Computer Science, CNU

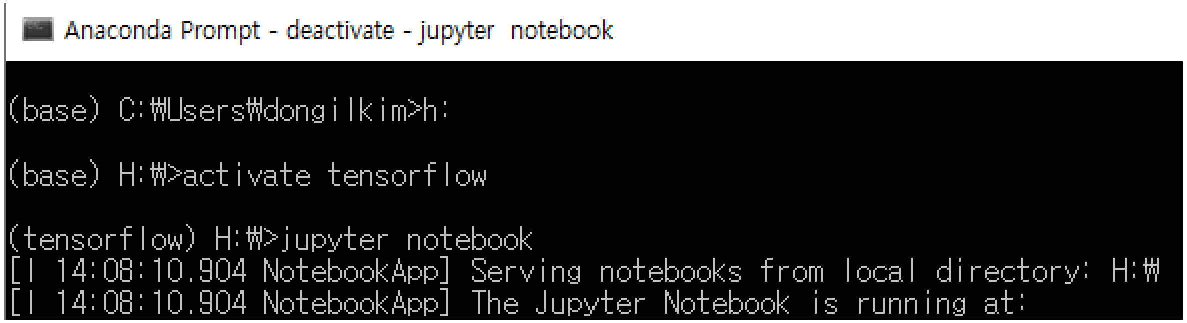
Lab 6: Neural Networks and Deep MLP

Dongil Kim

Dept. of Computer Science and Engineering

Chungnam National University

Python Launch



Anaconda Prompt - deactivate - jupyter notebook

```
(base) C:₩Users₩dongilkim>h:

(base) H:₩>activate tensorflow

(tensorflow) H:₩>jupyter notebook
[I 14:08:10.904 NotebookApp] Serving notebooks from local directory: H:₩
[I 14:08:10.904 NotebookApp] The Jupyter Notebook is running at:
```

이번 과정에서 Tensorflow를 활용할 예정입니다.

따라서 Lab 소개에서 말씀드린 방식으로
Tensorflow를 독립된 환경에서 설치해주시고
그 환경에 들어가서 개발환경을 실행시켜주시기 바랍니다.

문제

❖ Iris Data

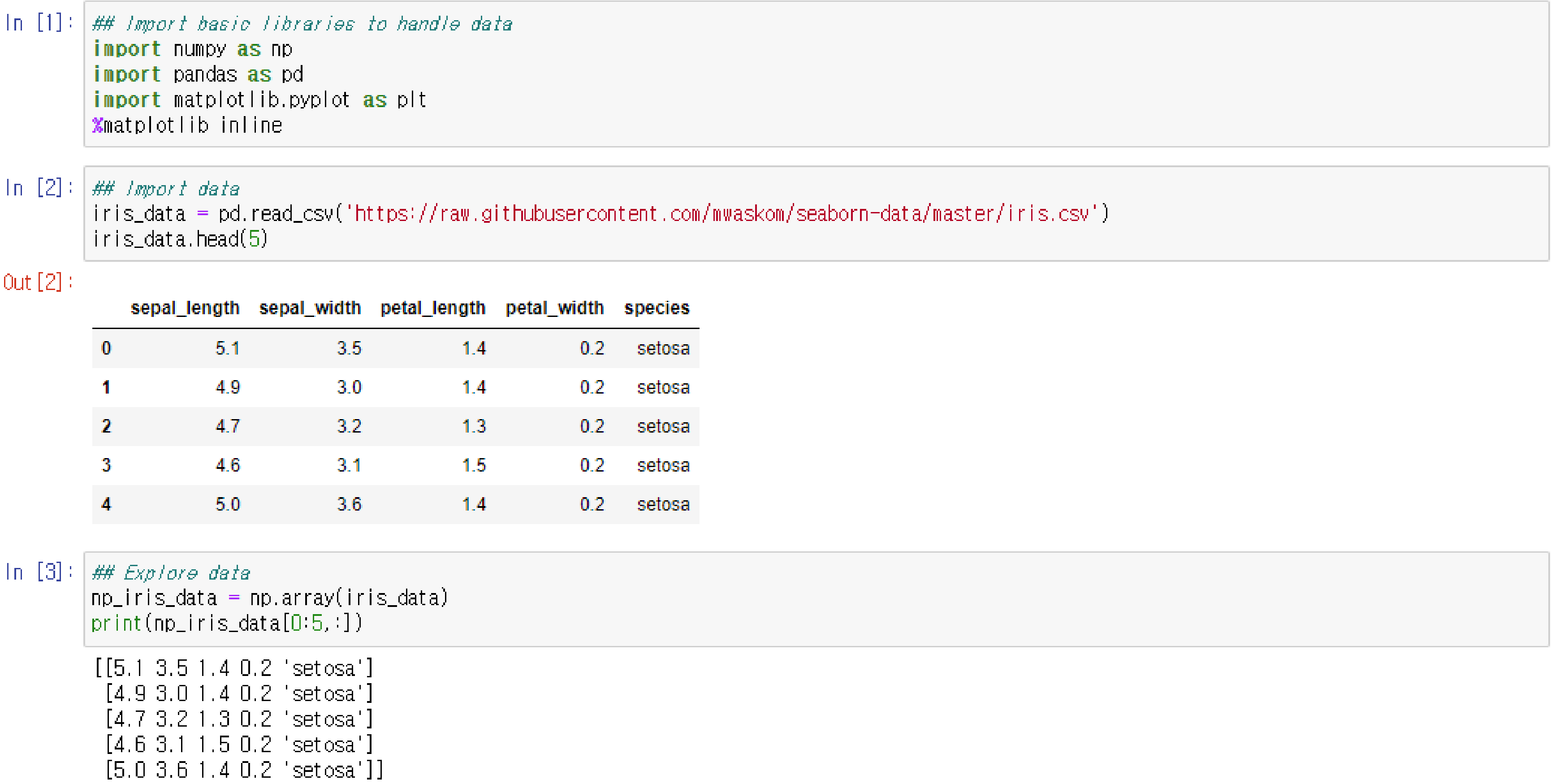
- 가장 쉽게 사용하는 toy example
- 3가지 종류의 Iris 꽃의 특성에 대한 데이터

