import numpy as np

X = np.array([['Korea', 44, 7200],

['Japan', 27, 4800],

['China', 30, 6100]])

from sklearn.preprocessing import OneHotEncoder

onehotencoder = OneHotEncoder()

# 원하는 열을 뽑아서 2차원 배열로 만들어서 전달하여야 한다.

XX = onehotencoder.fit\_transform(X[:,0].reshape(-1,1)).toarray()

print(XX)

X = np.delete(X, [0], axis=1) # 0번째 열 삭제

X = np.concatenate((XX, X), axis = 1) # X와 XX를 붙인다.

print(X)

[[0. 0. 1.] [0. 1. 0.] [1. 0. 0.]] [['0.0' '0.0' '1.0' '44' '7200'] ['0.0' '1.0' '0.0' '27' '4800'] ['1.0' '0.0' '0.0' '30' '6100']]

class\_vector =[2, 6, 6, 1]

from tensorflow.keras.utils import to\_categorical

output = to\_categorical(class\_vector, num\_classes = 7, dtype ="int32")

print(output)

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

from sklearn.preprocessing import MinMaxScaler

data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]

scaler = MinMaxScaler()

scaler.fit(data) # 최대값과 최소값을 알아낸다.

print(scaler.transform(data))

[[0. 0. ] [0.25 0.25] [0.5 0.5 ] [1. 1. ]]

import numpy as numpy

import tensorflow as tf

import matplotlib.pyplot as plt

# 데이터 다운로드

(train\_data, train\_labels), (test\_data, test\_labels) = \

tf.keras.datasets.imdb.load\_data( num\_words=1000)

# 원-핫 인코딩으로 변환하는 함수

def one\_hot\_sequences(sequences, dimension=1000):

results = numpy.zeros((len(sequences), dimension))

for i, word\_index in enumerate(sequences):

results[i, word\_index] = 1.

return results

train\_data = one\_hot\_sequences(train\_data)

test\_data = one\_hot\_sequences(test\_data)

model = tf.keras.Sequential()

model.add(tf.keras.layers.Dense(16, activation='relu', input\_shape=(1000,)))

model.add(tf.keras.layers.Dense(16, activation='relu'))

model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam',

metrics=['accuracy'])

# 신경망 훈련, 검증 데이터 전달

history = model.fit(train\_data,

train\_labels,

epochs=20,

batch\_size=512,

validation\_data=(test\_data, test\_labels),

verbose=2)

history\_dict = history.history

loss\_values = history\_dict['loss'] # 훈련 데이터 손실값

val\_loss\_values = history\_dict['val\_loss'] # 검증 데이터 손실값

acc = history\_dict['accuracy'] # 정확도

epochs = range(1, len(acc) + 1) # 에포크 수

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Loss Plot')

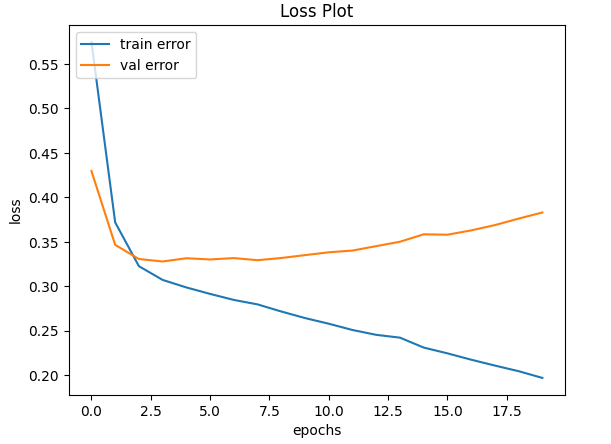
plt.ylabel('loss')

plt.xlabel('epochs')

plt.legend(['train error', 'val error'], loc='upper left')

plt.show()

