, 1, 2, 0, 0, 1, 2, 1, 0,
1, 1, 1, 2, 0, 2, 2, 1, 2, 2, 0]
```

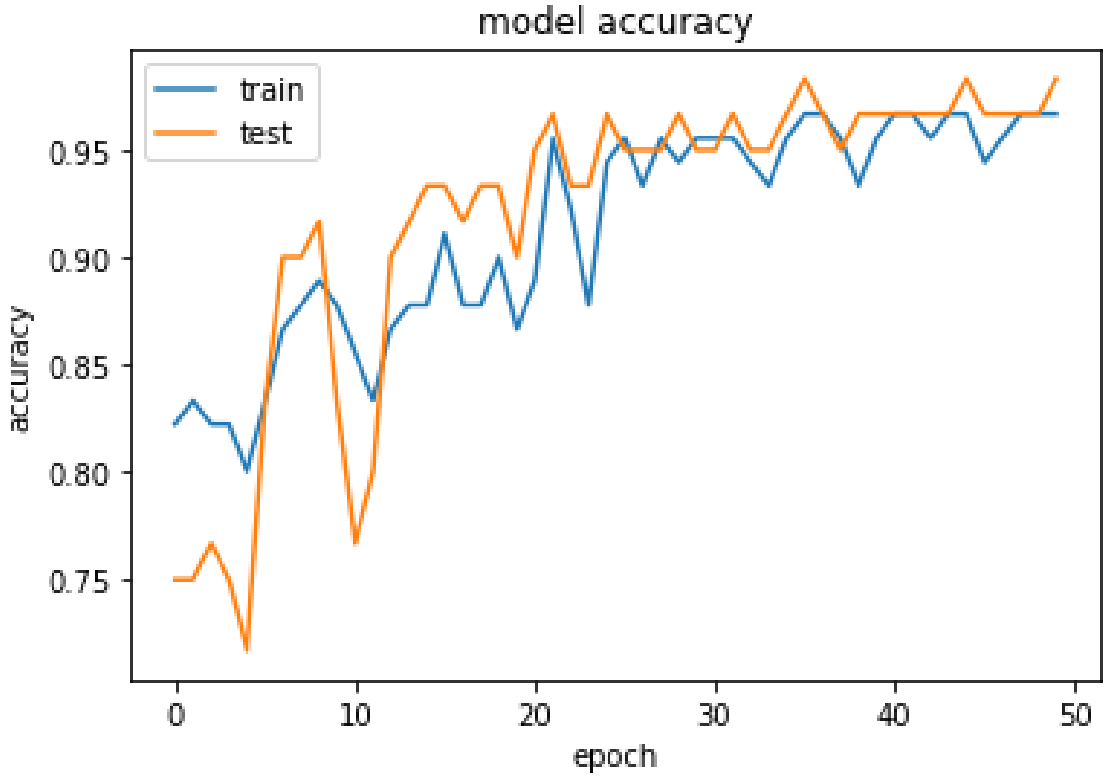
```
>>> print('Test loss:', score[0])
Test loss: 0.09916003793478012
>>> print('Test accuracy:', score[1])
Test accuracy: 0.9833333492279053
```

3. 다층신경망(DNN) 예제

```
# summarize history for accuracy
plt.plot(disp.history['accuracy'])
plt.plot(disp.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



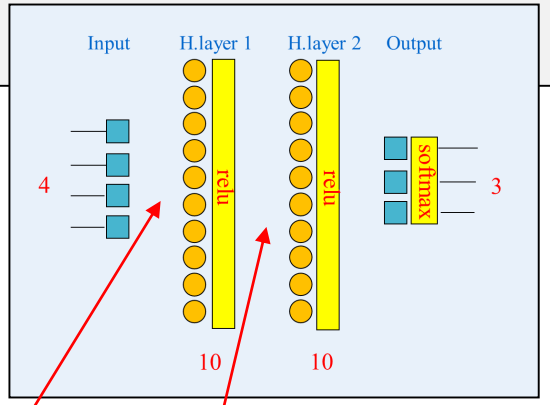
3. 다층신경망(DNN) 예제



Note. 이 코드는 실행할 때 마다 weight value 의 초기값을 random 하게 할당하기 때문에 학습할 때 마다 결과가 다르게 나온다.