Iris Versicolor

Iris Virginica

Iris Setosa

Data Import(Revised)



In [1]:
```
## Import basic libraries to handle data
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
## Import data
iris_data = pd.read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv')
iris_data.head(5)
```

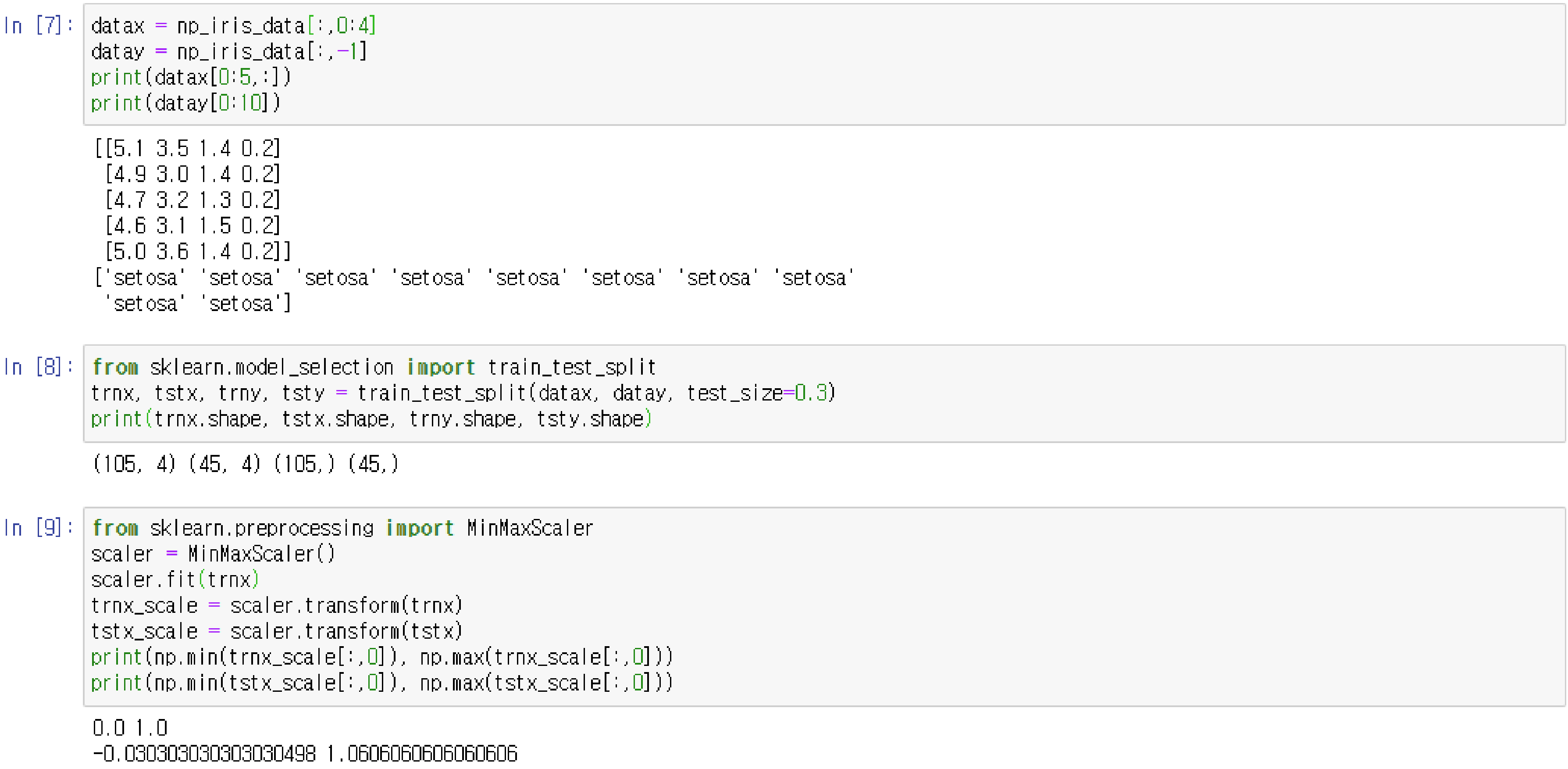
Out[2]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [3]:
```
## Explore data
np_iris_data = np.array(iris_data)
print(np_iris_data[0:5,:])
```

```
[[5.1 3.5 1.4 0.2 'setosa']
 [4.9 3.0 1.4 0.2 'setosa']
 [4.7 3.2 1.3 0.2 'setosa']
 [4.6 3.1 1.5 0.2 'setosa']
 [5.0 3.6 1.4 0.2 'setosa']]
```

Data Import(Revised)



```
In [7]: datax = np_iris_data[:,0:4]
        datay = np_iris_data[:,-1]
        print(datax[0:5,:])
        print(datay[0:10])
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.0 1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.0 3.6 1.4 0.2]]
['setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa']
```