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz> 17464789/17464789 [==============================] - 0s 0us/step Epoch 1/20 49/49 - 1s - loss: 0.5748 - accuracy: 0.7082 - val\_loss: 0.4297 - val\_accuracy: 0.8196 - 1s/epoch - 29ms/step Epoch 2/20 49/49 - 0s - loss: 0.3719 - accuracy: 0.8434 - val\_loss: 0.3466 - val\_accuracy: 0.8553 - 300ms/epoch - 6ms/step Epoch 3/20 49/49 - 0s - loss: 0.3226 - accuracy: 0.8652 - val\_loss: 0.3306 - val\_accuracy: 0.8592 - 272ms/epoch - 6ms/step Epoch 4/20 49/49 - 0s - loss: 0.3071 - accuracy: 0.8735 - val\_loss: 0.3279 - val\_accuracy: 0.8607 - 211ms/epoch - 4ms/step Epoch 5/20 49/49 - 0s - loss: 0.2987 - accuracy: 0.8759 - val\_loss: 0.3315 - val\_accuracy: 0.8597 - 193ms/epoch - 4ms/step Epoch 6/20 49/49 - 0s - loss: 0.2914 - accuracy: 0.8808 - val\_loss: 0.3302 - val\_accuracy: 0.8595 - 184ms/epoch - 4ms/step Epoch 7/20 49/49 - 0s - loss: 0.2846 - accuracy: 0.8838 - val\_loss: 0.3317 - val\_accuracy: 0.8586 - 259ms/epoch - 5ms/step Epoch 8/20 49/49 - 0s - loss: 0.2796 - accuracy: 0.8858 - val\_loss: 0.3293 - val\_accuracy: 0.8592 - 280ms/epoch - 6ms/step Epoch 9/20 49/49 - 0s - loss: 0.2716 - accuracy: 0.8890 - val\_loss: 0.3318 - val\_accuracy: 0.8595 - 325ms/epoch - 7ms/step Epoch 10/20 49/49 - 0s - loss: 0.2642 - accuracy: 0.8920 - val\_loss: 0.3350 - val\_accuracy: 0.8587 - 342ms/epoch - 7ms/step Epoch 11/20 49/49 - 0s - loss: 0.2579 - accuracy: 0.8958 - val\_loss: 0.3382 - val\_accuracy: 0.8582 - 295ms/epoch - 6ms/step Epoch 12/20 49/49 - 0s - loss: 0.2508 - accuracy: 0.8978 - val\_loss: 0.3402 - val\_accuracy: 0.8585 - 358ms/epoch - 7ms/step Epoch 13/20 49/49 - 0s - loss: 0.2454 - accuracy: 0.9004 - val\_loss: 0.3452 - val\_accuracy: 0.8562 - 273ms/epoch - 6ms/step Epoch 14/20 49/49 - 0s - loss: 0.2422 - accuracy: 0.9012 - val\_loss: 0.3501 - val\_accuracy: 0.8548 - 200ms/epoch - 4ms/step Epoch 15/20 49/49 - 0s - loss: 0.2311 - accuracy: 0.9075 - val\_loss: 0.3584 - val\_accuracy: 0.8514 - 194ms/epoch - 4ms/step Epoch 16/20 49/49 - 0s - loss: 0.2246 - accuracy: 0.9111 - val\_loss: 0.3580 - val\_accuracy: 0.8520 - 198ms/epoch - 4ms/step Epoch 17/20 49/49 - 0s - loss: 0.2174 - accuracy: 0.9146 - val\_loss: 0.3629 - val\_accuracy: 0.8509 - 196ms/epoch - 4ms/step Epoch 18/20 49/49 - 0s - loss: 0.2108 - accuracy: 0.9183 - val\_loss: 0.3688 - val\_accuracy: 0.8498 - 176ms/epoch - 4ms/step Epoch 19/20 49/49 - 0s - loss: 0.2045 - accuracy: 0.9201 - val\_loss: 0.3761 - val\_accuracy: 0.8496 - 189ms/epoch - 4ms/step Epoch 20/20 49/49 - 0s - loss: 0.1969 - accuracy: 0.9242 - val\_loss: 0.3830 - val\_accuracy: 0.8467 - 269ms/epoch - 5ms/step



import numpy as numpy

import tensorflow as tf

import matplotlib.pyplot as plt

# 데이터 다운로드

(train\_data, train\_labels), (test\_data, test\_labels) = \

tf.keras.datasets.imdb.load\_data( num\_words=1000)

# 원-핫 인코딩으로 변환하는 함수

def one\_hot\_sequences(sequences, dimension=1000):

results = numpy.zeros((len(sequences), dimension))

for i, word\_index in enumerate(sequences):

results[i, word\_index] = 1.

return results

train\_data = one\_hot\_sequences(train\_data)

test\_data = one\_hot\_sequences(test\_data)

model = tf.keras.Sequential()

model.add(tf.keras.layers.Dense(16,

kernel\_regularizer=tf.keras.regularizers.l2(0.001),

activation='relu', input\_shape=(1000,)))

model.add(tf.keras.layers.Dense(16,

kernel\_regularizer=tf.keras.regularizers.l2(0.001), activation='relu'))

model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam',

metrics=['accuracy'])

# 신경망 훈련, 검증 데이터 전달

history = model.fit(train\_data,

train\_labels,

epochs=20,

batch\_size=512,

validation\_data=(test\_data, test\_labels),

verbose=2)

history\_dict = history.history

loss\_values = history\_dict['loss'] # 훈련 데이터 손실값

val\_loss\_values = history\_dict['val\_loss'] # 검증 데이터 손실값

acc = history\_dict['accuracy'] # 정확도

epochs = range(1, len(acc) + 1) # 에포크 수

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Loss Plot')

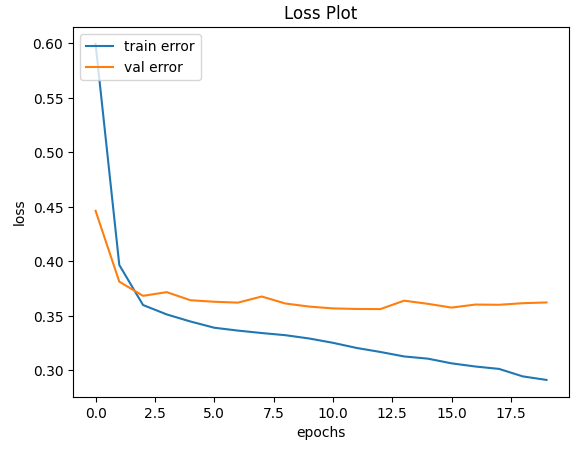
plt.ylabel('loss')

plt.xlabel('epochs')

plt.legend(['train error', 'val error'], loc='upper left')

plt.show()