3. 다층신경망(DNN) 예제

```
# model weights
for lay in model.layers:
    print(lay.name)
    print(lay.get_weights())
```




```
In [104]: for lay in model.layers:
     ...:     print(lay.name)
     ...:     print(lay.get_weights())
dense_32
[array([[ 0.5289092 , -0.15780598,  0.01167274, -0.02926999,  0.05397177,
          0.43013152,  0.17118984, -0.38582698,  0.5064401 ,  0.3571461 ],
        [ 0.3323511 , -0.33000568,  0.04344529, -0.31324032, -0.21702161,
          0.5186461 ,  0.8281973 ,  0.4229083 ,  0.10245298, -0.12464952],
        [ 0.56385374,  0.08948827, -0.29152125, -0.49173585, -0.1845164 ,
         -0.31305426, -0.64084774,  1.0362344 , -0.448405  , -0.4769767 ],
        [-0.11043015, -0.15350175, -0.45630628, -0.45117027, -0.21350479,
         -0.0442345 , -0.07903751,  0.9940308 , -0.5409244 ,  0.00975268]],
       dtype=float32), array([ 0.12229796,  0.        ,  0.        ,  0.        ,  0.        ,
        -0.02556496,  0.35797217, -0.20785096,  0.23644954,  0.12502621],
       dtype=float32)]
dense_33
[array([[ 0.35183033, -0.34312344, -0.31698382, -0.17515983,  0.28551614,
          0.47306767,  0.14369607,  0.26379722, -0.4349746 ,  0.54658985],
```

4 x 10

10 x 10

[Note]

- Save model & reload

```
# save model
model.save('path/to/location')

# load model
from tensorflow import keras
model = keras.models.load_model('path/to/location')
```

Reference:
https://www.tensorflow.org/guide/keras/save_and_serialize

[Note] Visualize model

```
> pip install pydot
> pip install graphviz
```

설치 후 VS Code (python)을 재실행 한다.

```
import tensorflow as tf
from matplotlib import pyplot as plt
from matplotlib import image as mpimg

tf.keras.utils.plot_model(model,
                          show_shapes=True,
                          to_file='d:/model.png')

image = mpimg.imread("d:/model.png")
plt.imshow(image)
plt.show()
```