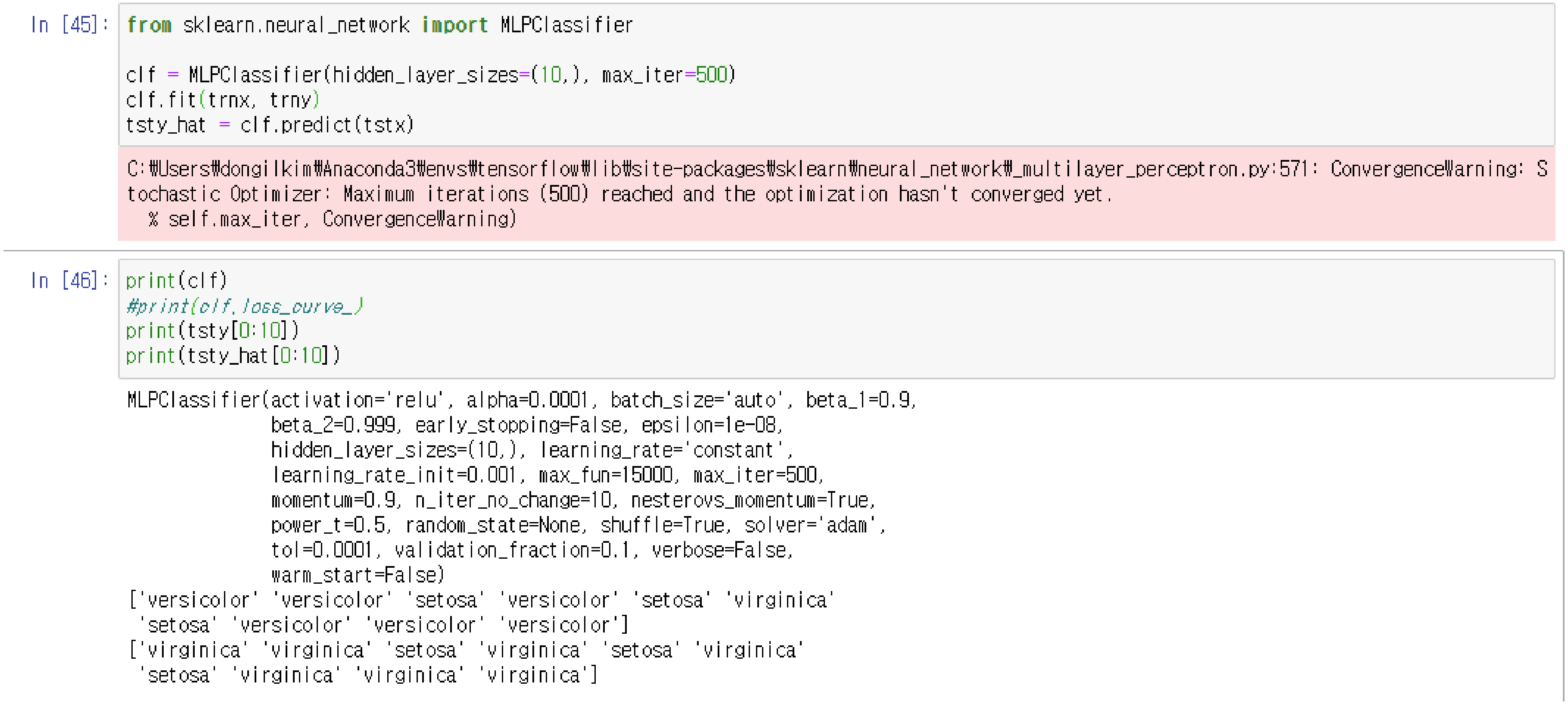
```
In [8]: from sklearn.model_selection import train_test_split
        trnx, tstx, trny, tsty = train_test_split(datax, datay, test_size=0.3)
        print(trnx.shape, tstx.shape, trny.shape, tsty.shape)
```

```
(105, 4) (45, 4) (105,) (45,)
```

```
In [9]: from sklearn.preprocessing import MinMaxScaler
        scaler = MinMaxScaler()
        scaler.fit(trnx)
        trnx_scale = scaler.transform(trnx)
        tstx_scale = scaler.transform(tstx)
        print(np.min(trnx_scale[:,0]), np.max(trnx_scale[:,0]))
        print(np.min(tstx_scale[:,0]), np.max(tstx_scale[:,0]))
```

```
0.0 1.0
-0.030303030303030498 1.0606060606060606
```

NN(Scikit-Learn)



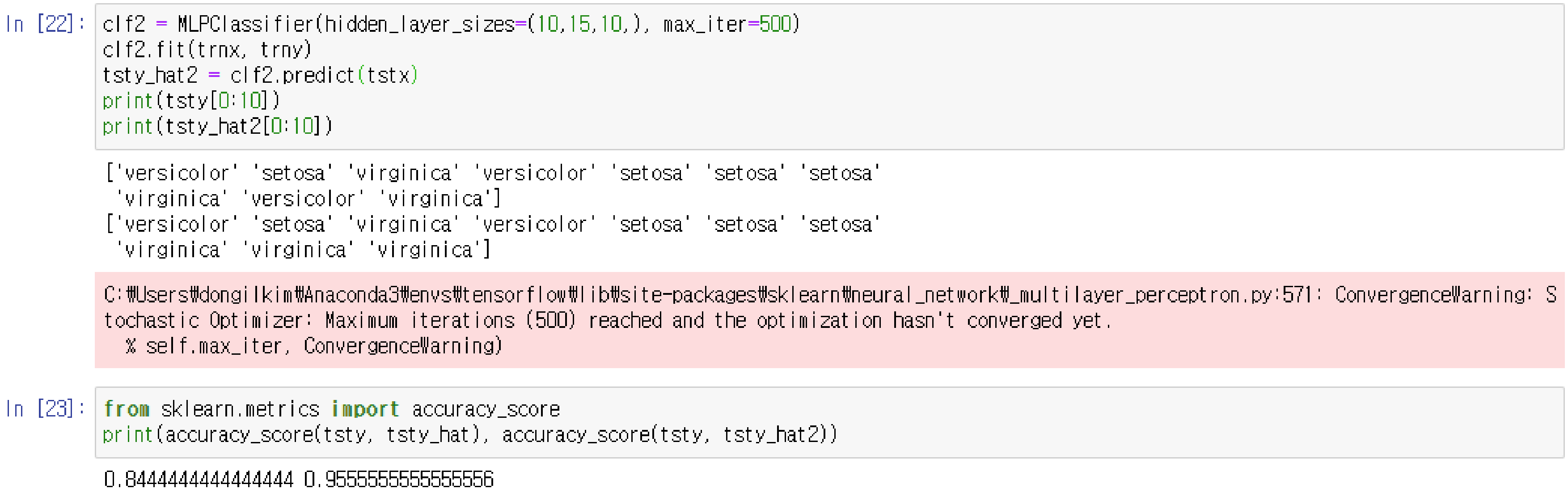
```
In [45]: from sklearn.neural_network import MLPClassifier

         clf = MLPClassifier(hidden_layer_sizes=(10,), max_iter=500)
         clf.fit(trnx, trny)
         tsty_hat = clf.predict(tstx)
```

```
C:₩Users₩dongilkim₩Anaconda3₩envs₩tensorflow₩lib₩site-packages₩sklearn₩neural_network₩_multilayer_perceptron.py:571: ConvergenceWarning: S
tochastic Optimizer: Maximum iterations (500) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
In [46]: print(clf)
         #print(clf.loss_curve_)
         print(tsty[0:10])
         print(tsty_hat[0:10])
```