Epoch 1/20 49/49 - 1s - loss: 0.5994 - accuracy: 0.7244 - val\_loss: 0.4463 - val\_accuracy: 0.8319 - 1s/epoch - 22ms/step Epoch 2/20 49/49 - 0s - loss: 0.3966 - accuracy: 0.8518 - val\_loss: 0.3813 - val\_accuracy: 0.8577 - 322ms/epoch - 7ms/step Epoch 3/20 49/49 - 0s - loss: 0.3599 - accuracy: 0.8675 - val\_loss: 0.3683 - val\_accuracy: 0.8609 - 281ms/epoch - 6ms/step Epoch 4/20 49/49 - 0s - loss: 0.3513 - accuracy: 0.8705 - val\_loss: 0.3717 - val\_accuracy: 0.8596 - 355ms/epoch - 7ms/step Epoch 5/20 49/49 - 0s - loss: 0.3448 - accuracy: 0.8724 - val\_loss: 0.3643 - val\_accuracy: 0.8618 - 269ms/epoch - 5ms/step Epoch 6/20 49/49 - 0s - loss: 0.3391 - accuracy: 0.8736 - val\_loss: 0.3629 - val\_accuracy: 0.8607 - 332ms/epoch - 7ms/step Epoch 7/20 49/49 - 0s - loss: 0.3364 - accuracy: 0.8759 - val\_loss: 0.3621 - val\_accuracy: 0.8609 - 289ms/epoch - 6ms/step Epoch 8/20 49/49 - 0s - loss: 0.3342 - accuracy: 0.8770 - val\_loss: 0.3677 - val\_accuracy: 0.8572 - 281ms/epoch - 6ms/step Epoch 9/20 49/49 - 0s - loss: 0.3322 - accuracy: 0.8761 - val\_loss: 0.3613 - val\_accuracy: 0.8594 - 334ms/epoch - 7ms/step Epoch 10/20 49/49 - 0s - loss: 0.3292 - accuracy: 0.8786 - val\_loss: 0.3585 - val\_accuracy: 0.8603 - 391ms/epoch - 8ms/step Epoch 11/20 49/49 - 0s - loss: 0.3252 - accuracy: 0.8773 - val\_loss: 0.3568 - val\_accuracy: 0.8600 - 375ms/epoch - 8ms/step Epoch 12/20 49/49 - 0s - loss: 0.3205 - accuracy: 0.8804 - val\_loss: 0.3563 - val\_accuracy: 0.8611 - 295ms/epoch - 6ms/step Epoch 13/20 49/49 - 0s - loss: 0.3168 - accuracy: 0.8800 - val\_loss: 0.3562 - val\_accuracy: 0.8606 - 280ms/epoch - 6ms/step Epoch 14/20 49/49 - 0s - loss: 0.3127 - accuracy: 0.8822 - val\_loss: 0.3639 - val\_accuracy: 0.8572 - 316ms/epoch - 6ms/step Epoch 15/20 49/49 - 0s - loss: 0.3107 - accuracy: 0.8830 - val\_loss: 0.3610 - val\_accuracy: 0.8584 - 405ms/epoch - 8ms/step Epoch 16/20 49/49 - 0s - loss: 0.3064 - accuracy: 0.8846 - val\_loss: 0.3576 - val\_accuracy: 0.8598 - 210ms/epoch - 4ms/step Epoch 17/20 49/49 - 0s - loss: 0.3035 - accuracy: 0.8866 - val\_loss: 0.3603 - val\_accuracy: 0.8574 - 193ms/epoch - 4ms/step Epoch 18/20 49/49 - 0s - loss: 0.3013 - accuracy: 0.8861 - val\_loss: 0.3601 - val\_accuracy: 0.8586 - 282ms/epoch - 6ms/step Epoch 19/20 49/49 - 0s - loss: 0.2944 - accuracy: 0.8915 - val\_loss: 0.3616 - val\_accuracy: 0.8574 - 212ms/epoch - 4ms/step Epoch 20/20 49/49 - 0s - loss: 0.2912 - accuracy: 0.8923 - val\_loss: 0.3623 - val\_accuracy: 0.8567 - 189ms/epoch - 4ms/step



import numpy as numpy

import tensorflow as tf

import matplotlib.pyplot as plt

# 데이터 다운로드

(train\_data, train\_labels), (test\_data, test\_labels) = \

tf.keras.datasets.imdb.load\_data( num\_words=1000)

# 원-핫 인코딩으로 변환하는 함수

def one\_hot\_sequences(sequences, dimension=1000):

results = numpy.zeros((len(sequences), dimension))

for i, word\_index in enumerate(sequences):

results[i, word\_index] = 1.

return results

train\_data = one\_hot\_sequences(train\_data)

test\_data = one\_hot\_sequences(test\_data)

model = tf.keras.Sequential()

model.add(tf.keras.layers.Dense(16, activation='relu', input\_shape=(1000,)))

model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Dense(16, activation='relu'))

model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam',

metrics=['accuracy'])

# 신경망 훈련, 검증 데이터 전달

history = model.fit(train\_data,

train\_labels,

epochs=20,

batch\_size=512,

validation\_data=(test\_data, test\_labels),

verbose=2)

history\_dict = history.history

loss\_values = history\_dict['loss'] # 훈련 데이터 손실값

val\_loss\_values = history\_dict['val\_loss'] # 검증 데이터 손실값

acc = history\_dict['accuracy'] # 정확도

epochs = range(1, len(acc) + 1) # 에포크 수

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

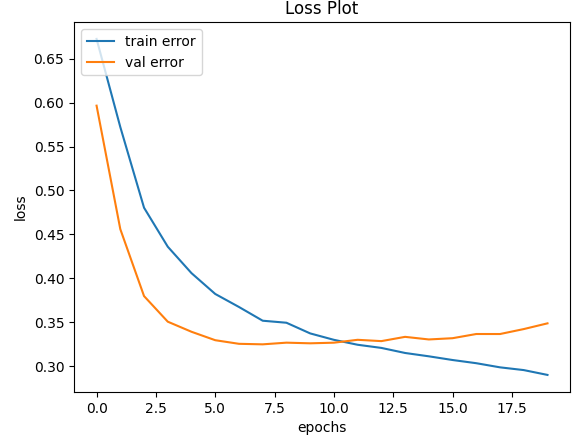
plt.title('Loss Plot')

plt.ylabel('loss')

plt.xlabel('epochs')

plt.legend(['train error', 'val error'], loc='upper left')

plt.show()