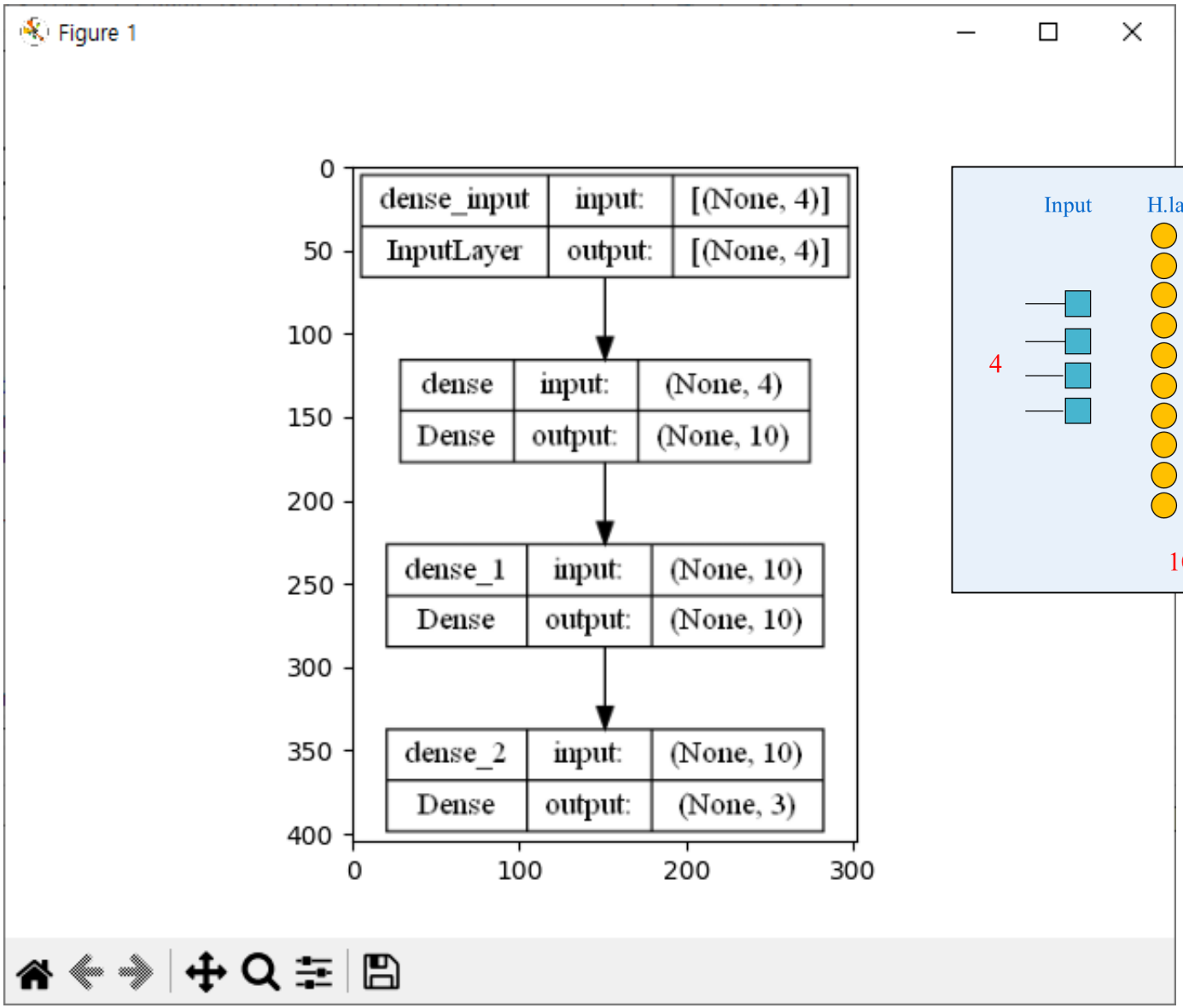
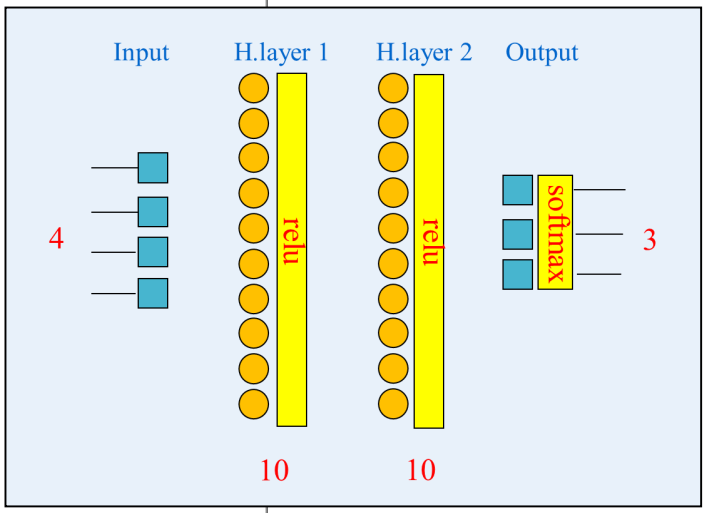


Figure 1
dense_input input: [(None, 4)]
InputLayer output: [(None, 4)]
dense input: (None, 4)
Dense output: (None, 10)
dense_1 input: (None, 10)
Dense output: (None, 10)
dense_2 input: (None, 10)
Dense output: (None, 3)
0
50
100
150
200
250
300
350
400
0
100
200
300