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(10,), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=500,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=None, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
['versicolor' 'versicolor' 'setosa' 'versicolor' 'setosa' 'virginica'
 'setosa' 'versicolor' 'versicolor' 'versicolor']
['virginica' 'virginica' 'setosa' 'virginica' 'setosa' 'virginica'
 'setosa' 'virginica' 'virginica' 'virginica']
```

NN(Scikit-Learn)



```
In [22]: clf2 = MLPClassifier(hidden_layer_sizes=(10,15,10,), max_iter=500)
         clf2.fit(trnx, trny)
         tsty_hat2 = clf2.predict(tstx)
         print(tsty[0:10])
         print(tsty_hat2[0:10])
```

```
['versicolor' 'setosa' 'virginica' 'versicolor' 'setosa' 'setosa' 'setosa'
 'virginica' 'versicolor' 'virginica']
['versicolor' 'setosa' 'virginica' 'versicolor' 'setosa' 'setosa' 'setosa'
 'virginica' 'virginica' 'virginica']
```

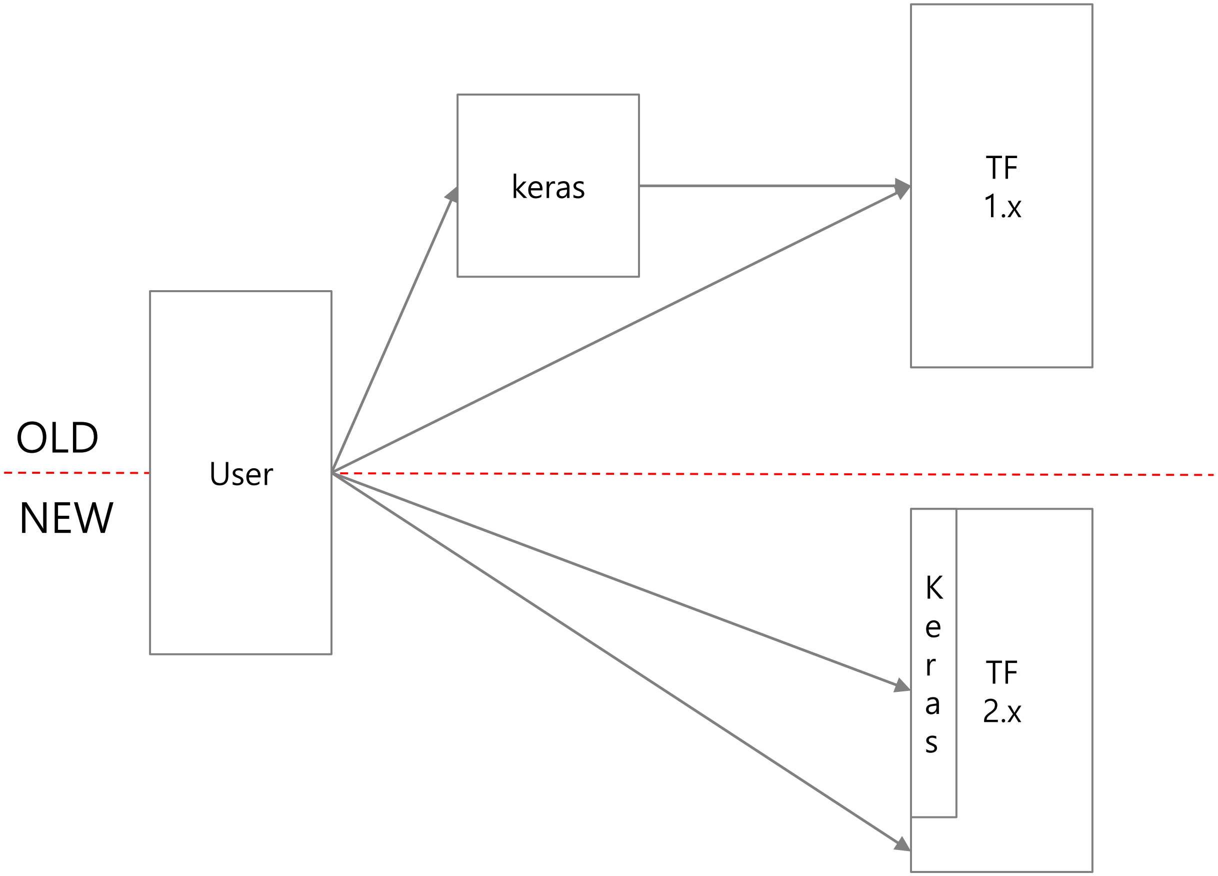
```
C:₩Users₩dongilkim₩Anaconda3₩envs₩tensorflow₩lib₩site-packages₩sklearn₩neural_network₩_multilayer_perceptron.py:571: ConvergenceWarning: S
tochastic Optimizer: Maximum iterations (500) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
In [23]: from sklearn.metrics import accuracy_score
         print(accuracy_score(tsty, tsty_hat), accuracy_score(tsty, tsty_hat2))
```

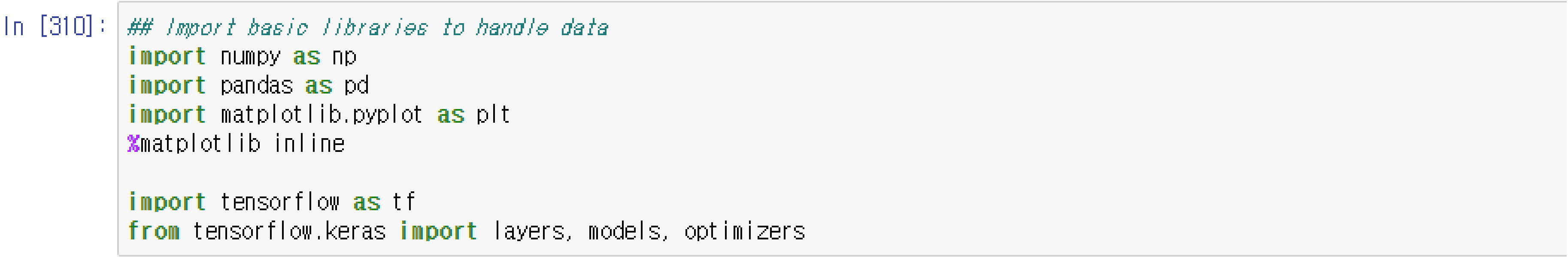
```
0.8444444444444444 0.9555555555555556
```

(참고) TF and Keras

❖ Keras at Tensorflow 2.x



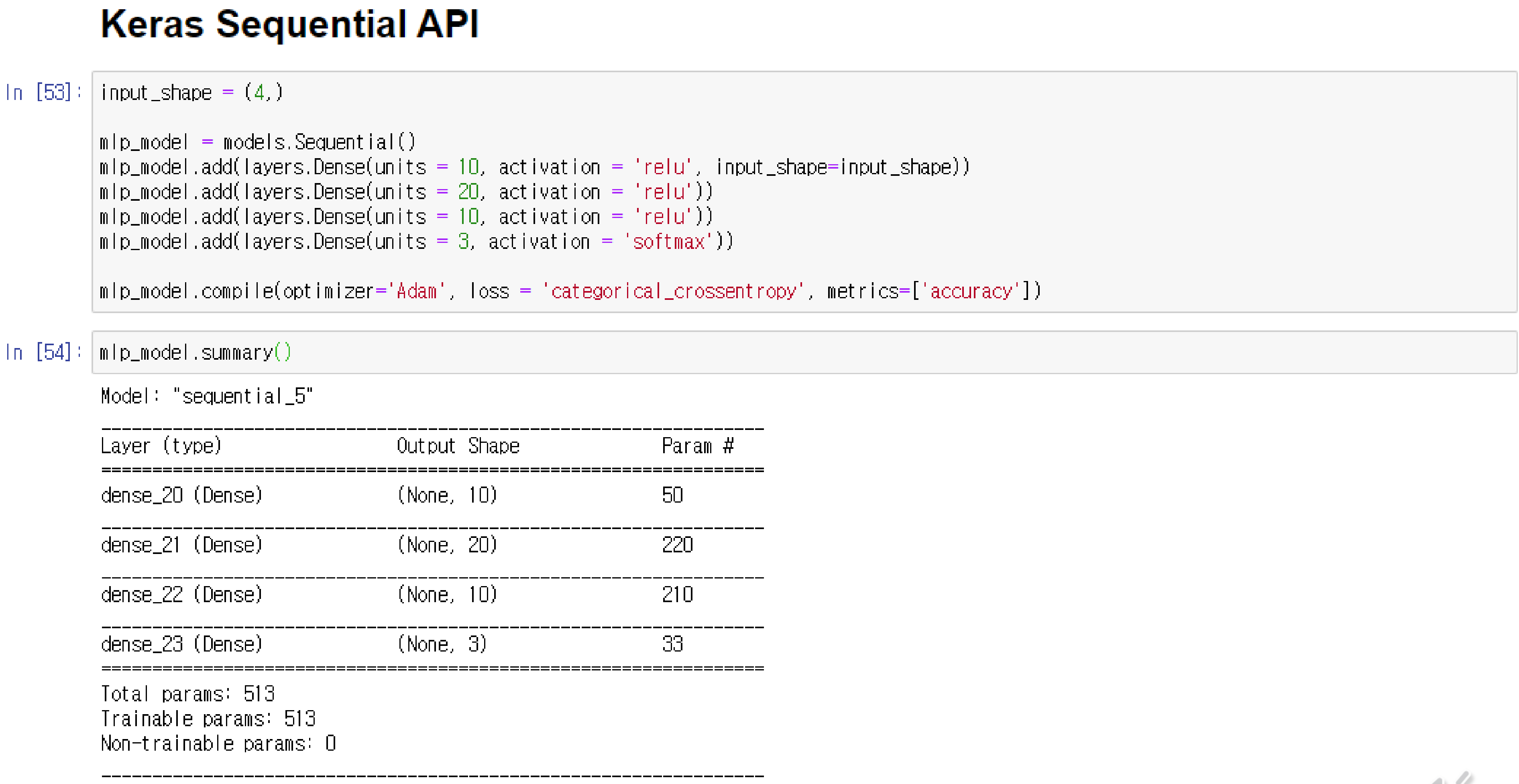
NN(Tensorflow-Keras)



In [310]:

```
## Import basic libraries to handle data
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