Epoch 1/20 49/49 - 1s - loss: 0.6725 - accuracy: 0.5696 - val\_loss: 0.5966 - val\_accuracy: 0.7648 - 1s/epoch - 21ms/step Epoch 2/20 49/49 - 0s - loss: 0.5722 - accuracy: 0.6999 - val\_loss: 0.4560 - val\_accuracy: 0.8286 - 292ms/epoch - 6ms/step Epoch 3/20 49/49 - 0s - loss: 0.4804 - accuracy: 0.7829 - val\_loss: 0.3798 - val\_accuracy: 0.8472 - 348ms/epoch - 7ms/step Epoch 4/20 49/49 - 0s - loss: 0.4360 - accuracy: 0.8122 - val\_loss: 0.3506 - val\_accuracy: 0.8534 - 358ms/epoch - 7ms/step Epoch 5/20 49/49 - 0s - loss: 0.4059 - accuracy: 0.8338 - val\_loss: 0.3392 - val\_accuracy: 0.8557 - 310ms/epoch - 6ms/step Epoch 6/20 49/49 - 0s - loss: 0.3822 - accuracy: 0.8438 - val\_loss: 0.3295 - val\_accuracy: 0.8571 - 341ms/epoch - 7ms/step Epoch 7/20 49/49 - 0s - loss: 0.3674 - accuracy: 0.8507 - val\_loss: 0.3254 - val\_accuracy: 0.8593 - 356ms/epoch - 7ms/step Epoch 8/20 49/49 - 0s - loss: 0.3517 - accuracy: 0.8596 - val\_loss: 0.3248 - val\_accuracy: 0.8578 - 215ms/epoch - 4ms/step Epoch 9/20 49/49 - 0s - loss: 0.3494 - accuracy: 0.8611 - val\_loss: 0.3267 - val\_accuracy: 0.8582 - 216ms/epoch - 4ms/step Epoch 10/20 49/49 - 0s - loss: 0.3373 - accuracy: 0.8677 - val\_loss: 0.3260 - val\_accuracy: 0.8591 - 286ms/epoch - 6ms/step Epoch 11/20 49/49 - 0s - loss: 0.3300 - accuracy: 0.8698 - val\_loss: 0.3267 - val\_accuracy: 0.8585 - 215ms/epoch - 4ms/step Epoch 12/20 49/49 - 0s - loss: 0.3244 - accuracy: 0.8740 - val\_loss: 0.3300 - val\_accuracy: 0.8590 - 298ms/epoch - 6ms/step Epoch 13/20 49/49 - 0s - loss: 0.3208 - accuracy: 0.8762 - val\_loss: 0.3285 - val\_accuracy: 0.8584 - 291ms/epoch - 6ms/step Epoch 14/20 49/49 - 0s - loss: 0.3150 - accuracy: 0.8768 - val\_loss: 0.3334 - val\_accuracy: 0.8579 - 205ms/epoch - 4ms/step Epoch 15/20 49/49 - 0s - loss: 0.3112 - accuracy: 0.8784 - val\_loss: 0.3304 - val\_accuracy: 0.8564 - 318ms/epoch - 6ms/step Epoch 16/20 49/49 - 0s - loss: 0.3070 - accuracy: 0.8796 - val\_loss: 0.3318 - val\_accuracy: 0.8574 - 225ms/epoch - 5ms/step Epoch 17/20 49/49 - 0s - loss: 0.3034 - accuracy: 0.8842 - val\_loss: 0.3366 - val\_accuracy: 0.8578 - 231ms/epoch - 5ms/step Epoch 18/20 49/49 - 0s - loss: 0.2987 - accuracy: 0.8852 - val\_loss: 0.3366 - val\_accuracy: 0.8562 - 290ms/epoch - 6ms/step Epoch 19/20 49/49 - 0s - loss: 0.2955 - accuracy: 0.8853 - val\_loss: 0.3422 - val\_accuracy: 0.8570 - 214ms/epoch - 4ms/step Epoch 20/20 49/49 - 0s - loss: 0.2900 - accuracy: 0.8898 - val\_loss: 0.3487 - val\_accuracy: 0.8564 - 292ms/epoch - 6ms/step

import matplotlib.pyplot as plt

import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x\_train, y\_train),(x\_test, y\_test) = mnist.load\_data()

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Flatten(input\_shape=(28,28)))

model.add(tf.keras.layers.Dense(512, activation='relu'))

model.add(tf.keras.layers.Dropout(0.2))

model.add(tf.keras.layers.Dense(10, activation='softmax'))

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

model.fit(x\_train, y\_train, epochs=5)

model.evaluate(x\_test, y\_test)

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>11490434/11490434 [==============================] - 0s 0us/step  
Epoch 1/5  
1875/1875 [==============================] - 8s 4ms/step - loss: 0.2196 - accuracy: 0.9348  
Epoch 2/5  
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0964 - accuracy: 0.9704  
Epoch 3/5  
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0689 - accuracy: 0.9783  
Epoch 4/5  
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0523 - accuracy: 0.9832  
Epoch 5/5  
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0439 - accuracy: 0.9858  
313/313 [==============================] - 1s 2ms/step - loss: 0.0661 - accuracy: 0.9816