Input
H.layer 1
H.layer 2
Output
4
relu
relu
softmax
3
10
10

5. Keras 제공 함수

- Initializers

https://keras.io/ko/initializers/

  - Zeros
  - Ones
  - Constant
  - RandomNormal
  - RandomUniform
  - glorot_uniform (default)
  - he_normal
  - He_uniform

Initialize weight, bias values

```
model.add(Dense(64,
                kernel_initializer='random_uniform',
                bias_initializer='zeros'))
```

5. Keras 제공 함수

- Activation 함수 https://keras.io/activations/
  - softmax
  - relu
  - tanh
  - sigmoid

  - elu
  - selu
  - softplus
  - softsign
  - hard_sigmoid
  - linear (default. No activation)

```
model.add(Dense(64, activation='tanh'))
```

5. Keras 제공 함수

- Loss function

https://keras.io/losses/

  - mean_squared_error
  - categorical_crossentropy
  - mean_absolute_error
  - mean_absolute_percentage_error
  - mean_squared_logarithmic_error
  - squared_hinge
  - hinge
  - categorical_hinge
  - logcosh
  - sparse_categorical_crossentropy
  - binary_crossentropy
  - kullback_leibler_divergence
  - poisson
  - cosine_proximity

```
model.compile(loss='mean_squared_error', optimizer='sgd')
```

5. Keras 제공 함수

- Optimizer
  - sgd
  - RMSprop
  - Adagrad
  - Adadelta
  - Adam
  - Adamax
  - Nadam

https://keras.io/optimizers/

Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses.

```
from keras import optimizers

model = Sequential()
model.add(Dense(64, kernel_initializer='uniform', input_shape=(10,)))
model.add(Activation('softmax'))

sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='mean_squared_error', optimizer=sgd)
```

5. Keras 제공 함수

- Metric

https://keras.io/metrics/

  - acc
  - binary_accuracy
  - categorical_accuracy
  - sparse_categorical_accuracy
  - top_k_categorical_accuracy
  - sparse_top_k_categorical_accuracy

A metric is a function that is used to judge the performance of your model

```
model.compile(loss='mean_squared_error',
              optimizer='sgd',
              metrics=['mae', 'acc'])
```

[Note]모델의 재현성(reproducibility) 문제

- 모델을 훈련(fitting) 할 때 마다 결과가 다르게 나옴
- Weight matrix random하게 초기화되기 때문임
- global seed와 local seed를 이용하여 해결 가능 (두가지 모두를 설정해야함)
  - Global seed : 프로그램 전체에 적용됨
  - Local seed: 함수 내에서 seed를 이용하는 경우 (ex: random.uniform, tf.keras.initializers.GlorotUniform)
  - GPU나 multi-core를 이용하는 경우는 해결 방법 없음

[Note]모델의 재현성(reproducibility) 문제

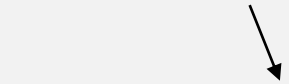
```
# define model (DNN structure)
epochs = 50
batch_size = 10

import tensorflow as tf
tf.random.set_seed(123)   # global seed
initializer = tf.keras.initializers.RandomUniform(seed=123)

model = Sequential()
model.add(Dense(10, input_dim=4, activation='relu',
                kernel_initializer=initializer))
model.add(Dense(10, activation='relu',
                kernel_initializer=initializer))
model.add(Dense(3, activation='softmax',
                kernel_initializer=initializer))
model.summary()  # show model structure

# Compile model
```

Local seed



[실습 1]

- 1. 예제 소스코드를 활용하여 liver.csv 데이터셋에 대한 classification 모델을 만들고 테스트 하시오. (train:test = 6:4) 첫번째 컬럼이 class label

- 2. 예제에 hidden layer 를 한층 더 추가 하되 node 수는 8 로 하고, activation 은 relu 함수를 적용하여 테스트 하시오 (liver.csv 데이터셋)

- 3. 2번문제에서 epoch 를 100, 150, 200 으로 변경하여 시행한 뒤 변경전과 결과(test dataset에 대한 loss, accuracy)를 비교하여 보시오