import tensorflow as tf
from tensorflow.keras import layers, models, optimizers
```

NN(Tensorflow-Keras-Sequential API)



Keras Sequential API

In [53]:

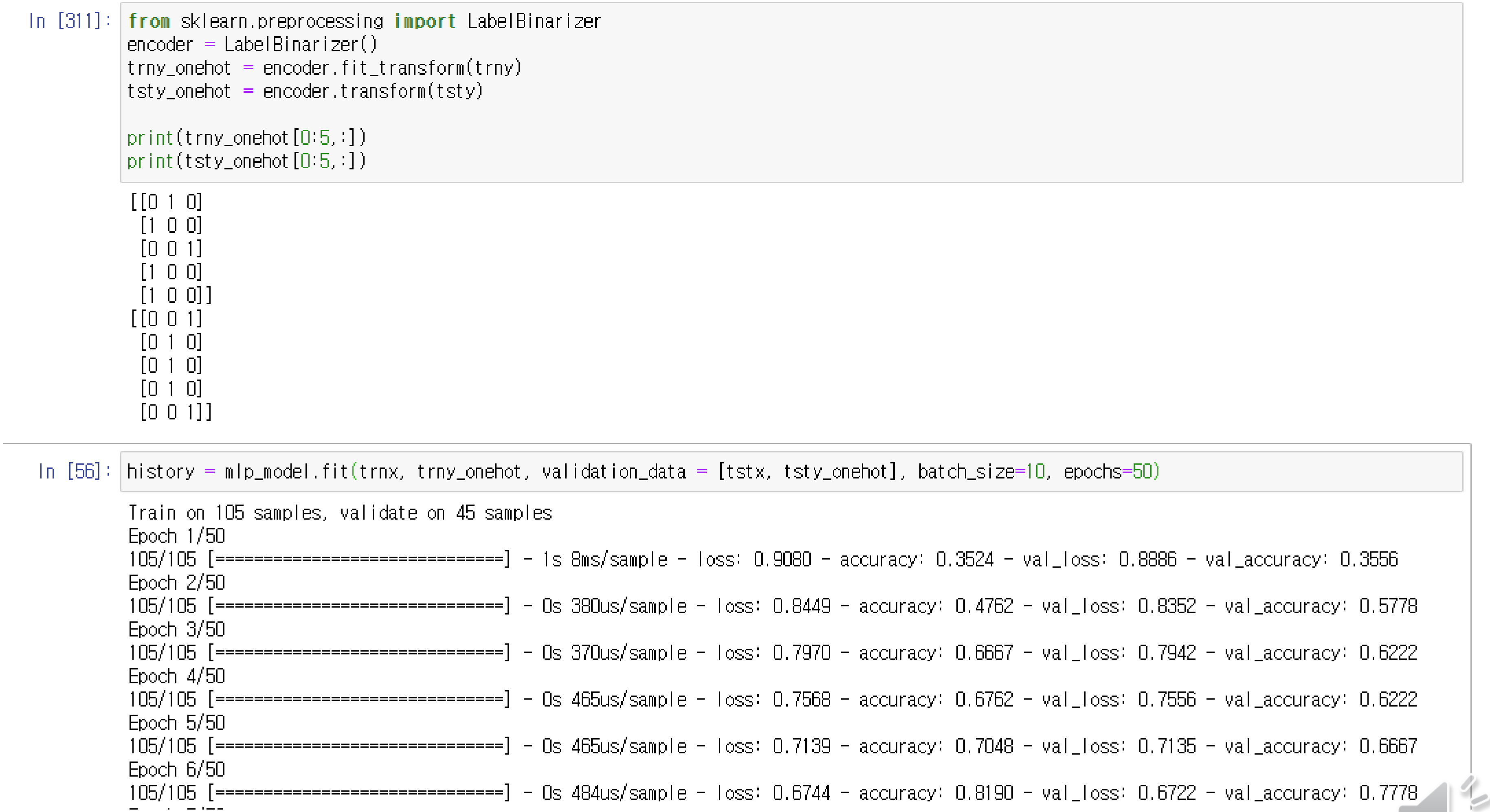
```
input_shape = (4,)

mlp_model = models.Sequential()
mlp_model.add(layers.Dense(units = 10, activation = 'relu', input_shape=input_shape))
mlp_model.add(layers.Dense(units = 20, activation = 'relu'))
mlp_model.add(layers.Dense(units = 10, activation = 'relu'))
mlp_model.add(layers.Dense(units = 3, activation = 'softmax'))

mlp_model.compile(optimizer='Adam', loss = 'categorical_crossentropy', metrics=['accuracy'])
```

In [54]:

```
mlp_model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_20 (Dense)             (None, 10)                50
_________________________________________________________________
dense_21 (Dense)             (None, 20)                220
_________________________________________________________________
dense_22 (Dense)             (None, 10)                210
_________________________________________________________________
dense_23 (Dense)             (None, 3)                 33
=================================================================
Total params: 513
Trainable params: 513
Non-trainable params: 0
_________________________________________________________________
```

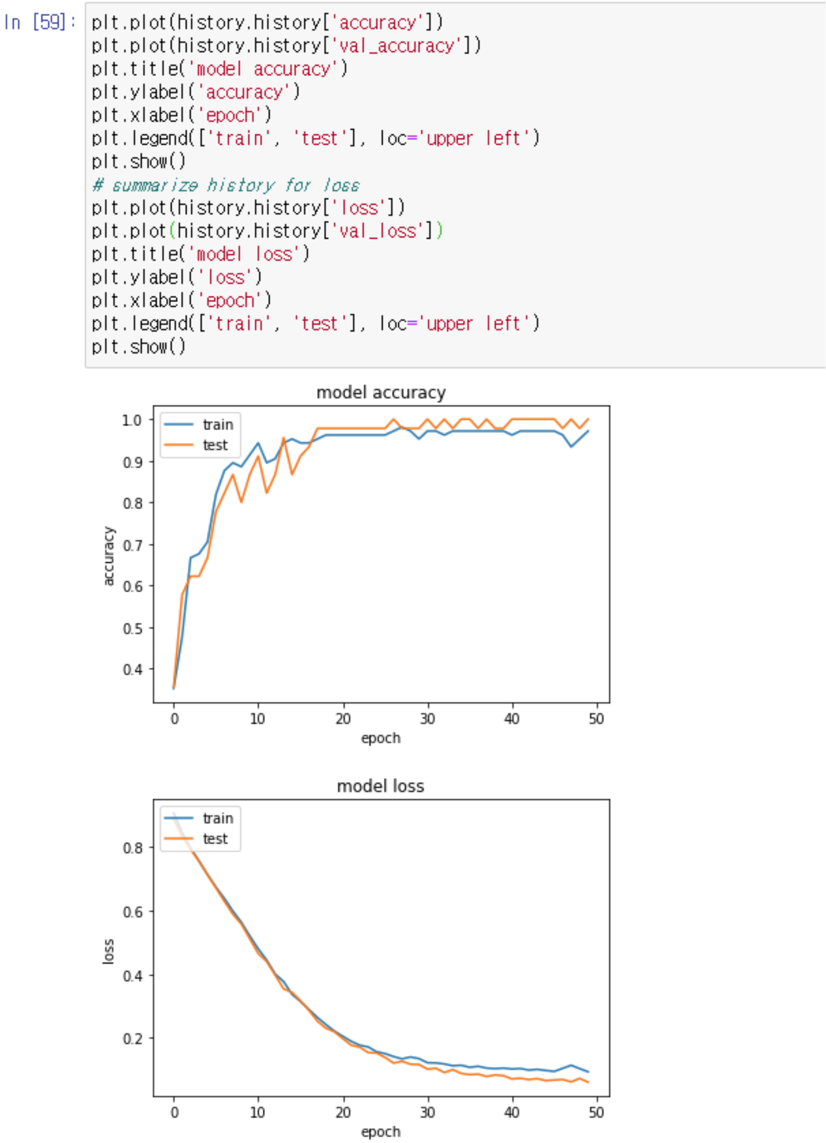
NN(Tensorflow-Keras-Sequential API)