[0.06613883376121521, 0.9815999865531921]

import tensorflow as tf

from tensorflow import keras

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras import datasets, layers, models

fashion\_mnist = keras.datasets.fashion\_mnist

(train\_images, train\_labels), (test\_images, test\_labels) = fashion\_mnist.load\_data()

plt.imshow(train\_images[0])

train\_images = train\_images / 255.0

test\_images = test\_images / 255.0

model = models.Sequential()

model.add(layers.Flatten(input\_shape=(28, 28)))

model.add(layers.Dense(128, activation='relu'))

model.add(layers.Dense(10, activation='softmax'))

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

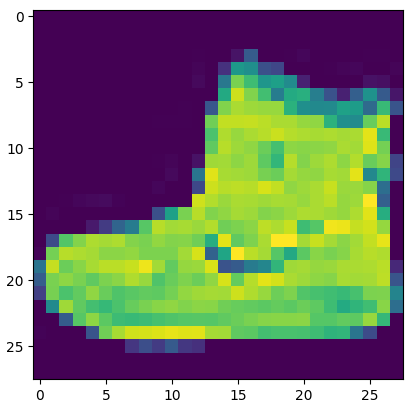
metrics=['accuracy'])

model.fit(train\_images, train\_labels, epochs=5)

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)

print('정확도:', test\_acc)

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz> 29515/29515 [==============================] - 0s 0us/step Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz> 26421880/26421880 [==============================] - 0s 0us/step Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz> 5148/5148 [==============================] - 0s 0us/step Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz> 4422102/4422102 [==============================] - 0s 0us/step Epoch 1/5 1875/1875 [==============================] - 5s 2ms/step - loss: 0.5009 - accuracy: 0.8238 Epoch 2/5 1875/1875 [==============================] - 4s 2ms/step - loss: 0.3731 - accuracy: 0.8651 Epoch 3/5 1875/1875 [==============================] - 4s 2ms/step - loss: 0.3347 - accuracy: 0.8785 Epoch 4/5 1875/1875 [==============================] - 4s 2ms/step - loss: 0.3140 - accuracy: 0.8861 Epoch 5/5 1875/1875 [==============================] - 4s 2ms/step - loss: 0.2939 - accuracy: 0.8915 313/313 [==============================] - 0s 1ms/step - loss: 0.3581 - accuracy: 0.8696 정확도: 0.8695999979972839



import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import tensorflow as tf

# 데이터 세트를 읽어들인다.

train = pd.read\_csv("train.csv", sep=',')

test = pd.read\_csv("test.csv", sep=',')

# 필요없는 컬럼을 삭제한다.

train.drop(['SibSp', 'Parch', 'Ticket', 'Embarked', 'Name',\

'Cabin', 'PassengerId', 'Fare', 'Age'], inplace=True, axis=1)

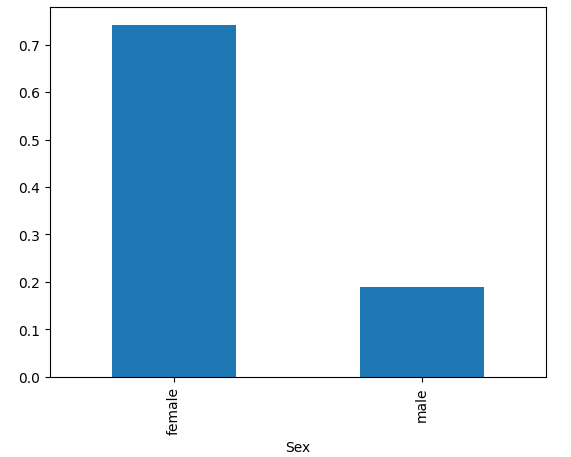
# 결손치가 있는 데이터 행은 삭제한다.

train.dropna(inplace=True)

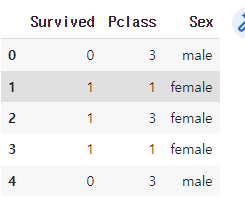
df = train.groupby('Sex').mean()["Survived"]

df.plot(kind='bar')

plt.show()



train.head()



for ix in train.index:

if train.loc[ix, 'Sex']=="male":

train.loc[ix, 'Sex']=1

else:

train.loc[ix, 'Sex']=0

# 2차원 배열을 1차원 배열로 평탄화한다.

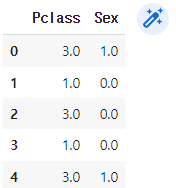
target = np.ravel(train.Survived)

# 생존여부를 학습 데이터에서 삭제한다.

train.drop(['Survived'], inplace=True, axis=1)

train = train.astype(float)

train.head()



model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(16, activation='relu', input\_shape=(2,)))

model.add(tf.keras.layers.Dense(8, activation='relu'))

model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

# 케라스 모델을 컴파일한다.

model.compile(loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

# 케라스 모델을 학습시킨다.

model.fit(train, target, epochs=30, batch\_size=1, verbose=1)