```
In [311]: from sklearn.preprocessing import LabelBinarizer
          encoder = LabelBinarizer()
          trny_onehot = encoder.fit_transform(trny)
          tsty_onehot = encoder.transform(tsty)

          print(trny_onehot[0:5,:])
          print(tsty_onehot[0:5,:])

          [[0 1 0]
           [1 0 0]
           [0 0 1]
           [1 0 0]
           [1 0 0]]
          [[0 0 1]
           [0 1 0]
           [0 1 0]
           [0 1 0]
           [0 0 1]]
```

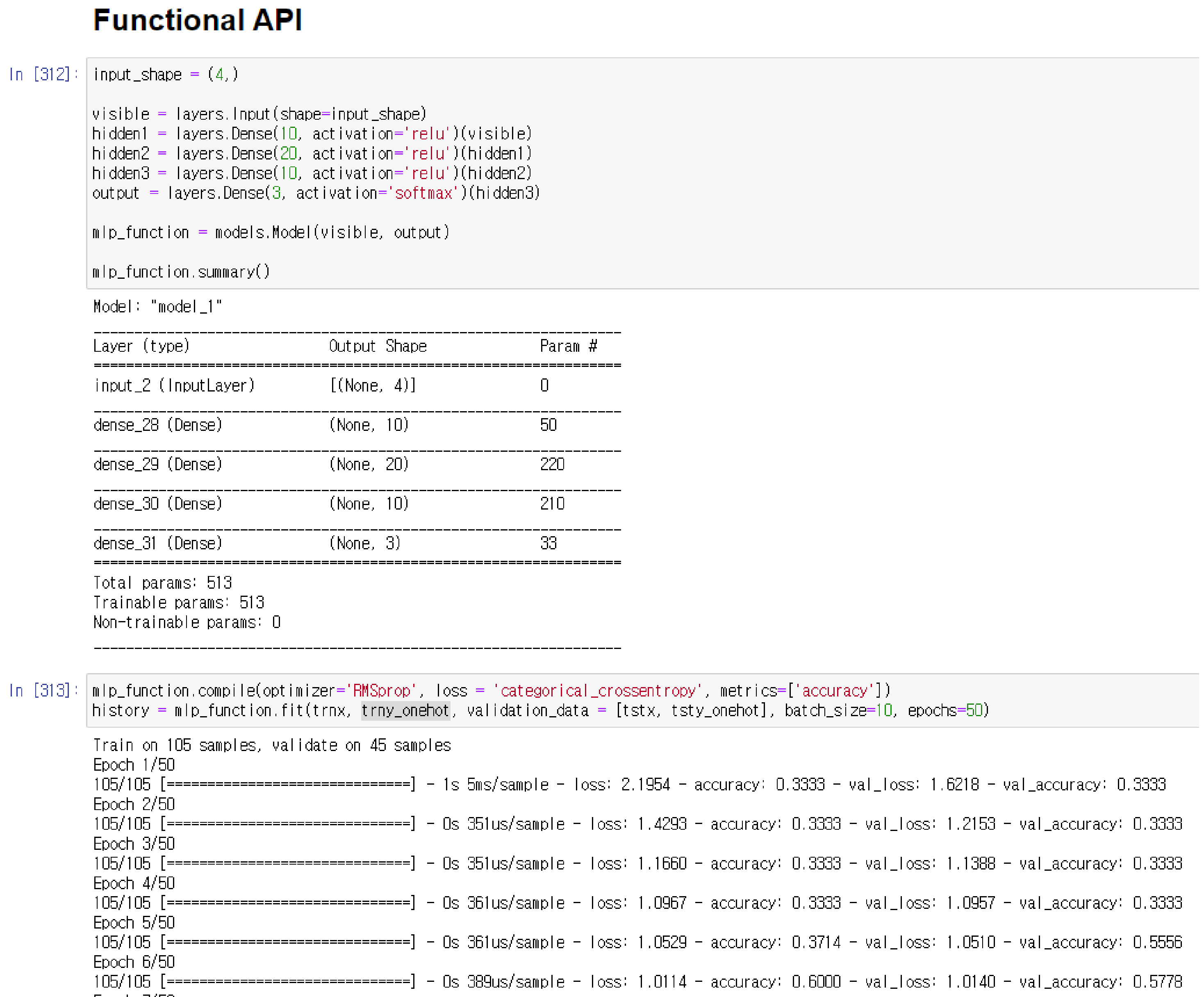
```
In [56]: history = mlp_model.fit(trnx, trny_onehot, validation_data = [tstx, tsty_onehot], batch_size=10, epochs=50)

         Train on 105 samples, validate on 45 samples
         Epoch 1/50
         105/105 [==============================] - 1s 8ms/sample - loss: 0.9080 - accuracy: 0.3524 - val_loss: 0.8886 - val_accuracy: 0.3556
         Epoch 2/50
         105/105 [==============================] - 0s 380us/sample - loss: 0.8449 - accuracy: 0.4762 - val_loss: 0.8352 - val_accuracy: 0.5778
         Epoch 3/50
         105/105 [==============================] - 0s 370us/sample - loss: 0.7970 - accuracy: 0.6667 - val_loss: 0.7942 - val_accuracy: 0.6222
         Epoch 4/50
         105/105 [==============================] - 0s 465us/sample - loss: 0.7568 - accuracy: 0.6762 - val_loss: 0.7556 - val_accuracy: 0.6222
         Epoch 5/50
         105/105 [==============================] - 0s 465us/sample - loss: 0.7139 - accuracy: 0.7048 - val_loss: 0.7135 - val_accuracy: 0.6667
         Epoch 6/50
         105/105 [==============================] - 0s 484us/sample - loss: 0.6744 - accuracy: 0.8190 - val_loss: 0.6722 - val_accuracy: 0.7778
```

NN(Tensorflow-Keras-Sequential API)



```
In [59]: plt.plot(history.history['accuracy'])
         plt.plot(history.history['val_accuracy'])
         plt.title('model accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
         plt.legend(['train', 'test'], loc='upper left')
         plt.show()
         # summarize history for loss
         plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])
         plt.title('model loss')
         plt.ylabel('loss')
         plt.xlabel('epoch')
         plt.legend(['train', 'test'], loc='upper left')
         plt.show()
```

NN(Tensorflow-Keras-Functional API)



Functional API

In [312]:

```
input_shape = (4,)

visible = layers.Input(shape=input_shape)
hidden1 = layers.Dense(10, activation='relu')(visible)
hidden2 = layers.Dense(20, activation='relu')(hidden1)
hidden3 = layers.Dense(10, activation='relu')(hidden2)
output  = layers.Dense(3, activation='softmax')(hidden3)

mlp_function = models.Model(visible, output)

mlp_function.summary()
```

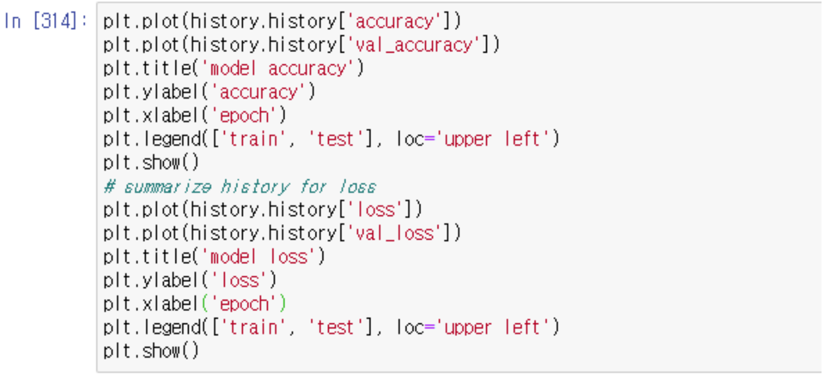
```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 4)]               0
_________________________________________________________________
dense_28 (Dense)             (None, 10)                50
_________________________________________________________________
dense_29 (Dense)             (None, 20)                220
_________________________________________________________________
dense_30 (Dense)             (None, 10)                210
_________________________________________________________________
dense_31 (Dense)             (None, 3)                 33
=================================================================
Total params: 513
Trainable params: 513
Non-trainable params: 0
_________________________________________________________________
```

In [313]:

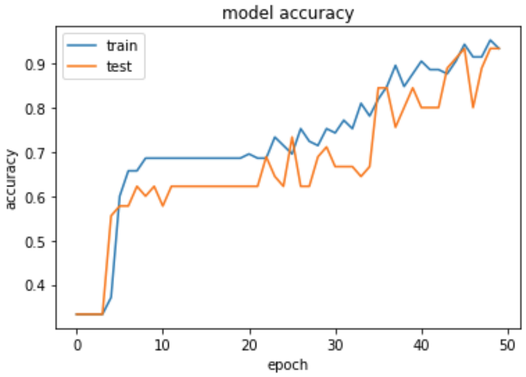
```
mlp_function.compile(optimizer='RMSprop', loss = 'categorical_crossentropy', metrics=['accuracy'])
history = mlp_function.fit(trnx, trny_onehot, validation_data = [tstx, tsty_onehot], batch_size=10, epochs=50)
```

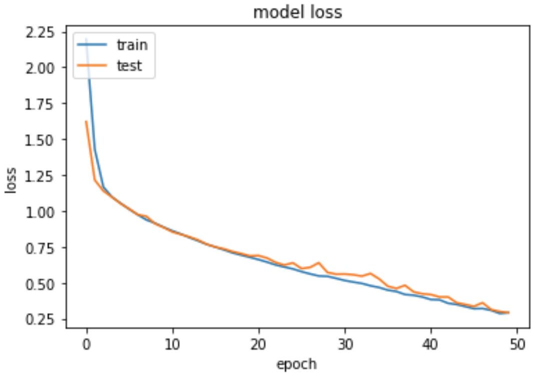
```
Train on 105 samples, validate on 45 samples
Epoch 1/50
105/105 [==============================] - 1s 5ms/sample - loss: 2.1954 - accuracy: 0.3333 - val_loss: 1.6218 - val_accuracy: 0.3333
Epoch 2/50
105/105 [==============================] - 0s 351us/sample - loss: 1.4293 - accuracy: 0.3333 - val_loss: 1.2153 - val_accuracy: 0.3333
Epoch 3/50
105/105 [==============================] - 0s 351us/sample - loss: 1.1660 - accuracy: 0.3333 - val_loss: 1.1388 - val_accuracy: 0.3333
Epoch 4/50
105/105 [==============================] - 0s 361us/sample - loss: 1.0967 - accuracy: 0.3333 - val_loss: 1.0957 - val_accuracy: 0.3333
Epoch 5/50
105/105 [==============================] - 0s 361us/sample - loss: 1.0529 - accuracy: 0.3714 - val_loss: 1.0510 - val_accuracy: 0.5556
Epoch 6/50
105/105 [==============================] - 0s 389us/sample - loss: 1.0114 - accuracy: 0.6000 - val_loss: 1.0140 - val_accuracy: 0.5778
```

NN(Tensorflow-Keras-Functional API)



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





Q&A