Epoch 1/30  
891/891 [==============================] - 2s 1ms/step - loss: 0.6048 - accuracy: 0.7385  
Epoch 2/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.5219 - accuracy: 0.7868  
Epoch 3/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.5006 - accuracy: 0.7868  
Epoch 4/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4902 - accuracy: 0.7868  
Epoch 5/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4844 - accuracy: 0.7868  
Epoch 6/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4776 - accuracy: 0.7868  
Epoch 7/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4738 - accuracy: 0.7868  
Epoch 8/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4691 - accuracy: 0.7868  
Epoch 9/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4668 - accuracy: 0.7834  
Epoch 10/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4643 - accuracy: 0.7845  
Epoch 11/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4603 - accuracy: 0.7868  
Epoch 12/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4595 - accuracy: 0.7800  
Epoch 13/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4583 - accuracy: 0.7789  
Epoch 14/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4591 - accuracy: 0.7789  
Epoch 15/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4558 - accuracy: 0.7879  
Epoch 16/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4549 - accuracy: 0.7946  
Epoch 17/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4564 - accuracy: 0.7845  
Epoch 18/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4558 - accuracy: 0.7778  
Epoch 19/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4530 - accuracy: 0.7946  
Epoch 20/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4554 - accuracy: 0.7856  
Epoch 21/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4536 - accuracy: 0.7901  
Epoch 22/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4555 - accuracy: 0.7901  
Epoch 23/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4553 - accuracy: 0.7778  
Epoch 24/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4550 - accuracy: 0.7856  
Epoch 25/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4546 - accuracy: 0.7811  
Epoch 26/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4554 - accuracy: 0.7834  
Epoch 27/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4540 - accuracy: 0.7901  
Epoch 28/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4553 - accuracy: 0.7868  
Epoch 29/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4539 - accuracy: 0.7868  
Epoch 30/30  
891/891 [==============================] - 1s 1ms/step - loss: 0.4541 - accuracy: 0.7890

<keras.callbacks.History at 0x7fbd1ee46980>

import numpy as np

import tensorflow as tf

model= tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(units = 8, input\_shape=(4,), activation='sigmoid'))

model.add(tf.keras.layers.Dense(units = 4, activation='sigmoid'))

model.compile(loss='mean\_squared\_error',optimizer='adam')

model.summary()

X=np.array([[1,1,1,1],[0,0,0,0],[1,0,1,0],[0,1,0,1],[1,0,0,1],[0,1,1,0],[0,0,1,1],[1,1,0,0]])

y=np.array([[1,0,0,0],[1,0,0,0],[0,1,0,0],[0,1,0,0],[0,0,1,0],[0,0,1,0],[0,0,0,1],[0,0,0,1]])

model.fit(X, y, batch\_size=2, epochs=10000)

print(model.predict(X))

Epoch 9984/10000 4/4 [==============================] - 0s 4ms/step - loss: 0.0225

Epoch 9985/10000 4/4 [==============================] - 0s 4ms/step - loss: 0.0225

Epoch 9986/10000 4/4 [==============================] - 0s 4ms/step - loss: 0.0225

Epoch 9987/10000 4/4 [==============================] - 0s 4ms/step - loss: 0.0224

Epoch 9988/10000 4/4 [==============================] - 0s 4ms/step - loss: 0.0224

Epoch 9989/10000 4/4 [==============================] - 0s 5ms/step - loss: 0.0224

Epoch 9990/10000 4/4 [==============================] - 0s 6ms/step - loss: 0.0225

Epoch 9991/10000 4/4 [==============================] - 0s 4ms/step - loss: 0.0224

Epoch 9992/10000 4/4 [==============================] - 0s 4ms/step - loss: 0.0224

Epoch 9993/10000 4/4 [==============================] - 0s 4ms/step - loss: 0.0224

Epoch 9994/10000 4/4 [==============================] - 0s 3ms/step - loss: 0.0224

Epoch 9995/10000 4/4 [==============================] - 0s 3ms/step - loss: 0.0224

Epoch 9996/10000 4/4 [==============================] - 0s 4ms/step - loss: 0.0224

Epoch 9997/10000 4/4 [==============================] - 0s 3ms/step - loss: 0.0224

Epoch 9998/10000 4/4 [==============================] - 0s 4ms/step - loss: 0.0224

Epoch 9999/10000 4/4 [==============================] - 0s 4ms/step - loss: 0.0224

Epoch 10000/10000 4/4 [==============================] - 0s 3ms/step - loss: 0.0224 1/1 [==============================] - 0s 147ms/step

[[6.6820693e-01 2.9704403e-04 3.2192037e-01 3.7581213e-02] [9.5482266e-01 6.8080085e-03 4.9363822e-02 2.1538399e-02] [6.9099784e-02 8.0246413e-01 2.2899997e-01 8.0721118e-02] [2.8610160e-04 9.4606131e-01 2.8974665e-02 2.8944066e-02] [3.4017745e-02 2.6229911e-03 9.3989664e-01 1.3039948e-04] [2.9731983e-01 1.8848903e-01 5.2574545e-01 1.6201718e-03] [1.1373084e-01 3.0850882e-02 6.0948979e-02 9.4178885e-01] [6.0766608e-02 3.1845894e-02 4.7203485e-02 9.2785770e-01]]

import tensorflow as tf

from tensorflow import keras

# 입력데이터

fashion\_mnist = keras.datasets.fashion\_mnist

(train\_images, train\_labels), (test\_images, test\_labels) = fashion\_mnist.load\_data()

# 데이터 정규화

train\_images = train\_images / 255.0

test\_images = test\_images / 255.0

# 데이터 평탄화

train\_images = train\_images.reshape((60000, 784))

test\_images = test\_images.reshape((10000, 784))

# 원핫 엔코딩

train\_labels = tf.keras.utils.to\_categorical(train\_labels)

test\_labels = tf.keras.utils.to\_categorical(test\_labels)

# DNN

# 모델 생성

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(512, activation='relu', input\_shape=(784,)))

model.add(tf.keras.layers.Dense(128, activation='relu'))

model.add(tf.keras.layers.Dense(10, activation='softmax')) # 출력층에 소프트 맥스 활성화 함수 사용

# 손실함수로 교차 엔트로피 사용

model.compile(optimizer='rmsprop',

loss='categorical\_crossentropy',

metrics=['accuracy'])

# 모델 학습

model.fit(train\_images, train\_labels, epochs=5, batch\_size = 128)

# 데스트

test\_loss\_d, test\_acc\_d = model.evaluate(test\_images, test\_labels)

# MLP

# 모델 생성

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(512, activation='relu', input\_shape=(784,)))

model.add(tf.keras.layers.Dense(10, activation='sigmoid'))

model.compile(optimizer='adam',

loss='mse',

metrics=['accuracy'])

# 모델 학습

model.fit(train\_images, train\_labels, epochs=5, batch\_size = 128)

# 데스트

test\_loss\_m, test\_acc\_m = model.evaluate(test\_images, test\_labels)

print('DNN 정확도:', test\_acc\_d)

print('MLP 정확도:', test\_acc\_m)

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz> 29515/29515 [==============================] - 0s 1us/step Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz> 26421880/26421880 [==============================] - 2s 0us/step Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz> 5148/5148 [==============================] - 0s 0us/step Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz> 4422102/4422102 [==============================] - 1s 0us/step Epoch 1/5 469/469 [==============================] - 9s 16ms/step - loss: 0.5483 - accuracy: 0.8001 Epoch 2/5 469/469 [==============================] - 6s 13ms/step - loss: 0.3805 - accuracy: 0.8598 Epoch 3/5 469/469 [==============================] - 5s 11ms/step - loss: 0.3364 - accuracy: 0.8753 Epoch 4/5 469/469 [==============================] - 6s 13ms/step - loss: 0.3096 - accuracy: 0.8844 Epoch 5/5 469/469 [==============================] - 5s 11ms/step - loss: 0.2899 - accuracy: 0.8912 313/313 [==============================] - 1s 3ms/step - loss: 0.3406 - accuracy: 0.8764 Epoch 1/5 469/469 [==============================] - 8s 15ms/step - loss: 0.0313 - accuracy: 0.7899 Epoch 2/5 469/469 [==============================] - 5s 11ms/step - loss: 0.0208 - accuracy: 0.8640 Epoch 3/5 469/469 [==============================] - 4s 9ms/step - loss: 0.0186 - accuracy: 0.8779 Epoch 4/5 469/469 [==============================] - 6s 12ms/step - loss: 0.0173 - accuracy: 0.8863 Epoch 5/5 469/469 [==============================] - 4s 9ms/step - loss: 0.0164 - accuracy: 0.8936 313/313 [==============================] - 1s 3ms/step - loss: 0.0180 - accuracy: 0.8805 DNN 정확도: 0.8763999938964844 MLP 정확도: 0.8805000185966492

from sklearn import datasets

from sklearn.model\_selection import train\_test\_split

import tensorflow as tf

classes = {0:'setosa', 1:'versicolor', 2:'virginica'}

iris = datasets.load\_iris()

X = iris.data

y = iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state = 1)

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(32, input\_shape = (4,), activation = 'relu'))

model.add(tf.keras.layers.Dense(32, activation='relu'))

model.add(tf.keras.layers.Dense(10, activation='softmax'))

model.compile(optimizer='rmsprop',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

model.fit(X\_train, y\_train, epochs = 10, batch\_size = 1)

test\_loss\_d, test\_acc\_d = model.evaluate(X\_test, y\_test)

y\_train = tf.keras.utils.to\_categorical(y\_train)

y\_test = tf.keras.utils.to\_categorical(y\_test)

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(32, input\_shape = (4,), activation = 'relu'))

model.add(tf.keras.layers.Dense(3, activation='sigmoid'))

model.compile(optimizer='rmsprop',

loss='mse',

metrics=['accuracy'])

model.fit(X\_train, y\_train, epochs = 10, batch\_size = 1)

test\_loss\_m, test\_acc\_m = model.evaluate(X\_test, y\_test)

print('DNN 정확도:', test\_acc\_d)

print('MLP 정확도:', test\_acc\_m)

Epoch 1/10 120/120 [==============================] - 2s 4ms/step - loss: 1.6732 - accuracy: 0.2583 Epoch 2/10 120/120 [==============================] - 1s 6ms/step - loss: 0.7309 - accuracy: 0.7333 Epoch 3/10 120/120 [==============================] - 1s 7ms/step - loss: 0.5247 - accuracy: 0.7167 Epoch 4/10 120/120 [==============================] - 2s 13ms/step - loss: 0.4294 - accuracy: 0.8083 Epoch 5/10 120/120 [==============================] - 0s 3ms/step - loss: 0.3659 - accuracy: 0.8333 Epoch 6/10 120/120 [==============================] - 0s 2ms/step - loss: 0.3308 - accuracy: 0.8583 Epoch 7/10 120/120 [==============================] - 0s 2ms/step - loss: 0.3005 - accuracy: 0.8917 Epoch 8/10 120/120 [==============================] - 0s 2ms/step - loss: 0.2732 - accuracy: 0.9167 Epoch 9/10 120/120 [==============================] - 0s 1ms/step - loss: 0.2456 - accuracy: 0.9500 Epoch 10/10 120/120 [==============================] - 0s 2ms/step - loss: 0.2234 - accuracy: 0.9333 1/1 [==============================] - 0s 140ms/step - loss: 0.1974 - accuracy: 1.0000 Epoch 1/10 120/120 [==============================] - 1s 1ms/step - loss: 0.3615 - accuracy: 0.3667 Epoch 2/10 120/120 [==============================] - 0s 1ms/step - loss: 0.2288 - accuracy: 0.3667 Epoch 3/10 120/120 [==============================] - 0s 2ms/step - loss: 0.1771 - accuracy: 0.5833 Epoch 4/10 120/120 [==============================] - 0s 2ms/step - loss: 0.1470 - accuracy: 0.7667 Epoch 5/10 120/120 [==============================] - 0s 2ms/step - loss: 0.1293 - accuracy: 0.7333 Epoch 6/10 120/120 [==============================] - 0s 1ms/step - loss: 0.1126 - accuracy: 0.7750 Epoch 7/10 120/120 [==============================] - 0s 1ms/step - loss: 0.1065 - accuracy: 0.8167 Epoch 8/10 120/120 [==============================] - 0s 2ms/step - loss: 0.1014 - accuracy: 0.8417 Epoch 9/10 120/120 [==============================] - 0s 1ms/step - loss: 0.0976 - accuracy: 0.7750 Epoch 10/10 120/120 [==============================] - 0s 2ms/step - loss: 0.0935 - accuracy: 0.8083 1/1 [==============================] - 0s 112ms/step - loss: 0.1092 - accuracy: 0.6667 DNN 정확도: 1.0 MLP 정확도: 0.6666666865348816

import pandas as pd

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

dataset = pd.read\_csv('winequality-red.csv', sep=';')

X = dataset.drop('quality', axis = 1)

y = dataset['quality']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state = 1)

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(512, activation='relu', input\_shape=(11,)))

model.add(tf.keras.layers.Dense(128, activation='relu'))

model.add(tf.keras.layers.Dense(10, activation='softmax'))

model.compile(optimizer='rmsprop',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

model.fit(X\_train, y\_train, epochs = 200, batch\_size = 32)

test\_loss, test\_acc = model.evaluate(X\_test, y\_test)

print(dataset['quality'].unique())

print('테스트 정확도:', test\_acc)

Epoch 191/200 35/35 [==============================] - 0s 5ms/step - loss: 0.6455 - accuracy: 0.7230 Epoch 192/200 35/35 [==============================] - 0s 4ms/step - loss: 0.6279 - accuracy: 0.7453 Epoch 193/200 35/35 [==============================] - 0s 3ms/step - loss: 0.6246 - accuracy: 0.7265 Epoch 194/200 35/35 [==============================] - 0s 3ms/step - loss: 0.6284 - accuracy: 0.7274 Epoch 195/200 35/35 [==============================] - 0s 3ms/step - loss: 0.6259 - accuracy: 0.7248 Epoch 196/200 35/35 [==============================] - 0s 3ms/step - loss: 0.6293 - accuracy: 0.7346 Epoch 197/200 35/35 [==============================] - 0s 3ms/step - loss: 0.6254 - accuracy: 0.7382 Epoch 198/200 35/35 [==============================] - 0s 4ms/step - loss: 0.6282 - accuracy: 0.7292 Epoch 199/200 35/35 [==============================] - 0s 3ms/step - loss: 0.6176 - accuracy: 0.7444 Epoch 200/200 35/35 [==============================] - 0s 3ms/step - loss: 0.6255 - accuracy: 0.7462 15/15 [==============================] - 0s 2ms/step - loss: 1.4051 - accuracy: 0.5979 [5 6 7 4 8 3] 테스트 정확도: 0.5979166626930237

import pandas as pd

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

dataset = pd.read\_csv('seeds\_dataset.txt', sep='\s+|\t', header = None)

X = dataset.drop(7, axis = 1)

y = dataset[7]

print(dataset[7].unique())

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state = 1)

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(32, activation='relu', input\_shape=(7,)))

model.add(tf.keras.layers.Dense(32, activation='relu'))

model.add(tf.keras.layers.Dense(4, activation='softmax'))

model.compile(optimizer='rmsprop',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

model.fit(X\_train, y\_train, epochs = 100, batch\_size = 32)

test\_loss, test\_acc = model.evaluate(X\_test, y\_test)

print(dataset[7].unique())

print('테스트 정확도:', test\_acc)

Epoch 90/100 5/5 [==============================] - 0s 3ms/step - loss: 0.2304 - accuracy: 0.9320 Epoch 91/100 5/5 [==============================] - 0s 3ms/step - loss: 0.2462 - accuracy: 0.9252 Epoch 92/100 5/5 [==============================] - 0s 3ms/step - loss: 0.2304 - accuracy: 0.8912 Epoch 93/100 5/5 [==============================] - 0s 3ms/step - loss: 0.2424 - accuracy: 0.9116 Epoch 94/100 5/5 [==============================] - 0s 3ms/step - loss: 0.2369 - accuracy: 0.9116 Epoch 95/100 5/5 [==============================] - 0s 4ms/step - loss: 0.2382 - accuracy: 0.9048 Epoch 96/100 5/5 [==============================] - 0s 3ms/step - loss: 0.2365 - accuracy: 0.9048 Epoch 97/100 5/5 [==============================] - 0s 4ms/step - loss: 0.2445 - accuracy: 0.9048 Epoch 98/100 5/5 [==============================] - 0s 3ms/step - loss: 0.2284 - accuracy: 0.8980 Epoch 99/100 5/5 [==============================] - 0s 3ms/step - loss: 0.2395 - accuracy: 0.9048 Epoch 100/100 5/5 [==============================] - 0s 3ms/step - loss: 0.2256 - accuracy: 0.9048 2/2 [==============================] - 0s 9ms/step - loss: 0.3309 - accuracy: 0.8730 [1 2 3] 테스트 정확도: 0.8730158805847168